Дж. Ортега

Введение в параллельные и векторные методы решения линейных систем

Дж. Ортега Введение в параллельные и векторные методы решения линейных систем

Введение в параллельные и векторные методы решения линейных систем

Introduction to Parallel and Vector Solution of Linear Systems

James M. Ortega
University of Virginia
Charlottesville, Virginia

Plenum Press New York and London

Дж. Ортега

Введение в параллельные и векторные методы решения линейных систем

Перевод с английского
Х. Д. Икрамова и И. Е. Капорина
под редакцией Х. Д. Икрамова

Москва «Мир» 1991

ББК 22.143
О70
УДК 519.852.6

Ортега Дж.

О70 Введение в параллельные и векторные методы решения линейных систем: Пер. с англ. — М.: Мир, 1991. — 367 с., ил.

ISBN 5-03-001941-3

Книга известного американского математика, знакомого советским читателям по его совместной с В. Рейнболдтом книге «Итерационные методы решения нелинейных систем уравнений со многими неизвестными» (М.: Мир, 1975) и по совместной с У. Пулом книге «Введение в численные методы решения дифференциальных уравнений» (М.: Наука, 1986). Его новая книга представляет собой один из первых в мировой литературе учебников по методам решения линейных систем на современных суперкомпьютерах.

Для математиков-прикладников, специалистов в области разработки математического обеспечения, аспирантов и студентов вузов.

О $\frac{2404010000\text{-}065}{041(01)\text{-}91}$ 127-91 **ББК 22.143**

Редакция литературы по математическим наукам

ISBN 5-03-001941-3 (русск.)
ISBN 0-306-42862-8 (англ.)

Предисловие переводчиков

Решение больших систем линейных уравнений принадлежит к кругу задач, для которых применение новейшей высокопроизводительной вычислительной техники дает максимальную выгоду. По этой причине большинство публикаций последнего десятилетия по численным методам линейной алгебры — их количество в 80-е годы значительно увеличилось — относится к области «параллельных вычислений» (неточный термин, под которым подразумеваются вопросы реализации методов на современных суперкомпьютерах, к какому бы классу архитектур — параллельных, векторных, систолических и т. п. — они ни причислялись).

Хотя существует немало книг, полностью или в значительной части посвященных параллельным вычислениям для задач линейной алгебры, это по преимуществу сборники трудов конференций. Имеются, правда, и монографии (упомянем, например, книги [Валях Е. Последовательно-параллельные вычисления. — М.: Мир, 1985] и [Воеводин В. В. Математические модели и методы в параллельных процессах. — М.: Наука, 1986]), но они ни в коей мере не являются учебниками. Да и трудно было до поры до времени ожидать появления учебника по предмету, еще столь неустоявшемуся, столь зависимому от стремительно изменяющейся технологии.

Однако в последние годы стали появляться книги учебного характера и в области параллельных линейно-алгебраических вычислений. Книга профессора Джеймса Ортеги — одна из первых публикаций такого рода. В ней сделана попытка на элементарном (как говорит само название книги) уровне объяснить и систематизировать сложившиеся принципы организации решения линейных систем на векторных и параллельных компьютерах. Эта попытка, на наш взгляд, вполне удалась. Особенно содержательна в книге третья глава, трактующая итерационные методы.

Не следует думать, что книга представляет интерес только для новичка. Библиографические разделы, завершающие каждый параграф, несут немало информации и для специалистов.

Книги, написанные Ортегой, отличаются завидным педагогическим мастерством. Две из них известны советским читателям: фундаментальная монография [Ортега Дж., Рейнболдт В. Итерационные методы решения нелинейных систем уравнений со многими неизвестными. — М.: Мир, 1975] и учебник [Ортега Дж., Пул У. Введение в численные методы решения дифференциальных уравнений. — М.: Наука, 1986]. Не является исключением и настоящая книга, родившаяся из обкатанного на протяжении многих лет курса лекций.

С согласия автора для русского перевода книги И. Е. Капориным было подготовлено дополнение к § 3.3, 3.4, названное «О предобусловливании и распараллеливании метода сопряженных градиентов». В нем обсуждаются последние результаты, относящиеся к одному из наиболее перспективных и быстро развивающихся направлений в численных методах решения больших систем уравнений на суперкомпьютерах.

При подготовке перевода работа была распределена так: И. Е. Капорин перевел третью главу и приложения 2—4; остальное перевел Х. Д. Икрамов.

Мы благодарим Л. Н. Королева за то, что он способствовал изданию этой книги, и Ю. С. Осипова за консультацию по терминологии.

Х. Д. Икрамов
И. Е. Капорин

Посвящается Саре и Скотту

Предисловие

Хотя истоки теории параллельных вычислений восходят к прошлому столетию, параллельные и векторные компьютеры стали доступны научному миру только в 70-е годы. Эффект, произведенный первыми из этих машин — 64-процессорным компьютером ILLIAC IV и векторными компьютерами фирм Texas Instruments, Control Data Corporation, а затем Cray Research Corporation, — был в известной степени ограниченным. Машин было мало и доступ к ним имели главным образом сотрудники нескольких государственных лабораторий. Однако теперь слабый ручеек превратился в бурный поток. В настоящее время введены в эксплуатацию более 200 высокопроизводительных векторных компьютеров, причем не только в государственных лабораториях, но и в университетах, а также во многих промышленных учреждениях. Кроме того, организованные Национальным научным фондом суперкомпьютерные центры сделали большие векторные компьютеры широко доступными академическому сообществу. Наконец, многие компании освоили производство векторных компьютеров меньшей производительности, но с очень высоким коэффициентом «эффективность/стоимость».

Быстро прогрессирует и направление, связанное с параллелизацией. Наиболее мощные современные суперкомпьютеры состоят из нескольких параллельно работающих векторных процессоров. Хотя число процессоров в таких машинах все еще относительно невелико (не превышает 8), ожидается, что в ближайшем будущем оно возрастет (до 16 или даже 32). В то же время существует множество исследовательских проектов машин с сотнями, тысячами и еще большим количеством процессоров. Несколько компаний уже сейчас продают параллельные компьютеры, в которых число процессоров достигает сотен или даже десятков тысяч.

Вероятно, главной движущей силой развития векторных и параллельных компьютеров были потребности научных вычислений, а одной из важнейших задач в области научных вычислений является решение систем линейных уравнений. Даже для

последовательных компьютеров эта задача остается полем активной деятельности, что особенно верно в отношении итерационных методов. Однако появление параллельных и векторных компьютеров сделало необходимым переосмысление даже основных алгоритмов, и этот процесс все еще продолжается. Более того, мы переживаем наиболее бурный период в истории компьютерных архитектур. Несомненно, потребуется еще несколько лет, чтобы какой-то один тип параллельной архитектуры утвердился в качестве доминирующего. Не исключено, что такого явного доминирования не будет никогда.

Отсюда следует, что книга на подобную тему обречена устареть почти в тот самый момент, когда она сдается в набор. Мы пытались противостоять этой опасности, не привязывая книгу ни к каким конкретным машинам, хотя читатель заметит следы влияния относительно старых компьютеров CDC CYBER 205 и CRAY-1. Однако существует немало основных идей, фактически не зависящих от конкретной машины; можно предположить, что они сохранят свое значение даже при том, что основанные на них конкретные алгоритмы могут потребовать модификации. Именно эти идеи мы пытались подчеркнуть.

Эта книга возникла из читаемого с начала 80-х годов семестрового курса для аспирантов первого года. Первоначально курс был ориентирован главным образом на анализ итерационных методов, но определенное внимание уделялось и векторным компьютерам, особенно машине CYBER 205. Со временем добавился материал по параллельным компьютерам, однако для практических заданий использовались CYBER 205 и, несколько позже, CRAY X-MP. Это отражено в упражнениях, которые сильно смещены в сторону CYBER 205.

Книга организована следующим образом. В первой главе обсуждаются некоторые основные характеристики векторных и параллельных компьютеров, а также особенности работы с алгоритмами для таких машин. Многие основные понятия разъясняются на примере сравнительно простой задачи матричного умножения. Во второй главе рассматриваются прямые методы, включающие *LU*-разложение, разложение Холесского и ортогональные факторизации. Предполагается, что читатель прослушал по крайней мере вводный курс численного анализа и знаком с большинством названных методов. Поэтому акцент делается на их организации для векторных и параллельных компьютеров. Третья глава посвящена итерационным методам. Поскольку в большинстве руководств по основам численного анализа этот раздел освещен довольно поверхностно или вообще отсутствует, в данной главе отведено больше места изложению основных (не зависящих от компьютерной системы)

свойств методов. Кроме того, для тех, кто хотел бы познакомиться с математической теорией методов более подробно, даны два приложения, где собраны многие стандартные теоремы о сходимости и ряд других результатов. Для чтения книги, кроме упомянутого выше знания начал численных методов, требуются еще некоторый опыт программирования и осведомленность в линейной алгебре. Краткая сводка необходимого материала линейной алгебры дается в приложении 4.

Многие важные вопросы в книге не затронуты и, как уже говорилось, сравнительно мало внимания уделено алгоритмам для конкретных современных машин. Однако каждый параграф заканчивается разделом «Литература и дополнения», где кратко резюмируются работы по соответствующей тематике и даются библиографические ссылки. Мы надеемся, что это поможет читателю углубить свои познания в интересующих его вопросах. Ссылки имеют вид [Автор, год], например [Jones, 1985]; по ним можно найти нужные публикации в списке литературы.

Эту книгу нужно читать так же, как любую другую книгу по математике, а не как рецептурное руководство. Неполные программные сегменты приводятся лишь в качестве иллюстраций, а не как основа рабочих кодов. Читателю, желающему использовать программу решения линейных уравнений для конкретной параллельной или векторной машины, мы настоятельно советуем *не начинать* с материала этой книги; особенно это касается прямых методов. Лучше выяснить, какие программы для данной машины уже написаны; наибольшую ценность представляют программы пакета LINPACK.

Мы придерживаемся следующих соглашений. Векторы обозначаются строчными, а матрицы прописными буквами. При нумерации уравнений указываются глава и параграф; так, (3.2.4) означает четвертое по счету уравнение в параграфе 3.2. Таким же образом нумеруются теоремы и определения.

Я признателен Сандре Шифлет, Беверли Мартин и в особенности Б. Энн Тёрли за перепечатку рукописи и многим студентам, а также рецензентам за высказанные ими замечания.

Шарлоттсвилл, штат Виргиния *Джеймс М. Ортега*

Глава 1

Введение

1.1. Векторные и параллельные компьютеры

С начала 1970-х годов стали появляться компьютеры, состоящие из ряда параллельно работающих процессоров или имеющие аппаратно реализованные команды для оперирования с векторами. Компьютеры второго типа мы будем называть *векторными компьютерами* (или *процессорами*), компьютеры первого типа — *параллельными компьютерами* (или *процессорами*).

Векторные компьютеры

В основе векторных компьютеров лежит концепция *конвейеризации*, т. е. явного сегментирования арифметического устройства на отдельные части, каждая из которых выполняет свою подзадачу для пары операндов. Иллюстрация для случая операции сложения чисел с плавающей точкой приведена на рис. 1.1.1. В этом примере сумматор для чисел с плавающей точкой разделен на шесть секций; каждая из них реализует

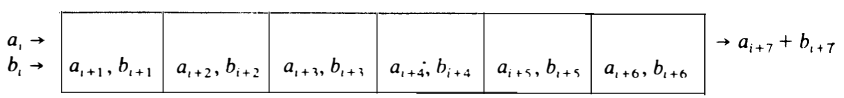


Рис. 1.1.1. Конвейер для сложения чисел с плавающей точкой

свою часть операции сложения. Всякий сегмент может работать только с одной парой операндов, а в целом на конвейере в данный момент времени могут находиться шесть пар операндов. Преимущество подобной сегментации в том, что результаты выдаются в 6 раз быстрее (а в общем случае в K раз, где K — число сегментов) по сравнению с арифметическим устройством, которое, получив пару операндов, не принимает новой пары до тех пор, пока не вычислит результат для первой пары. Однако для реализации этой возможности ускорения нужно подавать данные в арифметические устройства достаточно быстро, чтобы конвейер все время был загружен. С этим связано то обстоятельство, что аппаратная команда, например,

для операции сложения векторов устраняет необходимость в отдельных командах загрузки и запоминания данных. Одна и та же аппаратная команда управляет и загрузкой операндов, и запоминанием результатов.

Процессоры типа «память — память»

Фирма Control Data Corporation (CDC) выпустила ряд векторных процессоров, начиная с компьютера STAR-100 (1973 г.). Эволюция этой машины привела в конце 70-х годов к компьютеру CYBER 203, а затем, в начале 80-х годов, — к компьютеру CYBER 205. Все названные модели относятся к компьютерам

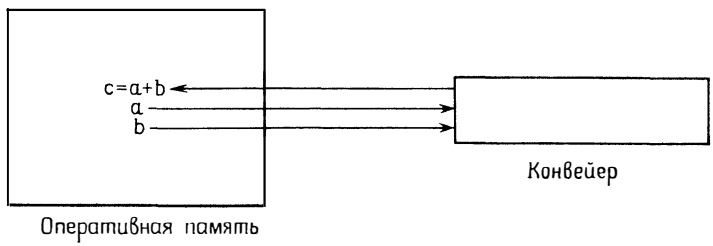


Рис. 1.1.2. Операция сложения в компьютере типа «память — память»

типа «память — память»; под этим подразумевается, что операнды векторных команд выбираются непосредственно из оперативной памяти и результат также записывается в оперативную память. Для операции сложения векторов это показано на рис. 1.1.2.

Процессоры типа «регистр — регистр»

С середины 70-х годов фирма Cray Research, Inc. производит векторные компьютеры, которые могут служить примером процессоров типа «регистр — регистр». Под этим подразумевается, что векторные команды получают свои операнды из очень быстрой памяти, именуемой *векторными регистрами*, и запоминают результаты опять-таки в векторных регистрах. Для операции сложения векторов это показано на рис. 1.1.3; предполагается, что каждый векторный регистр состоит из некоторого числа слов. Например, в машинах CRAY имеется восемь векторных регистров, емкость каждого — 64 числа с плавающей точкой. Операнды для векторного сложения выбираются из двух векторных регистров, и результат также записывается в векторный регистр. До сложения векторные регистры должны быть загружены из оперативной памяти, а в некоторый момент

времени после того, как сложение закончено, вектор-результат нужно переписать из векторного регистра в оперативную память. Для повышения эффективности компьютеров этого типа требуется как можно более интенсивно использовать дан-

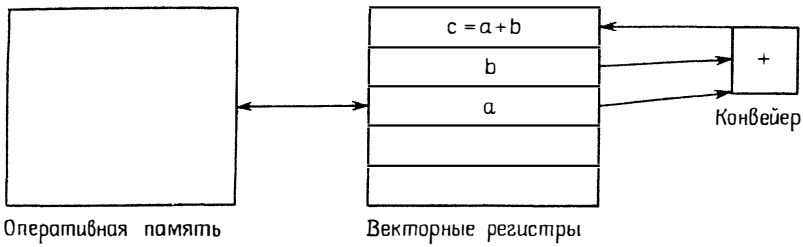


Рис. 1.1.3. Операция сложения в компьютере типа «регистр — регистр»

ные, пока они находятся в векторных регистрах. Несколько примеров такого использования будут даны в следующих параграфах.

Иерархии памяти

Векторные регистры играют роль, сходную с кэш-памятью обычных ЭВМ. Некоторые из современных векторных компьютеров имеют более сложную иерархию памяти. Например, в машине CRAY-2 каждому процессору помимо векторных регистров придана быстрая локальная память емкостью в 16 000 слов. Другие машины (скажем, компьютеры серии CRAY X-MP) снабжены массовой памятью, которая медленней оперативной, но значительно быстрее дисковой памяти. И, разумеется, у всех машин есть память на дисках. Важной задачей является использование различных типов хранения, обеспечивающее готовность данных всякий раз, как они нужны арифметическим устройствам.

Арифметические устройства

Как указано выше, для выполнения векторных операций векторные процессоры располагают конвейеризованными арифметическими устройствами. Однако конструкция этих устройств может быть различна. Машины фирмы CDC используют устройства *с изменяемой конфигурацией*, в которых один и тот же конвейер может выполнять различные арифметические операции. Однако до начала новой операции конвейер должен быть переналажен для нее. С другой стороны, машины CRAY обла-

дают раздельными конвейерами для сложения, умножения и некоторых других функций. У таких японских машин, как, например, NEC SX-2, имеется несколько конвейеров для операций одного типа (скажем, четыре конвейера для сложения, четыре для умножения и т. д.). Компьютер может иметь также несколько конвейеров с изменяемой конфигурацией. К примеру, CYBER 205 допускает 1, 2 или 4 конвейера, используемых в унисон для выполнения данной векторной операции (а не нескольких различных операций).

В составе аппаратно реализованных векторных операций всегда предусматриваются сложение и покомпонентное умножение двух векторов, а также либо покомпонентное деление

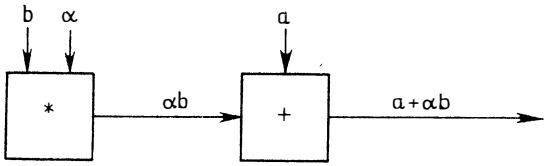


Рис. 1.1.4. Зацепление

векторов, либо формирование вектора из чисел, обратных к компонентам данного вектора. Могут иметься также векторные команды для более сложных операций: покомпонентного извлечения квадратного корня, скалярного произведения двух векторов, манипулирования с разреженными векторами и т. д. Некоторые операции можно реализовать весьма эффективно. Назовем *триадой* (в оригинале linked triad, т. е. буквально «сцепленная триада». — *Перев.*) операцию вида $\mathbf{a}+\alpha\mathbf{b}$, где $\mathbf{a}$ и $\mathbf{b}$ — векторы, а α — скаляр. Возможны и другие разновидности триады, например $(\mathbf{a}+\alpha)\mathbf{b}$; здесь через $\mathbf{a}+\alpha$ обозначен вектор, получаемый из $\mathbf{a}$ добавлением числа α к каждой компоненте. Компьютер CYBER 205 может вычислять триады почти с такой же скоростью, как суммы или произведения векторов. Триаду называют также операцией saxpy [1]); это название, пожалуй, более распространено, особенно среди пользователей компьютеров CRAY. Однако мы будем придерживаться термина «триада».

Машины с раздельными арифметическими конвейерами обычно допускают *зацепление* арифметических устройств. Это

[1]) Название saxpy для операции $y := ax + y$ впервые было употреблено авторами пакета BLAS [Lawson Ch. R. et al. ACM Transactions on Mathematical Software, 1979, v. 5, p. 308–323]; его первый символ указывает, что вычисления проводятся с обычной точностью (Single precision), все остальное есть символическое обозначение правой части оператора присваивания (P от plus (плюс)). — *Прим. перев.*

означает, что результаты, вычисляемые одним устройством, передаются другому без промежуточного возврата в регистр. Иллюстрация для случая триады приведена на рис. 1.1.4.

Большинство векторных компьютеров имеют отдельные устройства для скалярной арифметики. Эти устройства также могут быть конвейеризованы, но, в отличие от векторных конвейеров, не допускают векторных операндов. Они могут работать параллельно с векторными конвейерами и, по сравнению с максимальной скоростью последних, выдают результаты в 5—10 раз медленнее.

Время запуска

Использование векторных операций связано с накладными расходами, как показывает следующая приближенная формула для времени T векторной операции:

$$T = S + KN. \tag{1.1.1}$$

В этой формуле N — длина обрабатываемых векторов, K — промежуток времени, за который конвейер выдает результат, а S — *время запуска* конвейера. S есть время, необходимое для заполнения конвейера, включая время для подготовки операндов. Обычно S гораздо больше для машин типа «память — память», чем для машин типа «регистр — регистр», при условии, что во втором случае не учитывается время загрузки векторных регистров из оперативной памяти. В машинах с конвейерами изменяемой конфигурации S включает в себя время настройки конвейера.

Скорость выдачи результатов K тесно связана с *временем цикла* машины (называемым еще *тактовым периодом* или просто *тактом*). После заполнения конвейера каждый очередной результат выдается через такт. Следовательно, для многих машин K просто совпадает с временем цикла. Однако для машин с несколькими конвейерами K равно времени цикла, деленному на число конвейеров. Например, для компьютера CYBER 205 время цикла составляет 20 нс (нс $= 10^{-9}$ с), поэтому для двухконвейерной модели K равно 10 нс (сложение и покомпонентное умножение векторов), а для четырехконвейерной — 5 нс. Для машины NEC SX-2 время цикла составляет 6 нс и имеется по четыре конвейера для сложения и умножения. Следовательно, для этих операций $K = 1.5$ нс. Для более сложных операций (квадратный корень или скалярное произведение) значение K может быть больше.

Из формулы (1.1.1) следует, что (среднее) время получения одного результата равно

$$T_R = K + S/N. \tag{1.1.2}$$

График этой величины как функции от длины вектора N приведен на рис. 1.1.5. Этот график иллюстрирует необходимость работы с достаточно длинными векторами, чтобы амортизировать по многим результатам издержки на запуск. В современных векторных процессорах время выдачи результата имеет порядок нескольких наносекунд, тогда как время запуска составляет от нескольких десятков наносекунд для машин типа «регистр—регистр» до нескольких сотен наносекунд на машинах типа «память—память».

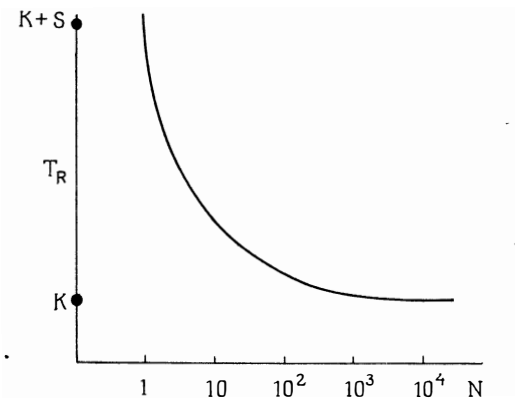


Рис. 1.1.5. График функции $T_R(N)$

Производительность конвейера можно характеризовать и числом результатов, выдаваемых им в единицу времени. Оно выражается формулой

$$R = T_R^{-1} = \frac{N}{S + KN}. \tag{1.1.3}$$

Если $S = 0$ или если $N \to \infty$, то из (1.1.3) следует

$$R_\infty = \frac{1}{K}. \tag{1.1.4}$$

Эта величина называется *асимптотической производительностью*. Она равна (недостижимой) максимальной скорости выдачи результатов, получаемой при игнорировании расходов на запуск. Пусть, например, $K = 10$ нс, тогда асимптотическая производительность составляет $R_\infty = 10^8$ результатов в секунду, другими словами, 100 мегафлопов (мегафлоп = 1 миллион операций с числами с плавающей точкой в секунду).

На рис. 1.1.6 изображена скорость выдачи результатов R как функция от N в предположении, что $K = 10$ нс, а S

принимает два значения: 100 нс и 1000 нс. Как видно из рисунка, с ростом N обе кривые стремятся к уровню асимптотической производительности, составляющему 100 мегафлопов. Однако расстояние до этого уровня для меньшего значения S поначалу намного меньше.

Некоторый интерес представляет величина $N_{1/2}$, определяемая как длина вектора, для которой достигается половина асимптотической производительности. Если, например, $K = 10$ нс, то из (1.1.3) следует, что $N_{1/2} = 100$ для $S = 1000$ и $N_{1/2} = 10$, если $S = 100$. Другой важной характеристикой

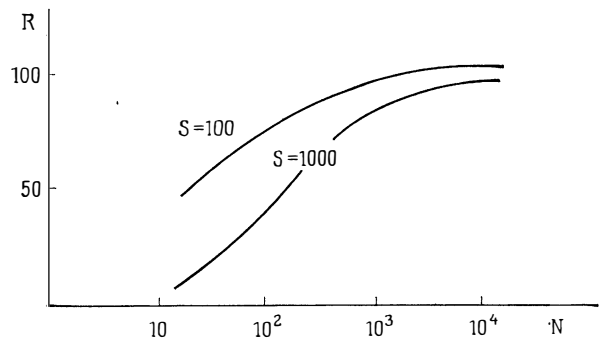


Рис. 1.1.6. График функции $R(N)$ (скорость выдачи результатов в мегафлопах)

является *граничная длина* N_c, для которой векторная арифметика сравнивается в скорости со скалярной. Предположим, что скалярную арифметику можно выполнять со средней скоростью 10 мегафлопов. Тогда граничная длина N_c — это значение N, для которого $R = 10$ мегафлопов. При $S = 1000$, пользуясь формулой (1.1.3), находим минимальное значение N из неравенства

$$\frac{N}{(1000 + 10N)\ 10^{-9}} \geqslant 10 \times 10^6.$$

Получаем $N_c = 12$. Таким образом, для векторов длины меньше 12 векторная арифметика медленнее скалярной. С другой стороны, при $S = 100$ имеем $N_c = 2$; здесь векторные операции более эффективны для всех векторных длин, за исключением тривиального случая векторов длины 1. Заметим, что значение граничной длины весьма чувствительно к скорости скалярной арифметики. Если бы в предыдущих примерах скорость скалярной арифметики равнялась только 5 мегафлопам, то было бы $N_c = 6$ при $S = 1000$ и $N_c = 1$ при $S = 100$.

Векторы

В векторных компьютерах накладываются ограничения на то, что следует понимать под вектором при выполнении векторных арифметических операций. В машинах типа «регистр — регистр» вектор для арифметической команды представляет собой последовательность смежных элементов векторного регистра, обычно начинающуюся с первого элемента этого регистра. Таким образом, для машин этого типа важно, что представляет собой вектор в оперативной памяти при загрузке векторного регистра. По существу это же самое важно и для реализации векторных арифметических операций в компьютерах типа «память — память».

Последовательно адресуемые элементы всегда составляют допустимый вектор; в некоторых машинах (скажем, машинах фирмы CDC) это единственный допустимый вид вектора. В дальнейшем мы будем пользоваться словом «смежные» как синонимом слова «последовательно адресуемые», несмотря на то, что последовательно адресуемые элементы физически обычно не соседствуют в памяти; как правило, они хранятся в различных банках памяти. (Однако в машинах фирмы CDC доступ к данным осуществляется посредством «суперслов», представляющих собой физически смежные восьмерки слов. Здесь к разным банкам памяти относятся суперслова.) В других машинах векторы образованы последовательностями элементов, имеющими постоянный шаг. Под *шагом* понимается адресное расстояние между соседними элементами последовательности. Так, элементы с адресами a, $a + s$, $a + 2s$, ... имеют постоянный шаг, равный s. В частном случае $s = 1$ получаем последовательно адресуемые элементы.

Предположим, что последовательность элементов не имеет постоянного шага или имеет постоянный шаг, бóльший 1, в то время как в данной машине вектор может состоять только из последовательно адресуемых элементов. Тогда, чтобы получить допустимый вектор, необходимо использовать дополнительное оборудование или программно реализованные команды переформатирования данных. Операция *сборки* отображает заданное число элементов, указываемых списком адресов, в вектор. Операция *слияния* объединяет два вектора в один. Операция *сжатия* отображает в вектор последовательность равноудаленных элементов. Разумеется, все эти операции требуют определенного времени, которое увеличивает накладные расходы векторных арифметических операций. Кроме того, по завершении векторной арифметики может понадобиться запомнить результаты не в виде вектора. Операция *рассылки*, обратная

к операции сборки, записывает элементы вектора в позиции, определяемые сопутствующим списком адресов. Главной заботой при конструировании алгоритмов для векторных компьютеров является организация данных, позволяющая минимизировать накладные расходы, проистекающие из необходимости использовать перечисленные операции управления данными.

Параллельные компьютеры

В основе параллельного компьютера лежит идея использования для решения одной задачи нескольких процессоров, работающих сообща. Расчет делается на то, что если одному процессору для выполнения задачи требуется время t, то p процессоров смогут решить эту задачу за время t/p. Однако это идеальное ускорение удается получить лишь в очень специальных ситуациях, и нашей целью является построение алгоритмов, способных извлечь из наличия нескольких процессоров максимальную выгоду для данной задачи.

Параллельный компьютер может иметь очень простые процессоры, пригодные только для малых или ограниченных задач, а может иметь весьма мощные векторные процессоры. Наше обсуждение будет в основном посвящено случаю, когда отдельный процессор представляет собой полноценный последовательный процессор умеренной производительности, но некоторое внимание будет уделено и компьютерам с векторными процессорами.

МКМД- и ОКМД-машины

Первый важный вид дихотомии в параллельных системах относится к тому, как управляются процессоры. В системах типа ОКМД (один поток команд — много потоков данных) все процессоры находятся под управлением главного процессора, называемого *контроллером*, или *управляющим процессором*; в каждый данный момент времени все процессоры выполняют одну и ту же команду (или все простаивают.) Таким образом, один поток команд воздействует на многие потоки данных, проходящих через отдельные процессоры. Компьютером типа ОКМД был ILLIAC IV, первая большая параллельная система, завершенная в начале 70-х годов. К этому же типу относятся ICL DAP, английская серийная машина, введенная в эксплуатацию в 1977 г., Goodyear MPP, построенная в начале 80-х годов специально для NASA, и Connection Machine, серийная машина середины 80-х годов. Индивидуальные процессоры этих машин представляют собой сравнительно простые однобитовые устройства; их 4096 в DAP, 16484 в MPP и 64936 в

Connection Machine. Концептуально векторные компьютеры также можно включить в класс ОКМД-машин, если считать, что элементы вектора обрабатываются независимо под управлением аппаратной векторной команды.

Большинство параллельных компьютеров, построенных после ILLIAC IV, представляют собой системы типа МКМД (много потоков команд — много потоков данных). Здесь индивидуальные процессоры работают под управлением своих собственных программ, чем достигается бóльшая гибкость заданий, выполняемых процессорами в каждый данный момент. В то же время возникает проблема синхронизации. В системе типа ОКМД синхронизация отдельных процессоров осуществляется контроллером, но в МКМД-системах приходится использовать другие механизмы, чтобы обеспечить выполнение процессорами своих заданий в правильном порядке и с правильными данными. Мы вернемся к вопросу о синхронизации в последующих параграфах.

Разделяемая или локальная память

Другой важный вид дихотомии для параллельных систем — это тип памяти: *разделяемая* или *локальная*. Устройство системы с разделяемой памятью показано на рис. 1.1.7. Здесь

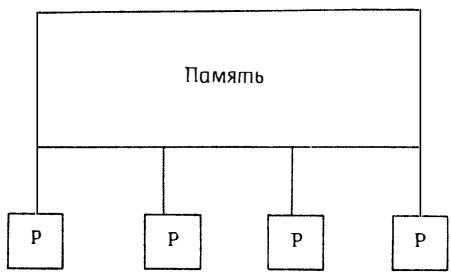


Рис. 1.1.7. Система с разделяемой памятью

все процессоры имеют доступ к общей памяти. (В дальнейшем мы будем пользоваться терминами «разделяемая память» и «общая память» попеременно). Каждый процессор может также иметь свою собственную локальную память для программ и промежуточных результатов. В этом случае общая память используется для данных и результатов, необходимых более чем одному процессору. Любое общение между индивидуальными процессорами идет через общую память. Главным достоинством систем с разделяемой памятью является потенциально очень

быстрое взаимодействие процессоров. Серьезный недостаток состоит в том, что общая память может потребоваться одновременно различным процессорам. В таких случаях возникают задержки с доступом к памяти; продолжительность подобных задержек, называемая *временем конфликтов* памяти, может расти с возрастанием числа процессоров.

Альтернативу системам с разделяемой памятью составляют системы с локальной памятью; в них каждый процессор может адресоваться только к собственной памяти. Обмены между процессорами происходят путем передачи сообщений, содержащих численную или иную информацию.

Схемы соединений

Вероятно, наиболее важным и интересным аспектом параллельных компьютеров является то, как общаются друг с другом отдельные процессоры. Это особенно важно для систем,

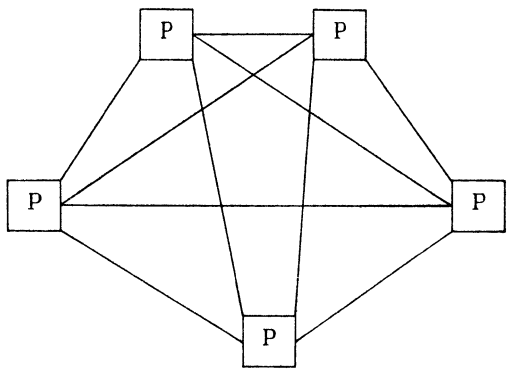


Рис. 1.1.8. Полностью связанная система

где процессоры имеют лишь локальную память, но немаловажно и для систем с разделяемой памятью, в которых соединения с общей памятью могут быть реализованы разными схемами. Кратко обсудим ряд наиболее распространенных схем взаимосвязей.

Полностью связанные системы. В полностью связанной системе каждый процессор имеет прямое соединение с любым другим процессором. Это показано на рис. 1.1.8. Полная связанность системы из p процессоров требует, чтобы из каждого процессора исходили $p - 1$ линий связи, что непрактично при большом p.

Коммутатор. Другим способом добиться полной связанности системы является *коммутатор*, изображенный на рис. 1.1.9. Как видно из рисунка, каждый процессор в принципе можно соединить с любым устройством памяти посредством переключателей, устанавливающих соединения. Достоинство этой схемы в том, что возможность доступа любого процессора к любому устройству памяти достигается с помощью небольшого числа

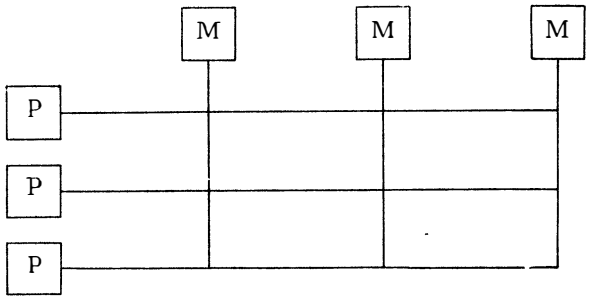


Рис. 1.1.9. Коммутатор

линий связи. Однако число переключателей, соединяющих p процессоров с p устройствами памяти, равно p^2, поэтому при больших p схема становится непрактичной. Эта схема использовалась для соединения 16 мини-компьютеров PDP-11 в одной из первых параллельных систем, а именно в системе C.mmp, разработке Университета Карнеги—Меллона (начало 70-х годов).

Шина и кольцо. Система с общей шиной показана на рис. 1.1.10. Здесь все процессоры соединены посредством (высокоскоростной) шины. Достоинством такой схемы является

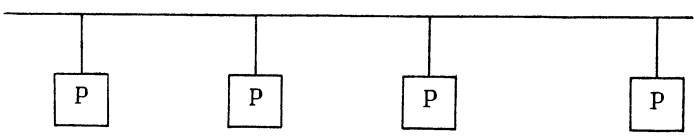


Рис. 1.1.10. Система с общей шиной

очень малое число линий связи, однако возможны задержки (*конфликт на шине*) при использовании шины несколькими процессорами. Это может стать серьезной проблемой при увеличении числа процессоров.

Кольцевая сеть — это схема с замкнутой шиной (см. рис. 1.1.11). Здесь данные перемещаются по кольцу и становятся

доступны процессорам по очереди. Схемы с общей шиной или кольцевые схемы различных типов применены в нескольких параллельных компьютерах. В некоторых системах шина используется для соединения процессоров с глобальной памятью. Примером может служить система FLEX/32 из 20 процессоров, созданная в середине 80-х годов фирмой Flexible Computer Corporation. В других случаях с помощью шины реализована система с локальной памятью и передачей сообщений. Примером является экспериментальная система ZMOB, разработанная в Университете штата Мэриленд в начале 80-х годов.

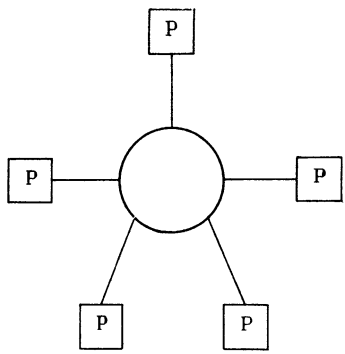


Рис. 1.1.11. Кольцевая связь

Решетка процессоров. Исторически сложилось так, что в число самых популярных схем соединения входит схема, в которой каждый процессор связан лишь с несколькими соседними. Простейшим примером является *линейный массив*, показанный на рис. 1.1.12. Здесь каждый процессор соединен с двумя соседями (за исключением концевых процессоров, имеющих лишь по одному соединению). Достоинство этой схемы — в ее простоте: из каждого процессора выходит не более двух линий

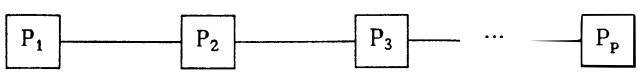


Рис. 1.1.12. Линейный массив

связи. Серьезный же недостаток состоит в том, что данные, прежде чем достичь своего конечного назначения, возможно, должны будут пройти через несколько промежуточных процессоров. (Заметим, что в этом отношении линейный массив отличается от системы с общей шиной, несмотря на сходство рисунков 1.1.10 и 1.1.12). Пусть, например (см. рис. 1.1.12), нужно переслать данные от процессора P_1 процессору P_p. В таком случае сначала следует послать их процессору P_2, затем передать процессору P_3, и т. д. Всего придется совершить $p - 1$ пересылок данных. Максимальное число пересылок, необходимых для переноса данных между любыми двумя процессорами системы, называется *коммуникационной длиной*, или *диа-*

метром системы. Кольцевой массив, изображенный на рис. 1.1.13, сокращает диаметр системы: максимальное расстояние между процессорами теперь составляет лишь около половины того, что было в случае линейного соединения. Связи в кольцевом массиве могут быть *однонаправленными* (скажем, на рис. 1.1.13 данные могут перемещаться только по часовой стрелке) или *двунаправленными*.

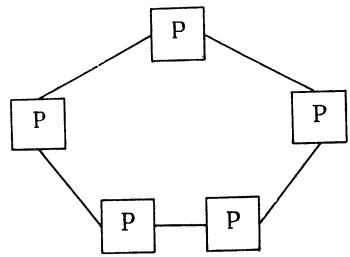


Рис. 1.1.13. Кольцевой массив

Большинство реально построенных решеточных массивов используют двумерные схемы соединений. Одна из простейших таких схем изображена на рис. 1.1.14. В ней процессоры расположены в виде правильной двумерной решетки и каждый процессор соединен с северным, южным, восточным и западным соседями. Кроме того, граничные процессоры могут быть соединены по принципу тора. Эта схема соединений «север-юг-восток-запад» была применена в компьютере ILLIAC IV, 64 процессора которого составляли массив 8×8.

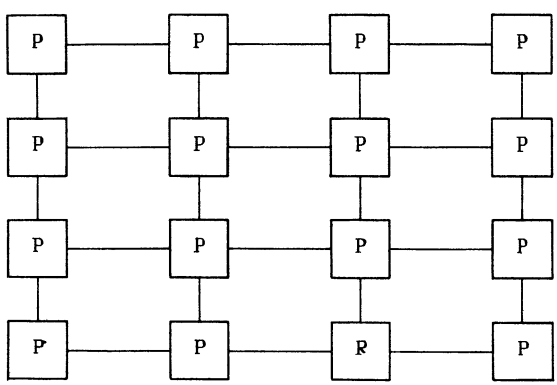


Рис. 1.1.14. Решетка процессоров

Достоинством схемы на рис. 1.1.14 опять-таки является простота. Недостаток, как и в случае линейного массива, состоит в том, что при обмене между отдаленными процессорами данные должны пройти через ряд промежуточных процессоров. Для p процессоров, расположенных в виде квадратного массива, диаметр системы составляет $O(\sqrt{p})$. Проблему обмена

данными до некоторой степени может облегчить введение дополнительных соединений, называемых *локальными связями*. Так, в схеме на рис. 1.1.14 можно соединить каждый процессор еще с четырьмя, находящимися от данного в северо-восточном, юго-восточном, юго-западном и северо-западном направлениях. Другая возможность: рассмотреть трехмерный массив процессоров, в котором каждый процессор соединен с шестью ближайшими соседями. Ясно, что добавление локальных связей уменьшает среднее время обмена, но это достигается ценой усложнения схемы.

Гиперкуб. Интересный вариант принципа локальных соединений получается, если мысленно погрузить схему в пространство большего числа измерений. Рассмотрим сперва трехмерную схему соединений, показанную на рис. 1.1.15. Здесь процессоры размещены в вершинах трехмерного куба, а ребра куба играют роль локальных связей между процессорами. Таким образом, каждый процессор соединен с тремя ближайшими соседями, т. е. ближайшими вершинами куба.

Теперь представим себе аналогичную схему соединений, но опирающуюся на куб в пространстве k измерений. Здесь процессоры также можно отождествить с 2^k вершинами этого k-мерного куба. Каждый процессор соединен вдоль ребер куба с k смежными вершинами. Такая схема соединений называется *гиперкубом*, или *двоичным k-кубом*. Разумеется, если $k > 3$, реально построить k-мерный куб нельзя, и для практической реализации этой конструкции ее следует отобразить в исходное пространство размерности $\leqslant 3$. Так, если предположить, что все процессоры на рис. 1.1.15 лежат в одной плоскости, то рисунок правильно указывает схему их соединения. В 4-кубе, т. е. в кубе четырехмерного пространства, имеется 16 процессоров, и каждый соединен с четырьмя другими. Можно считать, что решеточный массив на рис. 1.1.14 одновременно представляет собой изображение схемы соединений в 4-кубе. Рис. 1.1.16 иллюстрирует трехмерный способ визуализации схемы соединений для 4-куба; он показывает также, что 4-куб можно получить, связывая соответствующие вершины двух 3-кубов. В общем случае можно построить k-куб, соединяя соответствующие процессоры двух $(k-1)$-кубов.

В схеме гиперкуба число линий связи, выходящих из каждого процессора, возрастает по мере роста числа p самих процессоров и диаметр составляет лишь $\log p$. (Здесь и далее символ log обозначает двоичный логарифм). Так, для 64 процессоров (6-куб) каждый процессор соединен с шестью другими и диаметр равен 6, а для 1024 процессоров (10-куб) каждый процессор связан с десятью процессорами и диаметр равен 10.

Гиперкуб включает в себя некоторые из обсуждавшихся выше схем соединений. Например, игнорируя некоторые локальные связи, можно получить кольцевые или решеточные массивы.

По мере роста размерности гиперкуба растет не только число линий связи, выходящих из одного процессора, но и сложность схемы в целом. В конечном счете будет достигнут практический предел размерности гиперкуба.

Схема соединений в виде гиперкуба применена в нескольких вычислительных системах. В их число входит система

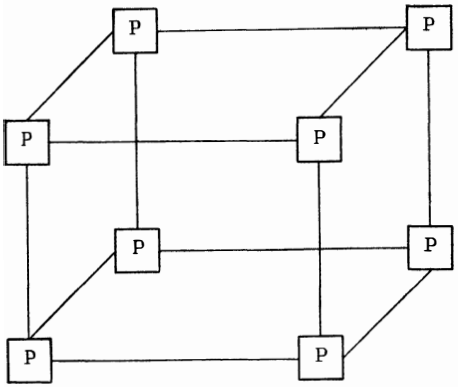


Рис. 1.1.15. Схема «3-куб»

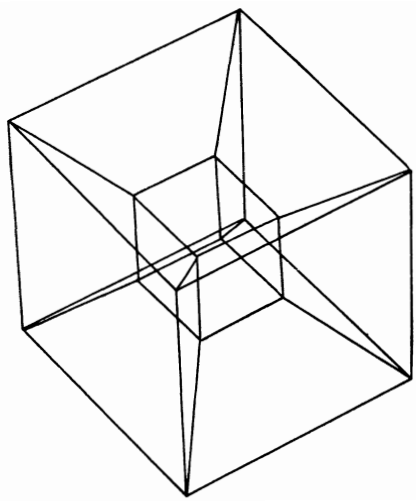


Рис. 1.1.16. Схема соединений в 4-кубе

Cosmic Cube, разработанная в начале 80-х годов в Калифорнийском технологическом институте. Ее концепции использованы в серийных системах фирм Intel Corp. (iPSC), Ncube Corp. и других.

Коммутационные сети. Довольно общий способ соединения разных процессоров или процессоров с устройствами памяти опирается на использование *коммутационных сетей.* (Коммутационные сети применяются для связи телефонных абонентов. Представьте себе задачу соединения всех телефонов страны посредством любой из рассмотренных ранее схем.) Простая коммутационная сеть (так называемая «бабочка». — *Перев.*) изображена на рис. 1.1.17. В левой части рисунка показаны восемь процессоров, в правой — восемь устройств памяти. Каждый прямоугольник представляет собой сдвоенный переключатель; отрезками указаны линии связи. Посредством

коммутационной сети каждый процессор может обмениваться с любым из устройств памяти. Пусть, например, процессору P_1 нужно получить доступ к памяти M_8. Тогда переключатель 1,1 устанавливается так, чтобы связать P_1 с переключателем 2,2, тот соединяется с переключателем (3,4), а последний — с памятью M_8.

В схеме на рис. 1.1.17 каждый процессор может произвести обмен с любым устройством памяти, поэтому здесь мы имеем систему с общей памятью, реализованную посредством коммутационной сети. Если бы устройства памяти в правой части ри-

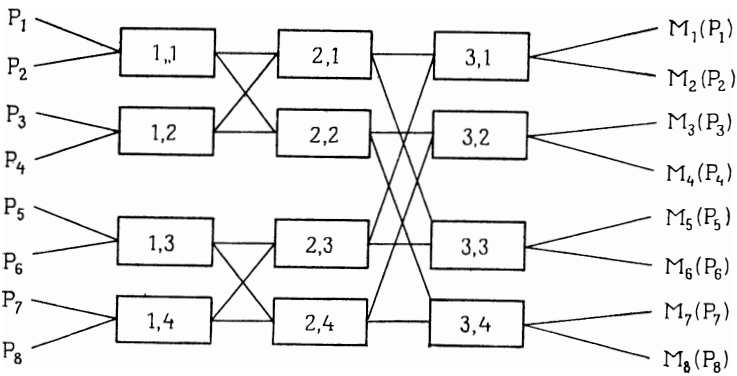


Рис. 1.1.17. Сеть типа «бабочка»

сунка были заменены процессорами (как указано в скобках), то тот же рисунок изображал бы систему с локальной памятью и передачей сообщений.

Коммутационная сеть, использующая, как на рис. 1.1.17, сдвоенные переключатели, потребовала бы $p/2$ переключателей (имеется в виду сеть, соединяющая p процессоров с p устройствами памяти или же с еще p процессорами. — *Перев.*) на каждом уровне переключения и $\log p$ уровней с общим числом $\frac{1}{2} p \log p$ переключателей. Это намного меньше, чем p^2 переключателей, необходимых для коммутатора на рис. 1.1.9; так, при $p = 2^{10}$ для коммутационной сети нужно только 5×2^{10} переключателей, а не 2^{20}.

Соединения по принципу коммутационной сети (не обязательно такой формы, как на рис. 1.1.17) использованы в нескольких построенных или спроектированных параллельных системах. Среди них системы Butterfly (фирма Bolt, Beranek and Newman, Inc.) и Ultracomputer (Нью-Йоркский университет.)

Гибридные схемы. Как следует из изложенного выше, каждая схема соединений имеет свои достоинства и недостатки. Это наводит на мысль о возможности комбинирования двух или более схем, позволяющего сочетать их лучшие качества и минимизировать недостатки. Предположим, например, что процессоры решетки на рис. 1.1.14 связаны еще и посредством шины. Тогда обмен между соседними процессорами может осуществляться через локальные связи, что освобождает шину от передачи этой части информации; обмен же между более отдаленными процессорами может происходить через шину. Гибридной системой этого типа был компьютер Finite Element Machine, сконструированный в конце 70-х годов в исследовательском центре NASA в г. Лэнгли. Другим примером может служить компьютер Connection Machine. В нем используются группы по 16 полностью связанных процессоров. Каждая такая группа вместе с памятью (своей для каждого процессора) реализована на одном кристалле. Между собой эти кристаллы соединены по принципу гиперкуба. Можно описать и ряд других гибридных схем.

Кластеры. Приемом, родственным гибридным схемам, является *кластеризация.* Кластерная схема иллюстрируется рисунком 1.1.18; здесь n кластеров и каждый из них состоит из

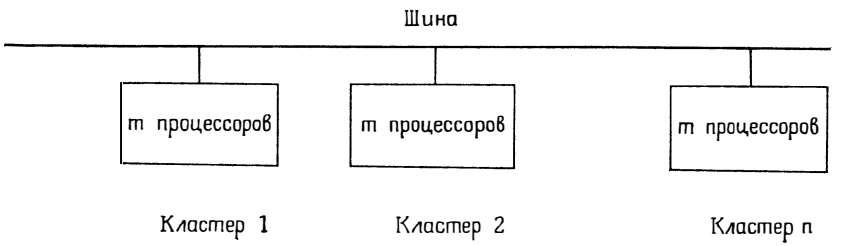


Рис. 1.1.18. Кластерная схема

m процессоров. В пределах кластера процессоры соединены некоторым образом (может использоваться любая из рассмотренных выше схем), а между собой кластеры связаны посредством шины. Соединения внутри кластеров называются *локальными*, а между кластерами — *глобальными.* В этой схеме расчет делается на возможность надлежащего сочетания этих двух типов взаимосвязей, т. е. на то, что для каждого процессора обмены будут происходить в основном внутри соответствующего кластера, потребность же в обменах между кластерами будет возникать менее часто.

Ясно, что кластерную схему можно реализовать разными способами. Внутри кластера связи можно организовать

посредством шины, кольца, решетки, гиперкуба и т. п. Любая из названных схем (а не только шина, как на рис. 1.1.18) может использоваться и для соединения кластеров. Отметим, что концепцию кластеризации можно применять рекурсивно, что приводит к кластерам из кластеров, и т. д. Компьютер Cm*, сконструированный в середине 70-х годов в Университете Карнеги—Меллона, был одной из первых систем, использовавших кластеризацию. Более современным примером может служить система CEDAR, разрабатываемая в Университете штата Иллинойс.

Схемы с изменяемой конфигурацией. Попытаться преодолеть ограничения, связанные с использованием фиксированной схемы взаимодействий, можно и так: придать схеме способность *изменять* шаблон соединений. Это можно сделать статически, до начала выполнения данной программы, или динамически, в ходе ее выполнения и под ее управлением. В начале 80-х годов были разработаны системы TRAC (Техасский университет в г. Остин) и Pringle (Университет Пэрдью и Вашингтонский университет), имевшие целью продемонстрировать полезность схем с изменяемой конфигурацией соединений.

Еще раз о связях

Теперь мы несколько более подробно обсудим взаимодействия в системах с локальной памятью. Рассмотрим типичную задачу о пересылке n чисел с плавающей точкой из памяти одного процессора P_1 в память другого процессора P_2. В общем случае такая пересылка реализуется посредством некоторой комбинации аппаратных средств и программного обеспечения, и ее стандартный сценарий может выглядеть следующим образом. Прежде всего данные загружаются в буферную память или собираются в последовательных ячейках памяти (процессора P_1. — *Перев.*). Затем выполняется команда *переслать*, результатом которой является перенос данных в буфер или память процессора P_2. Далее P_2 выполняет команду *принять*, и данные направляются на места своего конечного назначения в памяти P_2. Чтобы освободить основные процессоры от значительной части этой работы, можно использовать сопроцессоры.

В этом упрощенном описании опущены многие детали (в большей степени зависящие от конкретной системы), но главные пункты сохранены: чтобы переслать данные, их вначале нужно выбрать из памяти передающего процессора, должна быть предоставлена информация, куда следует их по-

слать, должен произойти физический перенос данных от одного процессора к другому и, наконец, данные нужно разместить в надлежащих ячейках памяти принимающего процессора. Для многих систем время такого обмена можно выразить приближенной формулой

$$t = s + \alpha n, \tag{1.1.5}$$

где s — время запуска, а α — время, необходимое для пересылки одного слова. Заметим, что вид этой формулы в точности совпадает с видом формулы (1.1.1) для времени исполнения векторной команды. В системах с локальной памятью, где обмены осуществляются посредством локальных связей между

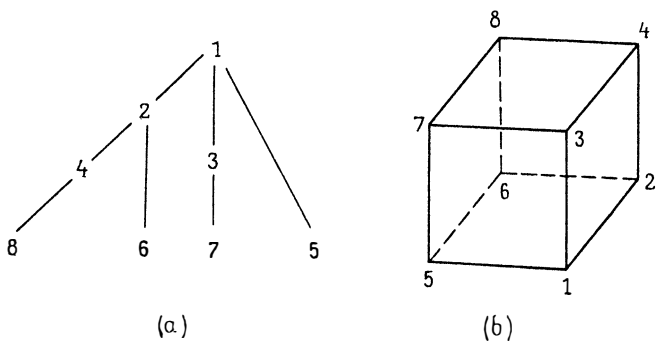


Рис. 1.1.19. Остовное дерево при распространении данных

соседними процессорами, пересылаемые данные, прежде чем достигнуть своего конечного назначения, возможно, должны будут пройти через ряд промежуточных процессоров. Поэтому, если формула (1.1.5) выражает время обмена между соседними процессорами, то общее время пересылки будет значительно большим.

Часто возникает необходимость в пересылке данных от одного процессора всем остальным; операция такого рода называется *распространением*. Мы укажем один из способов ее реализации для гиперкуба при условии, что в каждый данный момент времени любой процессор может общаться только с еще одним. На рис. 1.1.19a представлено *остовное дерево* (или минимальное остовное дерево) восьмипроцессорного гиперкуба; нумерация вершин последнего показана на рис. 1.1.19b. На первом временно́м шаге процессор 1 выполняет пересылку в процессор 2. На втором временно́м шаге происходит пересылка из процессора 1 в процессор 3 и из процессора 2 в процессор 4. На третьем временно́м шаге процессоры 1, 2, 3 и 4 совершают

пересылки соответственно в процессоры 5, 6, 7 и 8. Если имеется $p = 2^d$ процессоров, то потребуется $d = \log p$ временны́х шагов.

Параллельно-векторные системы

Мы закончим этот параграф кратким обсуждением параллельных систем, составленных из векторных компьютеров. Подобные параллельно-векторные системы предлагаются в настоящее время на рынке вычислительной техники, и такое положение, по всей видимости, сохранится в обозримом будущем. Серия CRAY X-MP была начата в 1982 г. двухпроцессорной машиной и продолжена в 1984 г. четырехпроцессорным вариантом. Предполагается, что в недалеком будущем число процессоров будет доведено до 16 (возможно, эта версия будет называться Y-MP). Система CRAY-2 имеет четыре процессора; число их, вероятно, будет увеличено. В 1987 г. введена в эксплуатацию восьмипроцессорная система ETA-10, преемница компьютеров серии CDC CYBER 200.

Выпускаются и параллельно-векторные системы с производительностью, не достигающей суперкомпьютерного уровня. Так, серийно выпускается гиперкуб Intel iPSC, построенный на векторных процессорах; фирма Alliant Corp. предлагает систему из восьми векторных процессоров.

Литература и дополнения к параграфу 1.1

1. Существует много книг и антологий, посвященных архитектуре векторных компьютеров В [Kogge, 1981] подробно обсуждается идея конвейеризации и ее применение в векторных компьютерах В книге [Hockney, Jesshope, 1981] описываются компьютеры CRAY-1 и CYBER 205, вводится длина $N_{1/2}$ половинной производительности и освещаются смежные вопросы. (Обобщение характеристики $N_{1/2}$ на параллельные машины предложено в статье [Hockney, 1987].) Среди сравнительно недавних книг — [Fernbach, 1986], где дается дополнительная информация о машинах CRAY, а также о японских векторных компьютерах, производимых фирмами Hitachi, Fujitsu и NEC, и [Stone, 1987], где рассматриваются конструкторские аспекты высокопроизводительных компьютеров По поводу систем CRAY X-MP см. публикации [Chen, 1984; Larson, 1984], а по поводу CYBER 205 — статью [Lincoln, 1982].

2. В ряде статей обсуждаются вопросы, представляющие специальный интерес для конкретных векторных компьютеров В системе CRAY X-MP загрузка регистров и запись в память могут быть сцеплены с векторными командами, и в работе [Dongarra, Hinds, 1985] отмечается, что при некоторых операциях X-MP действует как машина типа «память — память», причем векторные регистры играют роль буферов памяти В [Bucher, 1983] приводятся результаты измерений производительности для машин CRAY и других суперкомпьютеров. Статья [Hack, 1986] посвящена влиянию совмещения скалярных и векторных вычислений на общую производительность Результаты замеров производительности для гиперкубов Intel, Ncube и Ametak можно найти в [Dunigan, 1987].

3. Подробное обсуждение архитектуры параллельных компьютеров составляет содержание книги [Hwang, Briggs, 1984]. Различным аспектам этой темы посвящены также книги и антологии [Hockney, Jesshope, 1981; Snyder et al., 1985, Uhr, 1984; Paker, 1983] Архитектура конкретных машин рассматривается в статьях [Gottlieb et al., 1983; Seitz, 1985; Storaasli et al, 1982; Pfister et al., 1985] для машин New York University Ultracomputer, Cosmic Cube, Finite Element Machine, IBM RP3 соответственно. Классификация ОКМД и МКМД для параллельных компьютеров предложена в статье [Flynn, 1966].

4. Начиная приблизительно с 1983 г., на рынок был выпущен ряд серийных и относительно недорогих векторных и/или параллельных систем; их цены колеблются в пределах от 100 000 до 1 млн долларов, тогда как стоимость суперкомпьютера составляет примерно 10 млн. долларов. Среди этих систем векторный компьютер Convex C-1, восьмипроцессорная машина Alliant FX/8, машина FLEX/32 фирмы Flexible Computer Corp., компьютеры Encore и Sequent. Три последних представляют собой системы с разделяемой памятью и числом процессоров от 8 до 20 Некоторые данные о производительности системы Alliant FX/8 приведены в статье [Abu-Sufah, Malony, 1986].

5. Еще один подход к высокоскоростным вычислениям основывается на использовании присоединенных процессоров типа тех, что производятся фирмой Floating Point Systems, Inc. Присоединенный процессор, обычно употребляемый для участков интенсивного счета с хорошей векторизуемостью, работает под управлением основной машины, которой может быть универсальный компьютер или мини-ЭВМ Суперкомпьютерная система, построенная на основе присоединенных процессоров, описана в статье [Clementi et al., 1987]. Недавно фирма Floating Point Systems объявила о выпуске параллельной системы, в которой матричные процессоры соединены по схеме гиперкуба. Производительность 16-процессорной системы оценивается в 256 мегафлопов, что приближается к характеристикам суперкомпьютера

6. Много исследований посвящено двум другим подходам к параллельным вычислениям. В основе одного из них — понятие потока данных: вычисления управляются потоком данных и производятся тогда, когда необходимые данные получены Обзор этого направления дан в статье [Veen, 1986]. Второй подход связан с понятием систолического массива, представляющего собой решетку из (обычно малых) специализированных процессоров. Вычисления выполняются по мере того, как данные проходят сквозь массив. Хорошее обсуждение этой тематики можно найти в [Kung, 1982].

7. Организация распространения данных в гиперкубе, использующая понятие остовного дерева, рассматривается в статье [Geist, Heath, 1986]. Подробное обсуждение топологических свойств гиперкуба дано в техническом отчете [Saad, Schultz, 1985].

8 В ряде работ анализируются взаимодействия в параллельных системах. Так, в техническом отчете [Saad, Schultz, 1986] изучаются затраты на поддержание связи между процессорами для нескольких типов архитектуры, включая схему с общей шиной, кольцевую схему, двумерную решетку, гиперкуб и коммутационную сеть Основные операции переноса данных, рассматриваемые авторами, — это перенос данных от одного процессора к другому, от одного процессора ко всем остальным (распространение), от каждого процессора ко всем остальным процессорам (распространение из каждого процессора), рассылка данных от одного процессора ко всем остальным (или сбор данных из всех процессоров в данный процессор), мультирассылка (от каждого процессора ко всем остальным) или мультисборка. Авторы дают детальный вывод оценок затрат времени при различных предположениях

относительно архитектуры (например, относительно времени запуска, времени передачи и т п) и для различных алгоритмов переноса данных Сходные результаты содержатся в работах [Johnson, 1985a, 1987; Fox, Furmanski, 1987]. В них описаны алгоритмы обменов в гиперкубе, особенно полезные для матричных вычислений.

1.2. Основные понятия параллелизма и векторизации

В этом параграфе представлен ряд основных понятий и способов измерения параллелизма; они проиллюстрированы двумя очень простыми задачами. Мы начнем со случая параллельной системы, состоящей из p процессоров. Позднее будет рассмотрен вопрос о том, как обобщить на векторные компьютеры понятия, выработанные для параллельных. Дадим прежде всего следующее определение.

1.2.1. Определение. *Степенью параллелизма* численного алгоритма называется число его операций, которые можно выполнять параллельно.

Мы проиллюстрируем это определение несколькими примерами и по ходу дела уточним его.

Начнем с задачи сложения двух n-векторов **a** и **b**. Сложения

$$a_i + b_i, \quad i = 1, \ldots, n, \tag{1.2.1}$$

независимы и могут выполняться параллельно. Таким образом, степень параллелизма этого алгоритма равна n. Отметим, что понятие степени параллелизма не связано с числом процессоров нашей системы; оно является характеристикой параллелизма, внутренне присущего алгоритму. Разумеется, от числа процессоров зависит время, необходимое для завершения вычислений. Например, если $n = 1000$ и число процессоров p также равно 1000, то все суммы (1.2.1) можно вычислить за один временно́й шаг, однако при $p = 10$ потребуется 100 временны́х шагов.

Рассмотрим теперь задачу сложения n чисел $a_1, \ldots, a_n$. Обычный последовательный алгоритм

$$s = a_1, \quad s \leftarrow s + a_i, \ i = 2, \ldots, n, \tag{1.2.2}$$

непригоден для параллельных вычислений. Однако в самой задаче заключен немалый параллелизм. На рис. 1.2.1 показано, как можно осуществить суммирование восьми чисел в три этапа. На первом этапе параллельно выполняются четыре сложения, на втором — два, наконец, на последнем этапе — одно сложение. Это иллюстрирует общий принцип *разделяй и властвуй*. Задача суммирования разделена на меньшие подзадачи, которые могут решаться независимо.

Граф на рис. 1.2.1 называется *графом сдваивания* (в оригинале fan-in graph. — *Перев.*); он будет часто встречаться нам в дальнейшем. В частности, та же идея применима к вычислению произведения n чисел $a_1 a_2 \ldots a_n$, достаточно заменить на рис. 1.2.1 знак $+$ на знак $\times$. Точно так же, чтобы найти максимальное из n чисел $a_1, \ldots, a_n$, можно заменить знак операции сложения символом max; например, вместо $a_1 + a_2$ на рис. 1.2.1 стояло бы $\max(a_1, a_2)$, и т. д. Заметим, что граф на рис. 1.2.1 представляет собой *двоичное дерево*, поэтому операцию, выполняемую с помощью графа сдваивания, иногда называют *операцией на дереве* (в оригинале tree operation. — *Перев.*).

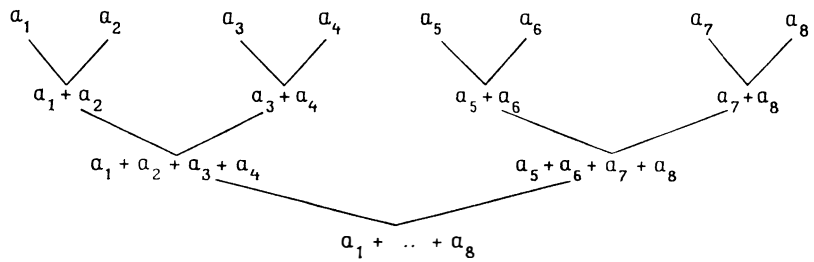


Рис. 1.2.1. Сложение методом сдваивания

Для $n = 2^q$ чисел алгоритм сдваивания состоит из $q = \log n$ этапов; на первом этапе выполняются $n/2$ сложений, на втором — $n/4$, и т. д., пока на последнем этапе не будет выполнено единственное сложение. Очевидно, что на первом этапе степень параллелизма равна $n/2$, на втором — $n/4$, и т. д. Это обстоятельство наводит на мысль о модификации определения 1.2.1 с целью получения меры параллелизма алгоритма, различные этапы которого имеют разную степень параллелизма.

1.2.2. Определение. *Средней степенью параллелизма* численного алгоритма называется отношение общего числа операций алгоритма к числу его этапов.

Для алгоритма сдваивания средняя степень параллелизма равна

$$\frac{1}{q}\left(\frac{n}{2} + \frac{n}{4} + \ldots + 1\right) = \frac{2^q - 1}{q} = \frac{n-1}{\log n} = O\left(\frac{n}{\log n}\right), \quad (1.2.3)$$

тогда как при сложении двух n-векторов средняя степень параллелизма равна n, т. е. совпадает со степенью параллелизма. Этот последний алгоритм обладает «идеальным» параллелизмом, в то время как для алгоритма сдваивания средняя степень параллелизма в $\log n$ раз меньше идеальной. И все же

алгоритм сдваивания можно считать хорошо параллелизуемым, особенно по сравнению с последовательным алгоритмом (1.2.2), для которого средняя степень параллелизма равна $(n-1)/(n-1)=1$.

Понятие *зернистости* связано со степенью параллелизма. *Крупнозернистость* задачи означает наличие в ней больших независимых подзадач, которые можно обрабатывать параллельно. Примером может служить задача решения шести различных больших систем линейных уравнений; решения этих систем комбинируются на более поздних стадиях вычислительного процесса. *Мелкозернистость* соответствует возможности параллельного выполнения малых подзадач; так, для сложения двух векторов подзадачей является сложение одноименных компонент.

Ускорение

Теперь мы введем две другие распространенные меры параллелизма.

1.2.3. Определение. *Ускорением* параллельного алгоритма называется отношение

$$S_p = \frac{\text{время выполнения алгоритма на одном процессоре}}{\text{время выполнения алгоритма в системе из } p \text{ процессоров}}. \quad (1.2.4)$$

Для задачи сложения векторов следовало бы ожидать, что $S_p = p$, т. е. ускорение максимально. Заметим однако, что в определении 1.2.3 подразумеваются действительные времена вычислений. Это делает определение более реалистичным, но затрудняет его использование в том случае, если требуемые времена неизвестны. Рассмотрим сложение n чисел с помощью алгоритма сдваивания в системе из $n/2$ процессоров с локальной памятью. Пусть a_1 и a_2 хранятся в памяти процессора 1, a_3 и a_4 — в памяти процессора 2, и т. д. Прежде чем можно будет выполнять сложения второго этапа, необходимо передать число a_3+a_4 процессору 1, число a_7+a_8 — процессору 3, и т. д. Аналогичные переносы данных требуются на каждом этапе, что, конечно, увеличивает общее время выполнения алгоритма. Предположим, что для одного сложения нужно время t, а для переноса одного числа — время αt; обычно α больше единицы. Игнорируя прочие издержки, имеем для алгоритма сложения (с учетом равенства $p=n/2$)

$$S_p = \frac{(n-1)t}{(\log n)(1+\alpha)t} = \frac{1}{1+\alpha}\,\frac{n-1}{\log n} = \frac{1}{1+\alpha}\,\frac{2p-1}{1+\log p}. \quad (1.2.5)$$

Заметим, что в данном случае ускорение есть средняя степень параллелизма, деленная на $1+\alpha$. Если α — величина, близкая

к единице, т. е. на обмены затрачивается примерно столько же времени, сколько на арифметику, то ускорение уменьшится приблизительно вдвое по сравнению с идеальной ситуацией, когда $\alpha = 0$. С другой стороны, если α велико, скажем, $\alpha = 10$, то время на обмены доминирует над временем вычислений; соответственно падает ускорение.

Ускорение S_p позволяет сравнить поведение данного алгоритма для одного и p процессоров. Однако, как мы не раз увидим в последующих главах, параллельный алгоритм может оказаться не лучшим выбором для последовательного компьютера. Поэтому более обоснованная мера выигрыша, получаемого при параллельных вычислениях, заключена в таком определении.

1.2.4. Определение. *Ускорением параллельного алгоритма по сравнению с наилучшим последовательным алгоритмом* называется отношение

$$S'_p = \frac{\text{время выполнения быстрейшего последовательного алгоритма на одном процессоре}}{\text{время выполнения параллельного алгоритма на системе из } p \text{ процессоров}}.$$

С ускорением связана эффективность параллельного алгоритма.

1.2.5. Определение. *Эффективностью* параллельного алгоритма называется величина

$$E_p = \frac{S_p}{p}.$$

Эффективностью параллельного алгоритма по отношению к наилучшему последовательному алгоритму называется величина

$$E'_p = \frac{S'_p}{p}.$$

Поскольку $S_p \leqslant p$ и $S'_p \leqslant S_p$, то $E'_p \leqslant E_p \leqslant 1$. Если алгоритм достигает максимального ускорения ($S_p = p$), то $E_p = 1$.

Одной из целей при конструировании параллельных алгоритмов является достижение по возможности большего ускорения; в идеальном случае $S'_p = p$. Однако мы уже видели на примере сложения n чисел, что этот идеал не всегда достижим. В самом деле, максимальное ускорение можно получить только для задач, по существу тривиальных. Главные факторы, обусловливающие отклонение от максимального ускорения, таковы:

1. Отсутствие максимального параллелизма в алгоритме и/или небалансированность нагрузки процессоров.

2. Обмены, конфликты памяти и время синхронизации.

Мы обсудим эти причины несколько более подробно, причем начнем со второй. Ранее были рассмотрены обмены между процессорами на примере задачи сложения n чисел. В общем случае в системе с локальной памятью обмен данными между процессорами будет необходим в разные моменты общего вычислительного процесса. В той степени, в какой процессоры не заняты полезной работой во время обменов, это составляет накладные расходы, которые мы пытаемся минимизировать. В системе с разделяемой памятью ту же роль играют конфликты памяти (см. § 1.1): процессоры простаивают или недогружены, пока ожидают данных, необходимых для продолжения счета.

Синхронизация необходима в тех случаях, когда некоторые отрезки вычислений должны быть закончены прежде, чем сможет возобновиться процесс в целом. В накладные расходы вносят вклад два аспекта синхронизации. Это, во-первых, время, нужное непосредственно для синхронизации; обычно синхронизация состоит в том, что каждый процессор выполняет некоторую проверку. Во-вторых, часть процессоров или даже почти все могут простаивать, ожидая разрешения на продолжение вычислений. В последующих главах мы рассмотрим несколько примеров синхронизации.

Хотя задержки, связанные с синхронизацией, обменами и конфликтами памяти, по своей природе весьма различны, их воздействие на общий процесс вычислений одинаково: они замедляют его на время, необходимое для подготовки данных, нужных для дальнейшего счета. Поэтому иногда мы будем объединять все три фактора задержки, как это сделано в следующем определении.

1.2.6. Определение. *Временем подготовки данных* называется задержка, вызванная обменами, конфликтами памяти или синхронизацией и необходимая для того, чтобы разместить данные, требующиеся для продолжения вычислений, в соответствующих ячейках памяти.

Обратимся теперь к фактору отсутствия максимального параллелизма. Он может проявляться по-разному. При сложении n чисел мы видели, что на первом этапе алгоритма параллелизм максимален, однако на каждом последующем этапе степень параллелизма уменьшается вдвое. В гл. 2 мы убедимся, что постепенное уменьшение степени параллелизма характерно для алгоритмов исключения при решении линейных систем.

Большинство алгоритмов представляют собой смесь фрагментов с тремя различными степенями параллелизма, которые мы будем называть максимальным, частичным и минимальным параллелизмом. Последний термин относится к тем фрагментам алгоритма, где может использоваться лишь один процессор. Но даже если алгоритм сам по себе обладает максимальным параллелизмом, ситуация для данной параллельной системы может осложняться проблемой балансировки нагрузки. Под *балансировкой нагрузки* понимается такое распределение заданий между процессорами системы, которое позволяет занять каждый процессор полезной работой по возможности бо́льшую часть времени. Это распределение называют иногда *задачей отображения* (мы хотим отобразить задачу и алгоритм на процессоры системы так, чтобы достичь максимального выравнивания нагрузки). Отметим, что внутренне алгоритму может быть присуща высокая степень параллелизма, однако балансировка нагрузки для конкретной системы может оказаться затруднительной. Например, при сложении векторов длины 9 в системе из восьми процессоров имеется несоответствие между идеальным параллелизмом задачи и системой, используемой для ее решения.

Балансировка нагрузки может осуществляться как статически, так и динамически. При *статической балансировке* задания (а в системах с локальной памятью, возможно, и данные) распределяются между процессорами до начала вычислений. При *динамической балансировке* задания (и данные) распределяются между процессорами в ходе вычислительного процесса. Полезным понятием, связанным с динамической балансировкой нагрузки, является понятие *банка заданий* (в оригинале pool of tasks — *Перев.*). Процессор получает из банка очередное задание, когда готов к его выполнению. В общем случае динамическая балансировка нагрузки эффективнее реализуется в системах с разделяемой памятью, поскольку для систем с локальной памятью при распределении заданий может еще потребоваться обмен данными между процессорами.

Рассмотрим теперь формальную модель ускорения, в которой

$$S_p = \frac{T_1}{(\alpha_1 + \alpha_2 k + \alpha_3 p)T_1 + t_a}, \qquad (1.2.6)$$

где T_1 — время, затрачиваемое единственным процессором, α_1 — доля операций, выполняемых только одним процессором, α_2 — доля операций, производимых со средней степенью параллелизма $k < p$, α_3 — доля операций, производимых со степенью параллелизма p, и t_a — общее время, требуемое для подготовки

данных. Обсудим несколько специальных случаев формулы (1.2.6).

Случай 1. $\alpha_1 = \alpha_2 = 0$, $\alpha_3 = 1$, $t_d = 0$. Здесь $S_p = p$ и ускорение максимально. Предпосылки данного случая заключаются в том, что все операции выполняются с максимальным параллелизмом и задержки отсутствуют.

Случай 2. $\alpha_1 = \alpha_3 = 0$, $\alpha_2 = 1$, $t_d = 0$. Теперь $S_p = k < p$ и ускорение равно всего лишь средней степени параллелизма.

Случай 3. $\alpha_2 = 0$, $t_d = 0$, $\alpha_1 = \alpha$, $\alpha_3 = 1 - \alpha$. В этом случае

$$S_p = \frac{1}{\alpha + (1 - \alpha)/p}. \qquad (1.2.7)$$

Формула (1.2.7) выражает *закон Уэра*, или *закон Амдаля* (Ware, Amdahl. — *Перев.*). Предпосылки данного случая заключаются в том, что все операции производятся либо с максимальным, либо с минимальным параллелизмом, и задержки отсутствуют. Хотя формула (1.2.7) соответствует очень упрощенной модели, она все же поучительна. Предположим, что в данной задаче половина операций может выполняться параллельно, а половина нет. Тогда $\alpha = 1/2$ и (1.2.7) принимает вид

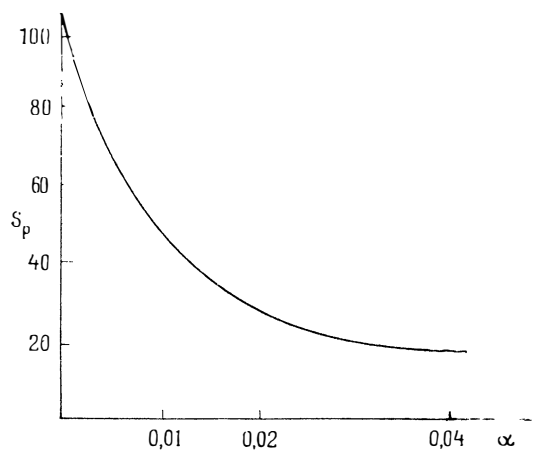


Рис. 1.2.2. Уменьшение ускорения

$$S_p = \frac{2}{1 + p^{-1}} < 2.$$

Таким образом, независимо от количества процессоров и даже при игнорировании всех затрат на обмены, синхронизацию и конфликты памяти ускорение всегда меньше 2. На рис. 1.2.2

приведен график величины S_p как функции от α при номинальном значении $p = 100$. Отметим очень быстрое убывание S_p для малых значений α. Если 1 % операций должен выполняться единственным процессором, то ускорение уменьшается вдвое: со 100 до 50.

Случай 4. Время t_d велико. Этот случай предназначен для иллюстрации следующего положения: каковы бы ни были значения α_1, α_2 и α_3, при достаточно большом t_d можно получить $S_p < 1$. Таким образом, вполне возможно, что для задачи с интенсивными обменами, многочисленными конфликтами памяти и большими издержками на синхронизацию использование нескольких процессоров оказывается менее выгодным, чем использование одного. Это, конечно, крайний случай, но во многих задачах достигается предел, за которым увеличение числа процессоров не оправдывает себя.

Вернемся к определениям ускорения 1.2.3 и 1.2.4. При их практическом применении возникает несколько трудностей. На параллельных машинах с локальной памятью можно решать тем бóльшие задачи, чем больше используется процессоров. В частности, задача, решаемая с большим числом процессоров, может не помещаться в оперативной памяти единственного процессора, поэтому к времени, затрачиваемому одним процессором, следовало бы добавить время на обмены с внешней памятью. В определенном смысле это вполне законная мера ускорения, но при некоторых рассмотрениях она может оказаться неадекватной. Еще одно родственное соображение: по-видимому, параллельные машины нужны только для тех задач, которые слишком велики, чтобы их можно было решить за разумное время с помощью единственного процессора. Отсюда вытекает, что исследование ускорения, достигаемого для данного алгоритма, должно проводиться для больших задач и, возможно, для относительно большого числа процессоров.

Подход к измерению ускорения, который потенциально может учесть обе названные проблемы, состоит в том, чтобы измерять скорость вычислений по мере того, как растут размер задачи и число процессоров. Рассмотрим, например, задачу матрично-векторного умножения $A\mathbf{x}$; более подробно она будет обсуждаться в следующем параграфе. Если A — $n \times n$-матрица, то потребуется $2n^2 - n$ операций (n^2 умножений и $n^2 - n$ сложений). Предположим, что при исходном порядке n вычисления на единственном процессоре идут со скоростью 1 мегафлоп. Затем мы удваиваем n, и число операций при большом n увеличивается примерно в 4 раза. Если задача решается на четырех процессорах со скоростью 4 мегафлопа, то мы достигли

максимального ускорения. Вообще, пусть размер задачи определяется параметром n, а число операций есть $f(n)$. Мы можем графически изобразить скорость вычислений (например, в мегафлопах) при растущем n и $p = \alpha f(n)$ (см. рис. 1.2.3); таким образом, число процессоров остается пропорциональным количеству операций. Пусть принят этот подход и выбран размер задачи n_1, посильный для одного процессора. Соответствующий

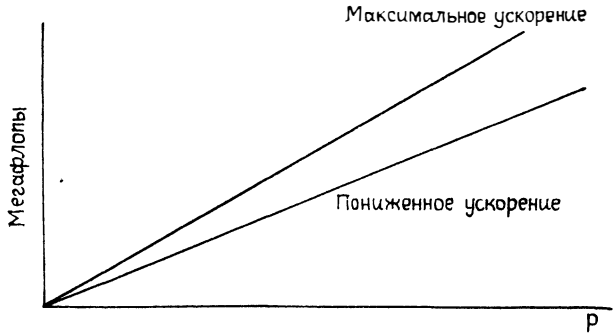


Рис. 1.2.3. Ускорение, выраженное через скорость вычислений

размер задачи n_p для системы из p процессоров определяется равенством $f(n_p) = pf(n_1)$, чтобы число операций, выполняемых p процессорами, в p раз превосходило число операций одного процессора.

Векторные компьютеры

До сих пор в этом параграфе мы ограничивались обсуждением параллельных компьютеров, однако многое из сказанного допускает естественные аналогии или интерпретации для векторных машин. С другой стороны, потребуются и некоторые дополнительные определения.

1.2.7. Определение. *Степенью векторизации* вычислений в векторном компьютере называется длина используемых векторов. Если на разных отрезках вычислений длины векторов различны, то их средняя длина называется *средней степенью векторизации.*

Заметим, что степень параллелизма задачи не обязательно совпадает со степенью векторизации. В качестве простого примера рассмотрим вычисление значений n функций $f_1(x_1)$, ..., $f_n(x_n)$, предполагая, что для вычисления каждого значения необходимо одно и то же время. Тогда степень параллелизма задачи равна n. Если к тому же все функции совпадают, т. е.

нужно вычислить значения одной функции при различных значениях x_i аргумента, то вычисления можно провести на векторном компьютере с векторами длины n (см. упражнение 1.2.3). Таким образом, степень векторизации равна n, как и степень параллелизма. Однако если все f_i задаются разными аналитическими формулами, т. е. векторизация невозможна, то степень векторизации равна 1, тогда как степень параллелизма сохраняет значение n.

Время подготовки данных для параллельных компьютеров имеет естественный аналог в случае векторных компьютеров.

1.2.8. Определение. *Временем подготовки данных* в векторном компьютере называется величина временной задержки, необходимой для перемещения данных, которые требуются для векторных операций, в надлежащие позиции.

Например, как говорилось в предыдущем параграфе, в некоторых векторных компьютерах аргументы векторной операции должны располагаться в последовательно адресуемых ячейках памяти. Чтобы обеспечить такое расположение, могут понадобиться переносы данных с соответствующей задержкой вычислений. Для больших задач, требующих доступа к внешним устройствам памяти, задержки, связанные с обменом, также могут быть включены во время подготовки данных. Заметим, что однопроцессорные векторные компьютеры внутренне синхронизованы и вопроса о синхронизации здесь не возникает.

Ускорения S_p и S'_p для параллельных компьютеров также имеют естественные аналоги в случае векторных компьютеров:

$$S_v = \frac{\text{время выполнения алгоритма при использовании только скалярной арифметики}}{\text{время выполнения алгоритма при использовании векторной арифметики}}; \quad (1.2.8)$$

$$S'_v = \frac{\text{время выполнения быстрейшего последовательного алгоритма на последовательном компьютере}}{\text{время выполнения векторного алгоритма на векторном компьютере}}. \quad (1.2.9)$$

Величина S_v, как и S_p, представляет собой внутреннюю меру ускорения, получаемого при использовании (там, где это возможно) векторных операций, а не одних только скалярных. Величина S'_v более практична — именно ее хотели бы знать люди, приобретающие векторные компьютеры!

Модель ускорения (1.2.6) можно интерпретировать для векторных компьютеров следующим образом. Пусть

$$S_v = \frac{T_1}{(a_1 + a_2/k + a_3/p)\,T_1 + t_d}, \quad (1.2.10)$$

где T_1 — время решения задачи при использовании только скалярной арифметики, p — увеличение скорости по сравнению со скалярной арифметикой при использовании векторных операций оптимальной длины, α_3 — доля операций, которые можно выполнять с векторами оптимальной длины, k — среднее ускорение векторных операций для векторов неоптимальной длины, α_2 — доля операций с векторами неоптимальной длины, α_1 — доля операций, выполняемых в скалярной арифметике, и t_d — общее время подготовки данных.

Использованный выше термин «оптимальная длина» можно интерпретировать таким образом. Для многих векторных компьютеров скорость вычислений растет с увеличением длины вектора по крайней мере до достижения некоторого максимального значения длины. Однако с практической точки зрения можно сказать, что длины векторов «оптимальны», если они позволяют получить, скажем, 99 % или 95 % максимальной производительности.

Как и для параллельных компьютеров, полезно рассмотреть некоторые специальные случаи.

Случай 1. $\alpha_1 = \alpha_2 = 0$, $\alpha_3 = 1$, $t_d = 0$. Здесь все операции производятся с векторами оптимальной длины, а задержки отсутствуют, поэтому S_v есть просто ускорение оптимальной векторной арифметики по сравнению со скалярной. Для большинства векторных компьютеров оно находится в пределах от 10 до 100.

Случай 2. $\alpha_1 = 0$, $\alpha_2 = 1$, $\alpha_3 = 0$, $t_d = 0$. Теперь ускорение равно $k < p$; это уменьшение по сравнению с оптимальным ускорением вызвано использованием векторов неоптимальной длины.

Случай 3. $\alpha_2 = 0$, $t_d = 0$, $\alpha_1 = \alpha$, $\alpha_3 = 1 - \alpha$. В этом случае (1.2.10) принимает вид

$$S_v = \frac{1}{\alpha + (1 - \alpha)\, p}.$$

Предпосылки этого случая таковы: доля α от общего числа операций выполняется в скалярной арифметике, а остальные операции — в оптимальной векторной арифметике. Падение ускорения вновь можно проиллюстрировать рис. 1.2.2, если сделать (нереалистическое) предположение о том, что векторная арифметика в 100 раз быстрее скалярной. При меньших коэффициентах ускорения сохраняется форма кривой на рис. 1.2.2. Важно заметить, что даже незначительное использование скалярной арифметики серьезно уменьшает ускорение. Интересен случай $\alpha = 1/2$. Здесь $S_v < 2$, что означает: если 50 % общего числа

операций должно производиться в скалярной арифметике, то можно получить ускорение не более чем в 2 раза по сравнению со скалярной программой, даже если векторные операции выполняются бесконечно быстро.

Параллельно-векторные компьютеры

Для параллельных систем, составленных из векторных компьютеров, введенные выше понятия нужно комбинировать. Как уже говорилось в предыдущем параграфе, многие системы ближайшего будущего должны состоять из сравнительно небольшого числа (от 4 до 16) мощных векторных компьютеров. Кроме того, многие параллельные системы из большего числа более медленных процессоров также смогут выполнять векторную арифметику. В идеальном случае эффективное использование параллельно-векторных систем требует загрузки процессоров относительно большими заданиями, не требующими больших обменов и синхронизации, а также допускающими высокую степень векторизации. С использованием понятия зернистости это можно выразить так: в случае параллельно-векторных компьютеров целью является конструирование параллельных алгоритмов с крупной зернистостью на уровне заданий; при этом каждое задание имеет векторизуемую мелкозернистость. Следовательно, нужна не столько высокая степень параллелизма, сколько относительно небольшой параллелизм, но на уровне крупных подзадач, обладающих к тому же высокой векторизуемостью.

Согласованность

В принципе можно построить параллельный или векторный алгоритм, который превосходит соответствующий скалярный алгоритм, пока порядок задачи достаточно мал, однако с ростом порядка становится менее выгодным, чем скалярный алгоритм. Вначале мы отразим эту возможность в приводимом ниже определении, а затем приведем нетривиальный пример.

1.2.9. Определение. Говорят, что векторный алгоритм, решающий задачу порядка n, *согласован* с наилучшим последовательным алгоритмом для той же задачи, если отношение $V(n)/S(n)$ остается ограниченным при $n \to \infty$. Здесь $V(n)$ и $S(n)$ — общее число арифметических операций для векторного и последовательного алгоритмов соответственно. Векторный алгоритм *не согласован*, если $V(n)/S(n) \to \infty$ при $n \to \infty$.

Аналогичное определение можно дать для параллельных алгоритмов.

Рекурсивное удвоение

Опишем теперь один потенциально полезный, но несогласованный алгоритм. Снова рассмотрим задачу суммирования n чисел $a_1, \ldots, a_n$, но предположим, что нам нужны и промежуточные суммы

$$s_i = s_{i-1} + a_i, \quad i = 2, \ldots, n, \quad s_1 = a_1. \tag{1.2.11}$$

Если выполнять вычисления (1.2.11) последовательно, то мы получим все требуемые промежуточные суммы как побочный продукт вычисления s_n. Однако параллельный алгоритм сдваивания (см. рис. 1.2.1) находит лишь некоторые из частичных сумм. Чтобы вычислить все частичные суммы (параллельным или векторным способом), можно организовать процесс так,

a_1		s_{11}	s_{11}		s_{11}	s_{11}			s_{11}
a_2	a_1	s_{12}	s_{12}		s_{12}	s_{12}			s_{12}
a_3	a_2	s_{23}	s_{23}	s_{11}	s_{13}	s_{13}			s_{13}
a_4	a_3	s_{34}	s_{34}	s_{12}	s_{14}	s_{14}			s_{14}
a_5	a_4	s_{45}	s_{45}	s_{23}	s_{25}	s_{25}	s_{11}		s_{15}
a_6	a_5	s_{56}	s_{56}	s_{34}	s_{36}	s_{36}	s_{12}		s_{16}
a_7	a_6	s_{67}	s_{67}	s_{45}	s_{47}	s_{47}	s_{13}		s_{17}
a_8	a_7	s_{78}	s_{78}	s_{56}	s_{58}	s_{58}	s_{14}		s_{18}

Рис. 1.2.4. Рекурсивное удвоение

как это показано на рис. 1.2.4, где столбцы представляют векторы. Та же идея применима к вычислению произведений (упражнение 1.2.2).

На рис. 1.2.4 s_{ij} обозначает сумму $a_i + \ldots + a_j$; таким образом, требуемые частичные суммы — это числа s_{1j}; именно они являются элементами финального вектора. Первое сложение векторов даст s_1 и s_2 (или, иначе, s_{11} и s_{12}), а также несколько промежуточных сумм. Второе сложение векторов порождает s_3 и s_4 и еще несколько промежуточных сумм. Наконец, при последнем сложении получаем s_5, s_6, s_7 и s_8. Этот алгоритм рекурсивного удвоения известен также под названием *каскадного* метода вычисления частичных сумм.

Пропуски компонент векторных операндов на рис. 1.2.4 указывают, что над соответствующими позициями операция не производится. Таким образом, первая векторная операция состоит из семи сложений, вторая из шести, а третья из четырех. Если $n = 2^k$, то процесс состоит из k векторных сложений, причем векторы имеют длины $n - 2^i$, $i = 0, 1, \ldots, k - 1$. Поэтому

средняя длина векторов при сложении равна

$$\frac{1}{k}\sum_{i=0}^{k-1}(n-2^i)=\frac{1}{k}(kn-2^k+1)=\frac{1}{\log n}[n(\log n-1)+1]. \tag{1.2.12}$$

Итак, складывается следующая ситуация. Чтобы получить все нужные суммы s_{1j}, требуется $\log n$ векторных или параллельных операций с векторами, средние длины которых указаны в (1.2.12). Следовательно, общее число результатов, получаемых при выполнении $\log n$ векторных операций, составляет $n\log n - n + 1$ по сравнению с $n-1$ результатами при последовательном вычислении. Поскольку $(n\log n - n)/n \to \infty$ при $n \to \infty$, алгоритм не согласован. Но, как мы сейчас покажем, это не означает, что он бесполезен.

Предположим, что векторный компьютер производит векторное сложение векторов длины n за время $T = s + \gamma n$, а скалярное сложение за время μ. Тогда в соответствии с (1.2.12) время вычисления $n-1$ сумм s_{1j} $(j = 2, \ldots, n)$ с помощью алгоритма рекурсивного удвоения при $n = 2^k$ равно

$$T_v(n) \equiv \sum_{i=0}^{k-1}[s+\gamma(n-2^i)] = ks+\gamma(kn-2^k+1) = \\ = s\log n + \gamma(n\log n - n + 1), \tag{1.2.13}$$

а время соответствующего скалярного вычисления равно $T_s(n) = \mu(n-1)$. В таблице 1.2.1 указано время векторных и скалярных вычислений при различных n; предполагается, что $s = 1000$, $\gamma = 10$, а $\mu = 200$ (все три значения в наносекундах).

Таблица 1.2.1. Время векторного алгоритма рекурсивного удвоения и скалярного алгоритма

n	$T_v(n)$	$T_s(n)$
8	3170	1400
32	6190	6200
128	15×10^3	25×10^3
1024	0.1×10^6	0.2×10^6
2^{20}	90×10^6	200×10^6
2^{21}	419×10^6	415×10^6

Согласно этой таблице, скалярный алгоритм эффективней векторного примерно до порядка $n = 32$, что объясняется доминированием времени запуска в векторных операциях при малом n.

В диапазоне значений n от 32 до 2^{20} преимущество на стороне рекурсивного удвоения. Однако для $n > 2^{21}$ член $n \log n$ начинает доминировать и скалярный алгоритм снова становится более эффективным. Таким образом, для диапазона значений n, включающего все достаточно большие, но все еще реальные длины векторов, алгоритм рекурсивного удвоения будет выгодней скалярного алгоритма. Этот вывод зависит, разумеется, от конкретных значений s, γ и μ, использованных в примере. Ключевым фактором является отношение коэффициентов при n, т. е. $\gamma(\log n - 1)$ и μ.

Анализ потоков данных

Как мы увидим в последующих параграфах, во многих случаях внутренний параллелизм алгоритма обнаружить весьма несложно. В других случаях он не столь очевиден. Систематическим средством выявления параллелизма является *граф предшествования* вычислительного процесса. Рис. 1.2.5 иллюстрирует это понятие на примере вычисления

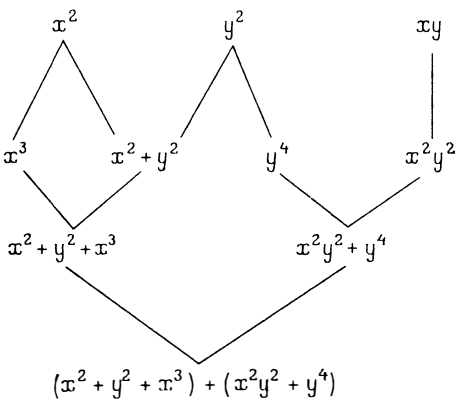


Рис. 1.2.5. Граф предшествования

$$f(x, y) = x^2 + y^2 + x^3 + y^4 + x^2y^2. \qquad (1.2.14)$$

Последовательное вычисление выражения (1.2.14) может происходить таким образом:

$$x,\ y,\ x^2,\ y^2,\ x^2 + y^2,\ x^3,\ (x^2 + y^2) + x^3,\ y^4,\ x^2y^2,$$
$$y^4 + x^2y^2,\ (x^2 + y^2 + x^3) + (y^4 + x^2y^2).$$

Это требует десяти арифметических операций. В соответствии с рис. 1.2.5 эти десять операций можно выполнить за четыре параллельных шага.

Использование графов типа графа на рис. 1.2.5 называется *анализом потоков данных*. Такой анализ показывает, какие вычисления нужно выполнить, чтобы подготовить данные для последующих вычислений. Мы применим этот подход в гл. 2 для выявления внутреннего параллелизма некоторых прямых методов решения линейных уравнений. Другой аспект анализа потоков данных можно назвать *принципом потока данных: как только нечто может быть сделано, нужно это сделать.*

Упражнения к параграфу 1.2

1. Сформулировать в соответствии с рис 1.2.1 алгоритм сдваивания для задачи сложения $s = 2^r$ векторов $a_1, \ldots, a_s$ длины n. Написать псевдокод алгоритма для векторной машины.

2. Сформулировать алгоритм рекурсивного удвоения, аналогичный алгоритму на рис. 1.2.4, для вычисления произведений

$$p_i = a_1 \ldots a_i, \quad i = 1, \ldots, n.$$

3. Пусть $f(x) = x^2 + x^3$. Написать псевдокод, использующий векторы длины n для вычисления функции f в точках $x_1, \ldots, x_n$.

4. Написать векторный псевдокод для сложения $s = 2^r$ чисел методом сдваивания. Рассмотреть управление памятью, считая, что элементы вектора должны располагаться в последовательно адресуемых ячейках памяти. Полагая, что векторные операции требуют $(1000+10m)$ нс для векторов длины m, а скалярные арифметические операции — 100 нс, найти критическое значение s_c, за которым векторная программа эффективней скалярной программы алгоритма (1.2.2).

5. Пусть $q(x) = a_0 + a_1 x + \ldots + a_n x^n$ — многочлен степени n, причем $n = r2^r$. Написать параллельную программу для вычисления значения $q(x)$, представив q в виде

$$q(x) = a_0 + q_1(x) + x^r q_2(x) + x^{2r} q_3(x) + \ldots + x^{(s-1)r} q_s(x),$$

где $s = 2^r$ и

$$q_i(x) = a_k x + \ldots + a_{k+r-1} x^r, \quad k = (i-1)r + 1.$$

Организовать процесс следующим образом:

a). Вычислить $x^2, \ldots, x^r$ (последовательно?).

b). Вычислить $q_1(x), \ldots, q_s(x)$ (параллельно).

c). Вычислить $x^r, x^{2r}, \ldots, x^{(s-1)r}$ (последовательно?).

d). Вычислить произведения $x^r q_2(x)$, $x^{2r} q_3(x)$, $\ldots$, $x^{(s-1)r} q_s(x)$ (параллельно).

e). Вычислить сумму $a_0 + q_1(x) + x^r q_2(x) + \ldots + x^{(s-1)r} q_s(x)$ (алгоритмом сдваивания)

Адаптировать свою параллельную программу к векторному компьютеру, реализуя шаги b, d и e с помощью векторных операций:

b′). Пусть v_1 — вектор, составленный из младших коэффициентов многочленов q_i, v_2 — вектор из коэффициентов при x^2 в тех же многочленах, и т. д. Образовать сумму $xv_1 + x^2 v_2 + \ldots + x^r v_r$.

e′). Использовать упражнение 1.2.4.

Литература и дополнения к параграфу 1.2

1. В той или иной форме представления о степени параллелизма и ускорении использовались уже многие годы. Определение 1.2.6 времени подготовки данных является новым, оно объединяет различные факторы, обусловливающие задержку вычислений вследствие неподготовленности данных, подлежащих обработке

2. Формула ускорения (1 2 7) приведена в статье [Ware, 1973] и (неявно) используется в [Amdahl, 1967] Более общая формула (1.2.6) является новой Ряд интересных замечаний относительно закона Уэра содержится в работах [Buzbee, 1983, 1985].

3. В ранних теоретических работах о параллельных алгоритмах наблюдалась тенденция игнорирования стоимости обменов и синхронизации, см. обзор соответствующих исследований в области алгоритмов линейной алгебры в статье [Heller, 1978]. Джентлмен [Gentleman, 1978] одним из первых указал на то, что время обменов может быть важным или даже доминирующим фактором в реальных вычислениях. Современный подход к этой проблеме изложен в работе [Gannon, van Rosendale, 1984]

4. Идея согласованности впервые была высказана в статье [Lambiotte, Voigt, 1975]

5 Предложение использовать производительность в мегафлопах для измерения ускорений в тех случаях, когда задача слишком велика и не помещается в памяти одного процессора, сделано в статье [Moler, 1986]. В этой же статье введено понятие *коэффициента использования плавающей точки* в параллельных вычислениях Эта характеристика определяется формулой $u = \gamma/(\tau p)$, где γ — скорость вычислений в мегафлопах для системы из p процессоров, а τ — скорость в мегафлопах для одного процессора Ясно, что $u = 1$ в случае максимального ускорения.

6 Анализ влияния затрат, связанных с обменами, на различные элементарные операции (например, сложение) проведен в работе [Parkinson, 1987]

7. Одна из первых статей, где в общем виде была поставлена задача балансировки нагрузки, — это [Bokhari, 1981] В работе [Fox et al , 1987] обсуждается проблема динамической балансировки нагрузки

8 Рекурсивное удвоение использовано в статье [Stone, 1973] как средство получения параллельных алгоритмов для рекуррентных соотношений. Независимо от Стоуна и примерно в одно время с ним рекурсивное удвоение изобрели Харвард Лоумакс и Роберт Даунз. В работе [Lambiotte, Voigt, 1975] отмечено, что оно приводит к несогласованным алгоритмам (это показано в основном тексте параграфа) Более подробное обсуждение рекурсивного удвоения и каскадного метода вычисления частичных сумм можно найти в книге [Hockney, Jesshope, 1981]

9. Перестройка вычислений, подобная переходу к алгоритму сдваивания (рис. 1 2.1) от последовательного алгоритма (1 2.2), поднимает вопрос о численной устойчивости. Другими словами, если исходный алгоритм численно устойчив, то сохранит ли он устойчивость после изменения порядка операций? Этот вопрос рассматривается в статье [Rönsch, 1984], в ней показано, что алгоритм сдваивания более устойчив, чем алгоритм (1.2 2) См. по этому поводу также [Gao, 1987].

10. Синхронизацию можно реализовать разными способами, и эти способы действительно применялись на практике Мы изложим сейчас некото-

рые основные идеи. Хороший общий обзор этой тематики дан в [Andrews, Schneider, 1983]. Отметим, что значительная часть понятий и терминологии, относящейся к синхронизации, возникла в теории операционных систем.

Критическим сечением программы называется последовательность операторов, которая должна выполняться как неделимое целое без прерываний. Это последовательная часть общей программы За критическим сечением обычно следует *ветвление*, инициирующее параллельно выполняемые участки программы. В месте *соединения* эти параллельные сегменты возвращаются к критическому сечению; см. рис. 1.2 6.

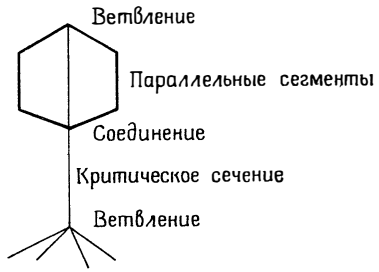


Рис. 1.2.6.

Родственным понятием является *барьер* [Jordan, 1986; Axelrod, 1986], представляющий собой логическую точку потока управления, которой должны достигнуть все процессы, прежде чем их можно будет продолжить.

В машинах с разделяемой памятью синхронизация обычно реализуется с помощью *разделяемых переменных*, к которым может обращаться несколько процессов или процессоров Так, например, в схеме «занят — ожидаю» процесс проверяет значение некоторой разделяемой переменной Определенное ее значение указывает, что процесс может продолжаться. О процессе, ожидающем этого значения, говорят, что он *прокручивается вхолостую* (в оригинале используется термин spinning — *Перев*.) Разделяемую переменную или переменные при этом называют *замко́выми переменными* (spin locks).

11. В работе [Greenbaum, 1986a] анализируются различные стратегии синхронизации, включая использование барьеров. Рассматривается также *локальная синхронизация* (в оригинале neighbor synchronization. — *Перев.*), при которой достаточно синхронизовать действия только соседей данного процессора P, посылающих ему свои данные. Так, в частности, обстоит дело при итерационном решении линейных уравнений в решеточном массиве процессоров. Усовершенствованием локальной синхронизации является *граничная синхронизация*; она основана на том, что во многих случаях достаточно обмена данными лишь в некоторых узлах параллельной системы (например в граничных узлах) Автор заключает, что локальный и граничный подходы дают весьма привлекательные схемы синхронизации для задач с достаточно крупной зернистостью, в которых время, затрачиваемое на синхронизацию, мало по сравнению с временем работы каждого процессора.

12. Способ реализации суммирования сдваиванием для данной параллельной системы зависит от характера соединений в ней. Рассмотрим, например, остовное дерево гиперкуба на рис. 1.1.19 Здесь сложение по методу сдваивания можно реализовать путем обхода остовного дерева в обратном порядке. Пусть, например, нужно сложить 16 чисел и в каждом процессоре поначалу хранятся два из них. На первом шаге все процессоры складывают свои два числа. Затем процессоры 5--8 пересылают свои суммы процессорам 1—4 (как и предписано графом) Процессоры 1—4 выполняют сложение, после чего процессоры 3 и 4 отправляют свои результаты процессорам 1 и 2. Последние производят сложение, а затем процессор 2 переправляет свою сумму процессору 1. Еще одно сложение в этом процессоре завершает вычисления.

1.3. Умножение матриц

В этом параграфе мы рассмотрим задачу вычисления произведений $A\mathbf{x}$ и AB, где A и B — матрицы, а $\mathbf{x}$ — вектор. Эти базисные операции важны для последующих глав. Кроме того, изучение матричного умножения позволит нам на примере очень простой математической задачи исследовать ряд основных вопросов параллельных и векторных вычислений. Сначала мы предположим, что A и B — заполненные матрицы; разреженные матрицы различных типов будут рассмотрены несколько позднее в этом же параграфе. Сперва мы обсудим векторные компьютеры, а затем перейдем к параллельным.

Умножение матрицы на вектор

Пусть A — матрица $m \times n$, а $\mathbf{x}$ — вектор длины n. Тогда

$$A\mathbf{x} = \begin{bmatrix} (\mathbf{a}_1, \mathbf{x}) \\ \cdot \\ \cdot \\ \cdot \\ (\mathbf{a}_m, \mathbf{x}) \end{bmatrix}, \tag{1.3.1}$$

где $\mathbf{a}_i$ — i-я строка матрицы A, а $(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} x_i y_i$ — обычное скалярное произведение. Таким образом, вычисление вектора $A\mathbf{x}$ сводится к вычислению m скалярных произведений. Прежде чем более подробно исследовать этот способ, рассмотрим другой стандартный подход к матрично-векторному произведению, а именно представление его в виде линейной комбинации столбцов матрицы A:

$$A\mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{a}_i, \tag{1.3.2}$$

где $\mathbf{a}_i$ теперь обозначает i-й столбец A.

Заметим, что различие представлений (1.3.1) и (1.3.2) можно рассматривать как различие двух способов доступа к данным, что показывают программы на рис. 1.3.1. В обоих случаях предполагается, что до начала выполнения программы переменной y_i присвоено нулевое значение. В левой программе на рис. 1.3.1 при каждом значении i в цикле по j вычисляется скалярное произведение i-й строки матрицы A и вектора $\mathbf{x}$; следовательно, эта программа реализует формулу (1.3.1). Правая программа соответствует формуле (1.3.2). Обратим внимание на то, что заключительный арифметический оператор одинаков в обеих программах, и различие только в порядке индексов. В дальнейшем мы увидим, что расположение индексов в

различном порядке приводит к разным алгоритмам и для задач матричного умножения и решения линейных систем.

В некоторых векторных компьютерах имеется аппаратно реализованная команда скалярного произведения; вскоре мы приведем соответствующий пример. В противном случае, чтобы применить представление (1.3.1), нужно вычислять скалярное произведение

$$t_i = x_i y_i, \quad i = 1, \ldots, n, \quad (\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} t_i. \tag{1.3.3}$$

Первым шагом при этом будет покомпонентное умножение векторов, а вторым — суммирование, параллелизм которого, как мы видели в предыдущем параграфе, не идеален. Для векторных компьютеров мы модифицируем алгоритм суммирования

Для $i = 1$ до m	Для $j = 1$ до n
Для $j = 1$ до n	Для $i = 1$ до m
$y_i = y_i + a_{ij} x_j$	$y_i = y_i + a_{ij} x_j$

Рис. 1.3.1. Формы ij и ji матрично-векторного умножения

сдваиванием из § 1.2 (см. рис. 1.2.1 и упражнение 1.2.4). Пусть, например, в данном компьютере понятие вектора предполагает расположение элементов в последовательно адресуемых ячейках памяти. Тогда мы сложим первые $n/2$ элементов с последними $n/2$ элементами, и на следующих шагах будем продолжать этот процесс сдваивания. Заметим, что в какой-то момент длина векторов станет меньше некоторого барьерного значения; тогда для сложения оставшихся чисел, возможно, более эффективной окажется скалярная арифметика.

Алгоритм, основанный на представлении (1.3.2), записывается так:

$$\mathbf{y} = 0, \quad \text{для} \quad i \text{ от } 1 \text{ до } n \text{ выполнить } \mathbf{y} = \mathbf{y} + x_i \mathbf{a}_i. \tag{1.3.4}$$

Как уже говорилось, операция типа «вектор плюс произведение вектора на число» называется триадой (или операцией saxpy); некоторые векторные компьютеры выполняют ее особенно эффективно.

Сравнение способов (1.3.1) и (1.3.2) начнем с примера. В число аппаратно реализованных команд машины CDC CYBER 205 входит команда скалярного умножения двух векторов длины n. В случае двухконвейерной машины приближенная формула времени выполнения этой команды имеет вид (2300 +

$+ 20n)$ нс, так что вычисление m скалярных произведений потребует

$$(2300m + 20nm) \text{ нс.} \tag{1.3.5}$$

Для той же машины время триады, выполняемой с векторами длины m, приближенно равно $(1700 + 10m)$ нс, поэтому для алгоритма (1.3.4) нужно

$$(1700n + 10nm) \text{ нс.} \tag{1.3.6}$$

Очевидно, что для большинства значений n и m время (1.3.5) примерно вдвое больше времени (1.3.6); в частности, это верно при $m = n$. Однако для малых значений m время (1.3.5) может оказаться меньшим из двух. Например, при $m = 5$ и $n > 6$

$$2300 \cdot 5 + 20 \cdot 5n < 1700n + 10 \cdot 5n.$$

Это отражает тот факт, что степень векторизации (или параллелизма) алгоритма (1.3.4) равна m, следовательно, при малых m он неэффективен.

Этот пример показывает, что выбор того или иного алгоритма может зависеть от величины некоторых параметров задачи; в дальнейшем мы встретимся еще с несколькими аналогичными примерами. Очень важен также способ хранения данных. Предположим, что матрица A хранится по столбцам; такое соглашение в отношении двумерных массивов принято в Фортране. (Однако в других языках, к примеру, в Паскале или PL1 двумерные массивы хранятся по строкам.) Тогда векторы, требуемые для алгоритма (1.3.2), располагаются в последовательно адресуемых ячейках памяти, в то время как в алгоритме скалярных произведений (1.3.1) строки матрицы представляют собой векторы с шагом m. Как отмечено выше, в некоторых векторных компьютерах в качестве операндов векторных арифметических команд допускаются только векторы с шагом 1, а если шаг больше 1, то перед началом вычислений нужно переместить данные в памяти. В других компьютерах разрешается шаг больший 1, однако скорость векторной операции может уменьшиться. В любом случае, если матрица A хранится по столбцам, то соображения, связанные с памятью, усиливают аргументацию в пользу алгоритма (1.3.2). С другой стороны, если A уже хранится по строкам, то, поскольку такой способ хранения удобней для алгоритма скалярных произведений, это может заставить предпочесть именно его. Только детальный анализ может показать, какой выбор следует сделать для конкретной машины.

Развертывание циклов

Рассмотрим теперь вопрос реализации алгоритма (1.3.4) для машин типа «регистр — регистр». Прямолинейная реализация предусматривала бы такую последовательность операций:

Загрузить $\mathbf{y} = 0$ в векторный регистр.
Загрузить $\mathbf{a}_1$ в векторный регистр.
Умножить: $x_1\mathbf{a}_1$. Результат записывается в регистр.
Сложить: $\mathbf{y} = \mathbf{y} + x_1\mathbf{a}_1$. Результат записывается в регистр.
Записать $\mathbf{y}$ в оперативную память
Загрузить $\mathbf{y}$.
Загрузить $\mathbf{a}_2$.

Предполагается, что векторы можно разместить в регистрах; в противном случае мы обрабатывали бы их по частям.

Приведенная программа очень неэффективна. Каждый цикл требует загрузки двух векторов, записи одного вектора в память и выполнения двух векторных арифметических операций. Что касается последних, то нам следовало бы применить зацепление, если конструкция машины это позволяет. Зацепление, иллюстрируемое рис. 1.1.4, позволяет направить результат операции $x_1\mathbf{a}_1$ непосредственно в сумматор. В этом случае сложение с $\mathbf{y}$ может начаться на фоне продолжающихся умножений, так что обе векторные операции будут выполняться почти одновременно. Теперь заметим, что загрузки и записи вектора $\mathbf{y}$ между векторными операциями совсем не обязательны; мы можем считывать текущее значение $\mathbf{y}$ непосредственно из регистра. Наконец, большинство машин типа «регистр — регистр» позволяют совмещать загрузки с арифметическими операциями. Таким образом, можно загружать $\mathbf{a}_2$, пока выполняется операция $\mathbf{y} + x_1\mathbf{a}_1$. (Отметим, однако, что $\mathbf{a}_2$ и $\mathbf{a}_1$ должны быть помещены в разные векторные регистры, поэтому при выборке векторов $\mathbf{a}_i$ из регистров в программе должны производиться переключения.)

С указанными изменениями программа могла бы принять следующий вид:

Загрузить $\mathbf{y} = 0$ в векторный регистр.
Загрузить $\mathbf{a}_1$ в векторный регистр.
Образовать сумму $\mathbf{y} = \mathbf{y} + x_1\mathbf{a}_1$ путем зацепления. Загрузить $\mathbf{a}_2$.
Образовать сумму $\mathbf{y} = \mathbf{y} + x_2\mathbf{a}_2$ путем зацепления. Загрузить $\mathbf{a}_3$.

Теперь каждый цикл требует лишь немного больше времени, чем одна векторная арифметическая операция, и такая программа будет работать почти в пять раз быстрее исходной.

Трансляторы обычно распознают ситуацию, в которой можно использовать зацепление. К сожалению, они не всегда способны установить, что промежуточные результаты можно оставить в векторных регистрах или что можно совместить загрузки и арифметические операции. Разумеется, этот недостаток можно устранить, переходя на язык ассемблера. При программировании на Фортране его можно в значительной степени смягчить с помощью техники *развертывания циклов*. Например, заменим цикл

$$\text{Для} \quad i = 1 \text{ до } n$$
$$\mathbf{y} = \mathbf{y} + x_i \mathbf{a}_i$$

циклом

$$\text{Для} \quad i = 2 \text{ до } n \text{ с шагом } 2$$
$$\mathbf{y} = \mathbf{y} + x_{i-1}\mathbf{a}_{i-1} + x_i \mathbf{a}_i.$$

В этом случае вектор $\mathbf{y}$ будет загружаться в регистры и записываться в память вдвое реже, чем в исходном варианте; кроме того, загрузка вектора $\mathbf{a}_i$ будет происходить на фоне выполнения операции $\mathbf{y} + x_{i-1}\mathbf{a}_{i-1}$. (Отметим, что при нечетном n потребуется дополнительный оператор для обработки вектора $\mathbf{a}_n$). В данном примере цикл был развернут *на глубину* 2. Развертывание на глубину 3 достигается в цикле

$$\text{Для} \quad i = 3 \text{ до } n \text{ с шагом } 3$$
$$\mathbf{y} = \mathbf{y} + x_{i-2}\mathbf{a}_{i-2} + x_{i-1}\mathbf{a}_{i-1} + x_i \mathbf{a}_i.$$

Аналогичным образом осуществляется развертывание на бо́льшую глубину. (Если n не кратно глубине, снова требуются дополнительные операторы.) На практике применялось развертывание циклов до глубины 8 и даже большей.

Умножение матриц

Проведенное обсуждение матрично-векторного умножения естественным образом обобщается на задачу умножения матриц, хотя, как мы увидим, здесь появляются и новые возможности. Пусть A и B — матрицы $m \times n$ и $n \times q$ соответственно, так что AB — матрица $m \times q$.

Матрично-векторное умножение в форме скалярных произведений в результате обобщения приводит к алгоритму *скаляр-*

ных произведений

$$C = AB = \begin{bmatrix} \mathbf{a}_1 \\ \cdots \\ \mathbf{a}_m \end{bmatrix} (\mathbf{b}_1 \ \ldots \ \mathbf{b}_q) = (\mathbf{a}_i \mathbf{b}_j). \tag{1.3.7}$$

Здесь матрица A разбита на строки, а матрица B — на столбцы, и вычисление произведения AB свелось к формированию mq скалярных произведений $\mathbf{a}_i\mathbf{b}_j$ строк A и столбцов B. В идеальном случае A следовало бы хранить по строкам, а B — по столбцам. Заметим, что при $q = 1$ матрица B представляет собой вектор и (1.3.7) превращается в алгоритм скалярных произведений для матрично-векторного умножения. Алгоритм (1.3.7) имеет те же достоинства и недостатки, что и соответствующий матрично-векторный алгоритм, и мы больше не будем обсуждать его.

Следующий алгоритм основывается на повторных матрично-векторных умножениях. Пусть $\mathbf{a}_i$ и $\mathbf{b}_i$ — i-е столбцы матриц A и B соответственно. Тогда

$$C = AB = (A\mathbf{b}_1, \ \ldots, \ A\mathbf{b}_q) = \left(\sum_{j=1}^{n} b_{j1}\mathbf{a}_j, \ \ldots, \ \sum_{j=1}^{n} b_{jq}\mathbf{a}_j \right). \tag{1.3.8}$$

Чтобы получить i-й столбец матрицы C, нужно умножить A на i-й столбец матрицы B; эти матрично-векторные произведения вычисляются как линейные комбинации столбцов матрицы A. Псевдокод алгоритма (1.3.8) приведен на рис. 1.3.2. Сам алгоритм иногда называют *алгоритмом средних произведений*[1]); наиболее удобно для него хранение матрицы A по столбцам, тогда как способ хранения B несуществен. Предположим, что B хранится по строкам, и пусть $\mathbf{a}_i$ и $\mathbf{b}_i$ теперь обозначают i-е строки в A и B. Существует альтернативный способ вычисления матрицы AB, который мы назовем *двойственным алгоритмом средних произведений:*

$$C = AB = \begin{bmatrix} \mathbf{a}_1 B \\ \cdots \\ \mathbf{a}_m B \end{bmatrix} = \begin{bmatrix} \sum a_{1j}\mathbf{b}_j \\ \cdots \\ \sum a_{mj}\mathbf{b}_j \end{bmatrix}. \tag{1.3.9}$$

Здесь i-я строка матрицы C представлена в виде линейной комбинации строк матрицы B. Если в (1.3.9) $m = 1$, то A вектор-

[1]) В оригинале middle product algorithm. Поскольку никаких «средних произведений» в алгебре нет, название алгоритма объясняется, по-видимому, игрой слов, основанной на том, что два других алгоритма перемножения матриц называются inner product algorithm (алгоритм внутренних, т. е. скалярных, произведений) и outer product algorithm (алгоритм внешних произведений). — *Прим. перев.*

строка. В этом случае (1.3.9) превращается в алгоритм типа (1.3.2) для умножения вектора-строки на матрицу.

Существует еще один алгоритм перемножения матриц; в нем используются внешние произведения. *Внешним произведе-*

```
Положить C = 0
Для i = 1 до q
    Для j = 1 до n
        c_i = c_i + b_ji a_j
```

Рис. 1.3.2. Алгоритм средних произведений для матричного умножения

нием вектора-столбца $\mathbf{u}$ длины m и вектора-строки $\mathbf{v}$ длины q называется матрица

$$\mathbf{u}\mathbf{v} = (v_1\mathbf{u}, \ldots, v_q\mathbf{u}) = (u_i v_j). \tag{1.3.10}$$

Внешнее произведение можно составить из произведений компонент вектора v и вектора u; в результате получается матри-

```
Положить C = 0
Для i = 1 до n
    Для j = 1 до q
        c_j = c_j + b_ij a_i
```

Рис. 1.3.3. Алгоритм внешних произведений для матричного умножения

ца, элемент (i, j) которой равен $u_i v_j$. На использовании произведений типа (1.3.10) основан *алгоритм внешних произведений* для матричного умножения:

$$C = AB = (\mathbf{a}_1, \ldots, \mathbf{a}_n)\begin{bmatrix} \mathbf{b}_1 \\ \cdots \\ \mathbf{b}_n \end{bmatrix} = \sum_{i=1}^{n} \mathbf{a}_i\mathbf{b}_i =$$

$$= \sum_{i=1}^{n} (\mathbf{b}_{i1}\mathbf{a}_i, \ldots, \mathbf{b}_{iq}\mathbf{a}_i). \tag{1.3.11}$$

Псевдокод алгоритма приведен на рис. 1.3.3.

Для этого алгоритма наиболее удобно хранение матрицы A по столбцам; способ хранения B несуществен. Если B хранится

по строкам, то можно применить иной способ

$$C = AB = \sum_{i=1}^{n} \mathbf{a}_i \mathbf{b}_i = \sum_{i=1}^{n} \begin{bmatrix} a_{1i}\mathbf{b}_i \\ \cdots \\ a_{mi}\mathbf{b}_i \end{bmatrix}, \qquad (1.3.12)$$

который мы назовем *двойственным алгоритмом внешних произведений*. В любом случае C образуется как сумма n внешних произведений; алгоритмы (1.3.11) и (1.3.12) различаются только способом вычисления этих внешних произведений.

Отметим, что, аналогично двум методам на рис. 1.3.1, все рассмотренные выше алгоритмы матричного умножения можно интерпретировать как различные упорядочения переменных цикла i, j и k в основной программе

Для ———
  Для ———
    Для ———
      $c_{ij} = c_{ij} + a_{ik}b_{kj}$.

Все шесть возможных вариантов, называемых иногда *ijk*-формами матричного умножения, выписаны явно в упражнении 1.3.3.

Сравнение алгоритмов

Теперь мы хотим сравнить алгоритмы средних и внешних произведений в предположении, что A хранится по столбцам. Аналогично проводится сравнение двойственных алгоритмов, если B хранится по строкам. В алгоритме внешних произведений (см. рис. 1.3.3) внутренний цикл формирует внешнее произведение $\mathbf{a}_i\mathbf{b}_i$, прибавляя его к текущей матрице C. Основной операцией является триада для векторов длины m, поэтому степень векторизации равна m. Для алгоритма средних произведений (рис. 1.3.2) базисная операция — это опять-таки триада, и степень векторизации снова равна m. Для векторных компьютеров, в которых данные выбираются непосредственно из памяти, оба алгоритма по существу эквивалентны. Однако при малых m и тот, и другой будут неэффективны; если B хранится по строкам и $q > m$, то следует предпочесть двойственные формы алгоритмов.

Для векторных компьютеров, использующих векторные регистры, эти алгоритмы могут иметь важные различия, к обсуждению которых мы сейчас перейдем. Для простоты предположим, что емкость регистров достаточна для хранения столбцов матрицы A. В противном случае в последующее обсуждение нужно внести очевидные изменения (упражнение 1.3.4). Заметим

прежде всего, что внутренний цикл алгоритма средних произведений полностью формирует столбец матрицы C, тогда как в алгоритме внешних произведений столбцы C накапливаются в течение всего процесса. Рис. 1.3.4 и 1.3.5 иллюстрируют оба способа вычислений для машины с векторными регистрами.

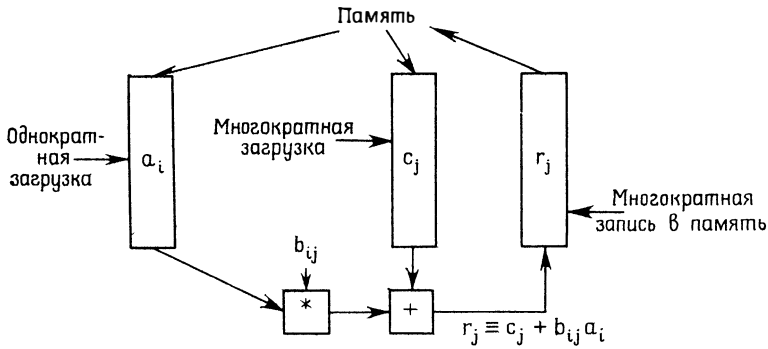


Рис. 1.3.4. Умножение матриц с помощью алгоритма внешних произведений

Алгоритм внешних произведений имеет то достоинство, что вектор $\mathbf{a}_i$, попав в векторный регистр, используется q раз. Однако есть и недостаток, состоящий в том, что после каждой век-

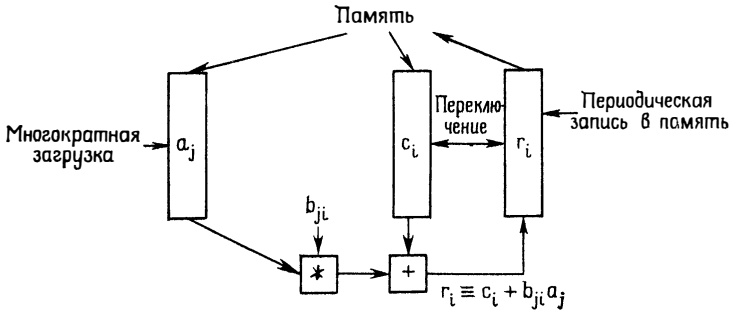


Рис. 1.3.5. Умножение матриц с помощью алгоритма средних произведений

торной операции возникает задержка, пока результат переписывается из регистра в память и загружается вектор $\mathbf{c}_j$. До некоторой степени этот недостаток можно смягчить, используя дополнительный регистр для $\mathbf{c}_{j+1}$: он будет загружаться на фоне выполнения арифметической операции. Это уменьшит задержку до времени, необходимого для записи результата, ценой введения некоторых дополнительных логических операторов;

последние организуют выборку $\mathbf{c}_j$ из разных регистров для соседних шагов процесса. Описанная стратегия полезна для машин типа CRAY-1, в которых арифметика может накладываться на операции с памятью, но в каждый момент разрешается или только загрузка, или только запись. Для таких машин, как CRAY X-MP, где разрешается совмещение операций загрузки и записи, можно полностью исключить задержки, связанные с памятью, путем привлечения еще одного векторного регистра $\mathbf{r}_{j+1}$ для результатов. Теперь основной шаг имеет вид

вычислить $\mathbf{r}_j$, загрузить $\mathbf{c}_{j+1}$, записать $\mathbf{r}_{j-1}$ в память,

причем операции загрузки и записи выполняются одновременно с арифметикой.

Перейдем к алгоритму средних произведений (см. рис. 1.3.5). Заметим, что теперь вектор $\mathbf{a}_j$ должен загружаться перед каждой векторной операцией, но результат записывается в память только один раз, по окончании внутреннего цикла. Однако регистры для $\mathbf{c}_i$ и $\mathbf{r}_i$ должны при каждой векторной операции меняться ролями, поскольку вектор-результат становится на следующем шаге входным вектором $\mathbf{c}_i$. При такой организации алгоритма перед каждой векторной операцией происходит задержка, вызванная загрузкой вектора $\mathbf{a}_j$. Но и теперь можно использовать дополнительный регистр, чтобы совместить эту загрузку с арифметикой согласно предписанию

вычислить $\mathbf{c}_i + b_{ji}\mathbf{a}_j$, загрузить $\mathbf{a}_{j+1}$.

Отметим, что для такого совмещения требуется лишь способность одновременно выполнять загрузку и арифметические операции. При программировании на Фортране мы аппроксимировали бы подобный режим путем развертывания циклов. Подводя итоги, мы заключаем, что алгоритм средних произведений предпочтителен для машин, допускающих совмещение арифметики и операций с памятью только одного типа. Если в компьютере загрузка и запись могут происходить одновременно, то оба алгоритма эквивалентны с точностью до второстепенных эффектов.

Параллельные компьютеры

Рассмотрим теперь те же алгоритмы в применении к параллельной системе из p процессоров. Пусть снова A — матрица $m \times n$. Начнем с вычисления матрично-векторного произведения с помощью линейных комбинаций: $A\mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{a}_i$. Для простоты предположим пока, что $p = n$, и пусть x_i и $\mathbf{a}_i$ приданы процессору i, как это показано на рис. 1.3.6. Все

произведения $x_i\mathbf{a}_i$ вычисляются с максимальным параллелизмом, а затем выполняются сложения по методу сдваивания (см. § 1.2), применяемому теперь к векторам (см. упражнение 1.2.1). Если рассматривается параллельная система с локальной памятью, то рис. 1.3.6 следует интерпретировать так: указанные на нем данные находятся в локальной памяти соответствующего процессора. В этом случае алгоритм сдваивания потребует пересылок данных между процессорами в ходе вычислений. Для системы с разделяемой памятью мы интерпретируем рис. 1.3.6

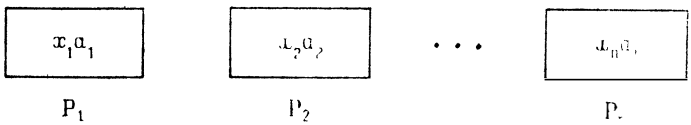


Рис. 1.3.6. Параллельный алгоритм линейных комбинаций

в смысле распределения заданий, т. е. P_i выполняет умножение $x_i\mathbf{a}_i$. В той части процесса, где применяется сдваивание, пересылки данных не нужны, поскольку все находится в глобальной памяти.

Чтобы гарантировать завершение умножений до начала сложения, требуется синхронизация. В машине с локальной памятью ее обеспечивает сама необходимость передачи данных. Предположим, например, что процессор P_1 должен сложить $x_1\mathbf{a}_1$ и $x_2\mathbf{a}_2$. Когда P_2 закончит свои умножения, он пошлет вектор $x_2\mathbf{a}_2$ в P_1. Тот начнет суммирование, завершив собственные умножения; однако он будет ожидать, если необходимые данные из P_2 еще не получены. Именно это ожидание реализует требуемую синхронизацию. Если бы его не было и P_1 начал сложение, пользуясь ячейками памяти, предназначенными для $x_2\mathbf{a}_2$, еще до того, как поступили верные значения, то были бы получены ошибочные результаты. В системе с глобальной памятью синхронизацию можно обеспечить разными способами (см. § 1.2), и какой из них лучше, зависит от конструктивных особенностей машины и ее программной среды. В качестве примера того, как можно реализовать синхронизацию, предположим, что P_2, закончив умножение, устанавливает «флаг» (которым может быть логическая переменная, принявшая значение true), и P_1, прежде чем начать сложение, проверяет этот флаг.

Алгоритм скалярных произведений в некоторых отношениях более привлекателен. По-прежнему будем считать, что $p = m$. Пусть x_i и $\mathbf{a}_i$ (последний символ обозначает теперь i-ю строку матрицы A) приданы процессору i (см. рис. 1.3.7). Каждый

процессор выполняет свое скалярное умножение, и при этом параллелизм максимален. Не требуются ни пересылки данных, ни сдваивание, а синхронизация нужна только в конце вычислений.

Хотя алгоритм скалярных произведений обладает максимальным параллелизмом, выбор того или иного алгоритма определяется обычно другими соображениями. Матрично-векторное умножение неизбежно является частью более широкого процесса вычислений, и для системы с локальной памятью главную роль в выборе алгоритма играет способ хранения A и

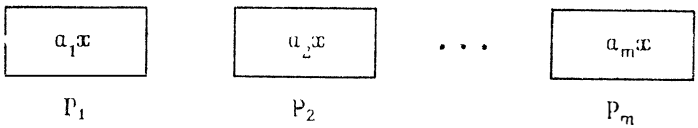


Рис. 1.3.7. Параллельный алгоритм скалярных произведений

$\mathbf{x}$ в момент, когда требуется произведение $A\mathbf{x}$. Например, если x_i и столбец $\mathbf{a}_i$ уже находятся в памяти i-го процессора, как на рис. 1.3.6, то, по всей вероятности, будет использован алгоритм линейных комбинаций, хотя степень параллелизма в нем ниже, чем в конкурирующем алгоритме. Еще одно существенное соображение — желаемое расположение результата по окончании умножения: в первом алгоритме вектор-результат размещается в памяти какого-то одного процессора, тогда как в другом он распределен между процессорами. Отметим кажущийся парадокс (по отношению к ранее рассмотренным векторным программам): алгоритм линейных комбинаций требует суммирования сдваиванием, а в алгоритме скалярных произведений такое суммирование не нужно.

В проведенном обсуждении предполагалось, что число процессоров равно числу столбцов или строк матрицы A. Обычно n и/или m значительно больше, чем число процессоров, и каждому процессору придаются несколько строк или столбцов. Для алгоритма скалярных произведений удобней всего, когда m кратно p и всякому процессору приписаны m/p строк. С математической точки зрения этот алгоритм эквивалентен умножению в блочной форме

$$A\mathbf{x} = \begin{bmatrix} A_1 \\ \cdots \\ A_p \end{bmatrix} \mathbf{x} = \begin{bmatrix} A_1\mathbf{x} \\ \cdots \\ A_p\mathbf{x} \end{bmatrix},$$

где матрица A_i составлена из m/p строк матрицы A. Если в процессор i переданы A_i и $\mathbf{x}$, то произведения $A_1\mathbf{x}$, ..., $A_p\mathbf{x}$

вычисляются с максимальным параллелизмом. Заметим, что не имеет значения, как выполняется умножение $A_i\mathbf{x}$: с помощью алгоритма скалярных произведений или линейного комбинирования столбцов. Однако для системы векторных процессоров можно использовать результаты предыдущего обсуждения. Аналогичные соображения применимы к разбиению A на группы столбцов (упражнение 1.3.5). Если p не является делителем m (или n), то нужно распределить строки (или столбцы) между процессорами по возможности равномерно.

Матричное умножение

Сходные соображения можно применить для задачи умножения матриц A и B. Мы выносим обсуждение параллельных алгоритмов скалярных, внешних и средних произведений в

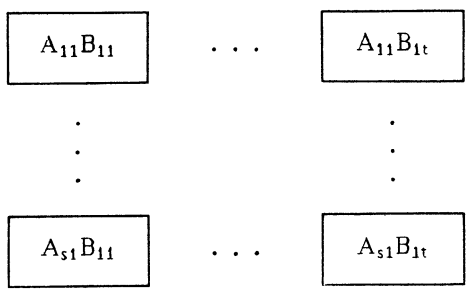


Рис. 1.3.8. Параллельное блочное умножение

упражнение 1.3.6. Возможно также построение алгоритмов, которые, как и (1.3.13), основаны на блочном представлении матриц A и B. Именно, если разбиения обеих матриц на блоки согласованы, то

$$C = AB = \begin{bmatrix} A_{11} & \dots & A_{1r} \\ \cdot & \cdot & \cdot \\ A_{s1} & \dots & A_{sr} \end{bmatrix} \begin{bmatrix} B_{11} & \dots & B_{1t} \\ \cdot & \cdot & \cdot \\ B_{r1} & \dots & B_{rt} \end{bmatrix} = \left(\sum_{k=1}^{r} A_{ik}B_{kj} \right). \quad (1.3.14)$$

Это представление можно реализовать разными алгоритмами. Пусть, например, число процессоров p равно числу st блоков матрицы C. При условии, что A и B приданы соответствующим процессорам, все эти st блоков можно вычислять одновременно (см. упражнение 1.3.7). Специальные случаи: 1) $s = 1$, т. е. матрица A разбита на группы столбцов; 2) $t = 1$, B разбита на группы строк. Еще два интересных специальных случая:

$s = t = 1$, что приводит к алгоритму *блочного скалярного произведения*

$$AB = \sum_{j=1}^{r} A_{1j}B_{j1}, \tag{1.3.15}$$

и $r = 1$, что дает алгоритм *блочного внешнего произведения*

$$AB = (A_{i1}B_{1j}). \tag{1.3.16}$$

Алгоритм (1.3.16) может быть полезен, например, если имеется $p = st$ процессоров, причем блоки A_{i1} и B_{1j} приписаны отдельным процессорам так, как это показано на рис. 1.3.8. Здесь конфигурация процессоров отражает блочную структуру произведения C и все произведения $A_{i1}B_{1j}$ вычисляются одновременно.

Блочные алгоритмы могут быть эффективны и для векторных компьютеров, имеющих быструю локальную память (таких, например, как CRAY-2).

Ленточные матрицы

До сих пор в данном параграфе предполагалось, что матрицы заполнены, т. е. все их элементы или бóльшая их часть ненулевые. Однако многие (если не почти все) матрицы, встре-

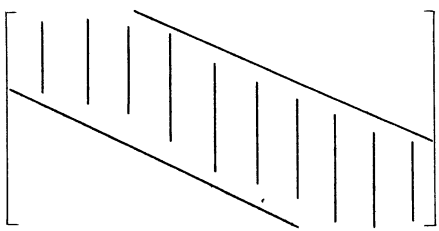


Рис. 1.3.9. Ленточная матрица

чающиеся в практических приложениях, разрежены, т. е. большинство из элементов — нули. Мы исследуем для различных типов разреженных матриц уже известные нам алгоритмы умножения и убедимся, что в некоторых случаях нужны новые алгоритмы. Начнем с ленточных матриц.

Матрица A порядка n называется *ленточной*, если

$$a_{ij} = 0, \quad i - j > \beta_1, \quad j - i > \beta_2, \tag{1.3.17}$$

см. рис. 1.3.9. Для простоты мы будем рассматривать лишь случай матрицы A *с симметричной лентой*, для которой $\beta_1 = \beta_2 = \beta$; число β называется *полушириной* (или просто

шириной) *ленты*. Таким образом, ненулевые элементы в A расположены только на главной диагонали и на 2β диагоналях, прилегающих к главной сверху и снизу. Отметим, что матрица с симметричной лентой не обязана быть симметричной.

Рассмотрим прежде всего матрично-векторное умножение $A\mathbf{x}$. Обсуждавшиеся выше алгоритмы скалярных произведений и линейных комбинаций описываются равенствами

$$A\mathbf{x} = \begin{bmatrix} (\mathbf{a}_1, \mathbf{x}) \\ \cdot \cdot \cdot \\ (\mathbf{a}_n, \mathbf{x}) \end{bmatrix} \tag{1.3.18}$$

и

$$A\mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{a}_i. \tag{1.3.19}$$

В (1.3.18) $\mathbf{a}_i$ обозначает i-ю строку матрицы A, а в (1.3.19) — ее i-й столбец. Если A хранится по столбцам, то длины векторов в (1.3.19) изменяются в пределах от $\beta + 1$ до $2\beta + 1$. Для некоторых классов задач типичными значениями β и n являются $n = 10^4$ и $\beta = 10^2$; в подобных случаях длины векторов в (1.3.19) достаточно велики. С другой стороны, для малых β степень векторизации алгоритма (1.3.19) неудовлетворительна. В предельном случае трехдиагональных матриц, для которых $\beta = 1$, длины векторов не превышают 3.

Аналогичные выводы справедливы для скалярных произведений в (1.3.18), поскольку длины векторов здесь такие же, как и в (1.3.19). Однако по отношению к формированию n скалярных произведений метод (1.3.18) имеет степень параллелизма n. Предположим, например, что в параллельной системе с локальной памятью, состоящей из $p = n$ процессоров, в памяти i-го процессора находятся $\mathbf{a}_i$ и $\mathbf{x}$; тогда все скалярные произведения можно вычислять одновременно. Если число процессоров $p = n/k$, то первые k строк матрицы A могут обрабатываться процессором 1, следующие k строк — процессором 2, и т. д. Таким образом, процессоры будут работать с максимальным параллелизмом с точностью до небольших потерь, связанных с тем, что верхние и нижние строки в A короче средних. Эти потери можно уменьшить за счет более тщательной балансировки нагрузки; то же самое относится к потерям, возникающим при n, не кратном p (см. упражнение 1.3.10). Таким образом, алгоритм скалярных произведений потенциально привлекателен для параллельных систем даже при очень небольших значениях β.

Сходные выводы можно сделать об умножении двух ленточных матриц $n \times n$. Заметим, что если полуширина ленты для

A равна α, а для B равна β, то полуширина ленты для AB в общем случае будет равна $\alpha + \beta$ (упражнение 1.3.11). Рассмотрим вначале алгоритм средних произведений (1.3.8):

$$C = AB = \left(\sum_{j=1}^{\beta+1} b_{j1}\mathbf{a}_j, \ \ldots, \ \sum_{j=n-\beta}^{n} b_{jn}\mathbf{a}_j \right). \tag{1.3.20}$$

Если предположить, что A хранится по столбцам, то длины векторов, участвующих в линейных комбинациях, изменяются в пределах от $1 + \alpha$ до $1 + 2\alpha$. Поэтому, как и прежде, алгоритм может быть полезен при достаточно больших значениях α. Если $\alpha < \beta$, то более выгодной, чем (1.3.20), может оказаться двойственная форма (1.3.9) алгоритма средних произведений, особенно при хранении матрицы B по строкам. Отметим, что и здесь в случае параллельной системы мы имеем возможность вычислять первые k столбцов в (1.3.20) с помощью процессора 1, следующие k столбцов — с помощью процессора 2, и т. д. При такой организации вычислений достигается высокая степень параллелизма; детали этого подхода вынесены в упражнение 1.3.12.

Перейдем теперь к алгоритму внешних произведений (1.3.11):

$$C = AB = \sum_{i=1}^{n} (b_{i1}\mathbf{a}_i, \ \ldots, \ b_{in}\mathbf{a}_i). \tag{1.3.21}$$

Снова предположим, что A хранится по столбцам. Исследование этого алгоритма проводится так же, как и для алгоритма (1.3.20), и длины векторов вновь определяются числом α. Если $\alpha < \beta$, а B хранится по строкам, то, вероятно, следует предпочесть двойственную форму (1.3.12). Проведенное ранее сравнение достоинств алгоритмов (1.3.20) и (1.3.21) в применении к векторным компьютерам с векторными регистрами непосредственно переносится на ленточные матрицы (упражнение 1.3.13). В случае параллельных компьютеров также имеются различные возможности реализации формулы (1.3.21). При наличии $p = n/k$ процессоров в каждом могут формироваться k внешних произведений; это потребует засылки k столбцов матрицы A и k строк матрицы B в память каждого процессора. Затем нужно сложить внешние произведения; когда произведения, сформированные в каждом процессоре, просуммированы в нем, выполняется межпроцессорное сдваивание. Более привлекателен вариант, когда первые k столбцов матрицы C формируются в процессоре 1, следующие k столбцов — в процессоре 2, и т. д. Этот вариант по существу совпадает с алгоритмом средних произведений.

Рассмотрим, наконец, алгоритм скалярных произведений (1.3.7):

$$C = AB = (\mathbf{a}_i \mathbf{b}_j). \qquad (1.3.22)$$

Элемент i, j матрицы C есть скалярное произведение i-й строки матрицы A и j-го столбца матрицы B. Поскольку полуширина ленты для C равна $\alpha + \beta$, то реально вычислять нужно лишь скалярные произведения, соответствующие элементам ленты. Как и в случае матрично-векторного умножения, алгоритм (1.3.22) привлекателен для параллельной организации независимо от значений α и β.

Умножение по диагоналям

Рассмотренные до сих пор алгоритмы неэффективны для векторных компьютеров, если матрицы имеют малую ширину ленты. Это же справедливо в отношении матриц с немногими ненулевыми диагоналями, которые не группируются возле главной. На рис. 1.3.10 показан пример матрицы $n \times n$, лишь четыре диагонали которой содержат ненулевые элементы. Матрицу с относительно небольшим числом ненулевых диагоналей назовем *диагонально разреженной матрицей*. Матрицы этого типа часто встречаются в практических приложениях, особенно при решении эллиптических или параболических уравнений с частными производными конечно-разностными или конечно-элементными методами; подробнее об этом будет сказано в гл. 3.

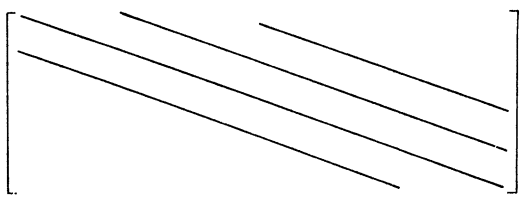
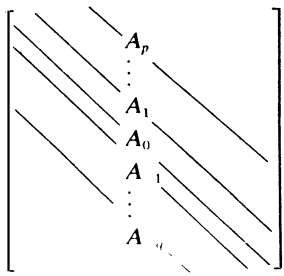


Рис. 1.3.10. Диагонально разреженная матрица

Диагонально разреженные матрицы могут иметь большую ширину ленты, поэтому обсуждавшиеся выше алгоритмы будут для них неэффективны из-за большого процента нулей в каждой строке или столбце. По той же причине неудовлетворительно и хранение таких матриц по строкам или столбцам; естественная для них форма — это хранение по диагоналям. При этом ненулевые диагонали матрицы превращаются в векторы, используемые в алгоритмах умножения, и мы сейчас посмотрим, как нужно организовать это умножение.

Начнем с матрично-векторного умножения $A\mathbf{x}$, где матрица A размера $n \times n$ имеет вид



(1.3.23)

Мы пока не предполагаем, что A имеет симметричную ленту или является диагонально разреженной матрицей; фактически A может быть даже заполнена. В (1.3.23) главная диагональ обозначена через A_0, диагонали ниже главной — через $A_{-1}, \ldots, A_{-q}$, а диагонали выше главной — через $A_1, \ldots, A_p$. Нетрудно показать (упражнение 1.3.14), что вектор Ax можно представить в виде

$$A\mathbf{x} = A_0\mathbf{x} \mathbin{\widehat{+}} A_1\mathbf{x}^2 \mathbin{\widehat{+}} \ldots \mathbin{\widehat{+}} A_p\mathbf{x}^{p+1} + A_{-1}\mathbf{x}_{n-1} \mathbin{\underset{\smile}{+}} \ldots \mathbin{\underset{\smile}{+}} A_{-q}\mathbf{x}_{n-q}, \tag{1.3.24}$$

где

$$\mathbf{x}^j = (x_j, \ldots, x_n), \quad x_{n-j} = (x_1, \ldots, x_{n-j}). \tag{1.3.25}$$

Умножения в формуле (1.3.24) следует понимать как поэлементные произведения векторов A_i (т. е. диагоналей матрицы A) и векторов (1.3.25). Участвующие в (1.3.24) векторы имеют разные длины (например, длина вектора $A_1\mathbf{x}^2$ равна $n-1$), и символ $\widehat{+}$ означает прибавление более короткого вектора к первым компонентам более длинного; так, $A_1\mathbf{x}^2$ прибавляется к первым $n-1$ компонентам вектора $A_0\mathbf{x}$. Аналогично, $\underset{\smile}{+}$ обозначает прибавление более короткого вектора к последним компонентам более длинного.

Обсудим теперь различные специальные случаи формулы (1.3.24) в применении к векторным компьютерам. Если A — заполненная матрица ($p = q = n-1$), то в (1.3.24) будет $2n-1$ векторных произведений; кроме того, длины векторов изменяются от 1 до n. Поэтому для заполненных матриц алгоритм (1.3.24) в сравнении с алгоритмом линейных комбинаций неэффективен. Его стоит рассматривать лишь в том случае, если A уже хранится по диагоналям, но для заполненной матрицы вероятность такого способа хранения невелика.

Предположим, с другой стороны, что матрица A трехдиагональная, т. е. $p = q = 1$. Тогда (1.3.24) принимает вид

$$A\mathbf{x} = A_0\mathbf{x} \mathbin{\hat{+}} A_1\mathbf{x}^2 \mathbin{\underset{\smile}{+}} A_{-1}\mathbf{x}_{n-1}, \qquad (1.3.26)$$

и длины векторов равны n, $n-1$ и $n-1$; следовательно, при больших n достигается почти максимальная степень векторизации. Вообще, алгоритм (1.3.24) весьма привлекателен для матриц с малой шириной ленты, но по мере того, как лента расширяется, его преимущества по сравнению с алгоритмом линейных комбинаций уменьшаются. Существует такое значение β_0 ширины ленты (зависящее от n и конкретного компьютера), что при $\beta < \beta_0$ из двух названных алгоритмов более выгодным является алгоритм (1.3.24), а при $\beta > \beta_0$ — алгоритм линейных комбинаций (см. упражнение 1.3.15).

Алгоритм (1.3.24) очень привлекателен и для диагонально разреженных матриц. Предположим, например, что ненулевыми являются только диагонали A_0, A_1, A_{30}, A_{-1} и A_{-30}. Тогда (1.3.24) превращается в

$$A\mathbf{x} = A_0\mathbf{x} \mathbin{\hat{+}} A_1\mathbf{x}^2 \mathbin{\hat{+}} A_{30}\mathbf{x}^{31} \mathbin{\underset{\smile}{+}} A_{-1}\mathbf{x}_{n-1} \mathbin{\underset{\smile}{+}} A_{-30}\mathbf{x}_{n-30}. \qquad (1.3.27)$$

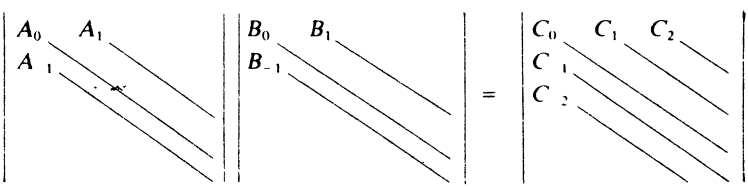
Как и в трехдиагональном случае, $A_1\mathbf{x}^2$ и $A_{-1}\mathbf{x}_{n-1}$ — векторы длины $n-1$, тогда как длины векторов $A_{30}\mathbf{x}^{31}$ и $A_{-30}\mathbf{x}_{n-30}$ равны $n-30$. Если n велико, то эти векторные длины вполне удовлетворительны. С другой стороны, если заменить A_{30} и A_{-30} на A_{n-p} и $A_{-(n-p)}$, то длины векторов для этих диагоналей малы при малом p, поэтому векторные операции $A_{n-p}\mathbf{x}^{n-p+1}$ и $A_{-(n-p)}\mathbf{x}_p$ неэффективны. Однако для такого рода матрицы алгоритм линейных комбинаций тоже не составляет приемлемой альтернативы, и, возможно, при обработке очень коротких векторов нужно переключиться на скалярные операции. Чем больше ненулевых диагоналей в диагонально разреженной матрице, тем больше сближаются эффективности алгоритма (1.3.24) и алгоритма линейных комбинаций. В какой-то момент соображения, связанные со способом хранения матриц, станут решающими при выборе алгоритма.

Для параллельных компьютеров вычисление (1.3.24) можно организовать следующим образом. Ненулевые диагонали матрицы распределяются между процессорами так, чтобы объемы данных, приписанные процессорам, были как можно более сбалансированными. Вектор $\mathbf{x}$ также следует распределить; по крайней мере, каждый процессор должен получить необходимую ему часть этого вектора. Теперь произведения $\mathbf{x}$ и диагоналей можно вычислять с высокой степенью параллелизма, однако в дальнейшем придется выполнять межпроцессорное

сдваивание, и здесь потеря параллелизма неизбежна. Кроме того, в случае использования параллельных компьютеров нет особого стимула для того, чтобы пользоваться умножением по диагоналям, поскольку, как отмечено выше, при малой ширине ленты можно эффективно реализовать алгоритм скалярных произведений. То же самое справедливо для диагонально разреженных матриц. Здесь также можно распределить строки матрицы A между процессорами и выполнять скалярные умножения параллельно. Если система состоит из последовательных процессоров, то в каждом достаточно хранить ненулевые элементы соответствующих строк вместе с индексными списками, указывающими местоположение этих элементов. Таким образом, для параллельных систем умножение, организованное по диагоналям, по-видимому, не имеет особого смысла, если только система не составлена из векторных компьютеров.

Матричное умножение по диагоналям

Матрично-векторное умножение с помощью диагонального алгоритма можно обобщить на случай вычисления произведения AB диагонально хранимых матриц, однако детали алгоритма при этом усложняются. Пусть A и B — матрицы $n \times n$. Рассмотрим для простоты только случай умножения трехдиагональных матриц.



(1.3.28)

Диагонали матрицы C вычисляются по формулам

$$C_0 = A_0B_0 + A_{-1}B_1 + A_1B_{-1}, \quad C_1 = A_0B_1 + A_1B_0^2,$$
$$C_{-1} = A_0^2B_{-1} + A_{-1}B_0, \quad C_2 = A_1B_1^2, \quad C_{-2} = A^2{}_{-1}B_{-1}. \tag{1.3.29}$$

Здесь использовано обозначение A_t^j для вектора, начинающегося с j-й позиции вектора A_t. Мы придерживаемся также соглашения, что если векторы имеют разные длины, то рассматриваются только операции, определяемые более коротким вектором. Например, при формировании диагонали C_1 вектор A_0 имеет длину n, а B_1 — длину $n-1$, поэтому символ A_0B_1 обозначает вектор длины $n-1$, полученный при игнорировании

последнего элемента в A_0. Чтобы пояснить это, рассмотрим следующий пример 4-го порядка:

$$\begin{bmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34} \\ & & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & & \\ b_{21} & b_{22} & b_{23} & \\ & b_{32} & b_{33} & b_{34} \\ & & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ & c_{42} & c_{43} & c_{44} \end{bmatrix}. \tag{1.3.30}$$

Диагонали матрицы C задаются такими формулами:

$$\begin{aligned} & c_{11} = a_{11}b_{11} + a_{12}b_{21}, \\ & c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32}, \\ C_0: \quad & c_{33} = a_{32}b_{23} + a_{33}b_{33} + a_{34}b_{43}, \\ & c_{44} = a_{43}b_{34} + a_{44}b_{44}; \\ & c_{12} = a_{11}b_{12} + a_{12}b_{22}, \\ C_1: \quad & c_{23} = a_{22}b_{23} + a_{23}b_{33}, \\ & c_{34} = a_{33}b_{34} + a_{34}b_{44}; \\ & c_{21} = a_{21}b_{11} + a_{22}b_{21}, \\ C_{-1}: \quad & c_{32} = a_{32}b_{22} + a_{33}b_{32}, \\ & c_{43} = a_{43}b_{33} + a_{44}b_{43}; \\ & c_{13} = a_{12}b_{23}, \\ C_2: \quad & c_{24} = a_{23}b_{34}; \\ & c_{31} = a_{32}b_{21}, \\ C_{-2}: \quad & c_{42} = a_{43}b_{32}. \end{aligned} \tag{1.3.31}$$

Задача составления псевдокода для алгоритма (1.3.28)—(1.3.29) вынесена в упражнение 1.3.16.

Формулы типа (1.3.29) для произвольных ленточных матриц с принципиальной точки зрения вывести несложно, однако детали могут оказаться довольно громоздкими. В упражнении 1.3.17 рассматривается ситуация, когда перемножаются трехдиагональная и пятидиагональная матрицы.

Представление о том, что умножение матриц — простая задача, относится к фольклору параллельных и векторных вычислений. Как мы увидели в данном параграфе, такое представление справедливо лишь в том случае, если должное внимание уделяется структуре матриц и если алгоритмы адаптированы к этой структуре.

Упражнения к параграфу 1.3

1 Найти значения m и n, для которых величина (1.3.5) меньше, чем (1.3.6)

2 Написать псевдокоды, аналогичные псевдокодам на рис 1.3.2 и 1.3.3, для двойственных алгоритмов (1.3.9) и (1.3.12)

3 Матричное умножение $C = AB$, где A и B — матрицы размеров $m \times n$ и $n \times q$ соответственно, может быть описано любым из следующих шести циклов:

Для $i = 1$ до m
  Для $j = 1$ до q
    Для $k = 1$ до n
      форма ijk

Для $j = 1$ до q
  Для $i = 1$ до m
    Для $k = 1$ до n
      форма jik

Для $i = 1$ до m
  Для $k = 1$ до n
    Для $j = 1$ до q
      форма ikj

Для $j = 1$ до q
  Для $k = 1$ до n
    Для $i = 1$ до m
      форма jki

Для $k = 1$ до n
  Для $i = 1$ до m
    Для $j = 1$ до q
      форма kij

Для $k = 1$ до n
  Для $j = 1$ до q
    Для $i = 1$ до m
      форма kji

Во всех случаях тело цикла представляет собой арифметический оператор $c_{ij} = c_{ij} + a_{ik}b_{kj}$. Показать, что формы ijk, ikj и kij отвечают соответственно алгоритмам скалярных, средних и внешних произведений, тогда как формы jki и kji описывают двойственные алгоритмы средних и внешних произведений Показать также, что форма jik отвечает «двойственному алгоритму скалярных произведений», не обсуждавшемуся в основном тексте

4. Пусть векторные регистры имеют длину s. Обсудить изменения, необходимые для алгоритмов внешних и средних произведений на рис. 1.3.4 и 1.3.5 при произвольных размерах m, n и q. Матрицы A и B имеют размеры соответственно $m \times n$ и $n \times q$

5 Сформулировать алгоритм типа алгоритма на рис. 1.3.6 для случая $p \neq n$

6 Обсудить в деталях алгоритмы скалярных, средних и внешних произведений для матричного умножения в применении к параллельной системе из p процессоров

7. Обсудить требования к хранению информации, необходимые для эффективной реализации алгоритма (1.3.14) в случае параллельной системы с $p = st$ процессорами.

8 Пусть A — матрица $m \times n$, а B — матрица $n \times q$ Предположим, что имеется $p = nmq$ процессоров. Указать, какие элементы матриц A и B должны храниться в памяти каждого процессора, чтобы все nmq произведений, необходимых для формирования AB, можно было вычислять одновременно. (Заметим, что это потребует, чтобы одни и те же элементы A и/или B хранились в памяти более чем одного процессора.) Показать далее, что для образования нужных сумм требуется $O(\log n)$ шагов сложения по методу

сдваивания. Вывести отсюда, что матричное умножение можно выполнить за $1 + O(\log n)$ параллельных шагов

9 Сформулировать алгоритм рекурсивного удвоения (см. § 1 2) для сумм и произведений матриц.

10. Пусть параллельная система состоит из p процессоров. Для матрицы A с шириной ленты β обсудить распределение работы между процессорами, обеспечивающее их максимальную загрузку, для алгоритма скалярных произведений (1 3.18) Рассмотреть, в частности, случай, когда n не кратно p.

11. Пусть A и B — матрицы $n \times n$ с полушириной ленты соответственно α и β Показать, что матрица AB в общем случае имеет полуширину ленты $\alpha + \beta$

12. Пусть ленточные матрицы A и B размера $n \times n$ имеют полуширину ленты соответственно α и β Для параллельного компьютера с p процессорами организовать вычисление (1 3 20) так, чтобы добиться максимального параллелизма

13. Сравнить достоинства алгоритмов (1.3 20) и (1 3 21) в применении к векторным компьютерам, использующим векторные регистры и а) допускающим совмещение арифметики только с загрузкой или только с записью в память; b) допускающим совмещение арифметики и с загрузкой, и с записью в память

14 Проверить равенство (1 3.24)

15. Определить значение β для полуширины ленты, при котором алгоритм (1 3 24) матрично-векторного умножения теряет преимущество перед алгоритмом линейных комбинаций Считать, что A имеет симметричную ленту, схема хранения удобна для обоих алгоритмов и затраты времени на векторные команды описываются формулой $T = (1000 + 10n)$ нс.

16. Написать псевдокод для умножения двух трехдиагональных матриц (1 3 28) с использованием векторных операций (1 3 29) Считать, что A хранится в виде линейного массива A_0, A_1, A_{-1}, матрица B — в виде линейного массива B_0, B_1, B_{-1}, а C — в виде линейного массива C_0, C_1, C_2, C_{-1}, C_{-2}.

17 Вывести формулы, аналогичные формулам (1 3 31), для произведения $C = AB$, где A — трехдиагональная матрица $n \times n$, B — пятидиагональная матрица $n \times n$ (ненулевыми являются главная диагональ и по две диагонали, прилегающие к главной с каждой стороны)

Литература и дополнения к параграфу 1.3

1. Более подробное обсуждение многих алгоритмов, описанных в этом параграфе, можно найти в книге [Hockney, Jesshope, 1981], где и вводится термин «алгоритм средних произведений»

2 Различные формы циклов, показанные на рис. 1 3 1 и в упражнении 1 3.3, приводятся в статье [Dongarra, Gustavson, Karp, 1984] В этой статье также обсуждаются конфликты в банках памяти компьютеров CRAY-1, которые могут вызвать ухудшение временных характеристик при вычислении скалярных произведений, зацеплении с записями в память, и т д. Здесь же введен термин gaxpy (generalized saxpy — обобщенная операция saxpy) для обозначения линейных комбинаций векторов

3. В статье [Sorensen, 1984] отмечается, что вычисление суммы $\mathbf{y} + A\mathbf{x}$ с помощью представления $\mathbf{y} + \Sigma x_i \mathbf{a}_i$ может быть не очень удачным способом для некоторых параллельных машин вследствие необходимости синхронизо-

вать доступ к y. В этой работе обсуждается также возможность повышения производительности матрично-векторного умножения за счет буферизации.

4 Термины «матрица с симметричной лентой» и «диагонально разреженная матрица» не являются стандартными, однако для матриц этих типов нет общепринятых терминов.

5 Алгоритм умножения по диагоналям был предложен в работе [Madsen et al., 1976], там этот процесс разобран очень детально. Еще одно достоинство диагональной схемы хранения матрицы состоит в том, что транспонирование осуществляется гораздо проще, чем если бы матрица хранилась по строкам или столбцам. В самом деле, транспонированную матрицу можно получить простым изменением указателей к первым элементам диагоналей; в расположении матричных элементов никаких изменений делать не нужно.

6 В статьях [McBryan, van de Velde, 1985, 1986a, 1987] рассматриваются несколько подходов к организации матричного умножения в параллельных машинах. В работе [1986b] те же авторы вводят схему хранения, в которой $2n-1$ диагоналей матрицы $n \times n$ упакованы в n одномерных массивов путем «циклического склеивания»

7. В работах [Melhem, 1987a, b] рассматривается интересное обобщение диагоналей, которое автор называет *полосами*. Грубо говоря, полосы — это «изогнутые» диагонали. Автор строит алгоритмы матричного умножения, основанные на схеме хранения такого типа

8 В статье [Hayes, Devloo, 1986] описан алгоритм матрично-векторного умножения, предназначенный для матриц вида $A = \sum A_i$. (При этом основной мотивацией является случай, когда A_i соответствуют матрицам отдельных элементов в конечноэлементной задаче.)

9. В работе [Jalby, Meier, 1986] рассматривается задача эффективного использования двухуровневой памяти (оперативная память плюс быстрая локальная или кэш-память) векторных машин. Детально исследуется матричное умножение по блокам, где размеры блоков выбираются так, чтобы минимизировать общее время. Авторы подходят к этой оптимизационной задаче с позиций раздельной минимизации времени вычислений и времени, затрачиваемого на операции с памятью, а затем находят компромиссное решение. Другой блочно ориентированный алгоритм, рассчитанный главным образом на архитектуру гиперкуба, приводится в статье [Fox, Otto, Hey, 1987].

10 Алгоритмы умножения ленточных матриц для нескольких типов систолических массивов рассматриваются в статье [Cheng, Sahni, 1987]

11. Параллельные и векторные алгоритмы умножения разреженных матриц общего вида принципиально более сложны, чем алгоритмы, обсуждавшиеся нами в основном тексте. Некоторое представление об этой задаче можно получить из статей [Reed, Patrick, 1984, 1985a, b].

12. Развертывание циклов обсуждается в [Dongarra, Hinds, 1979]. Как отмечается в этой работе, техника развертывания циклов полезна и для последовательных компьютеров, поскольку уменьшает накладные расходы на организацию циклов и дает некоторые другие преимущества. Более общее обсуждение этой техники в контексте набора инструментальных средств для преобразования циклов DO содержится в статье [Cowell, Thompson, 1986].

13 Пакет BLAS представляет собой набор подпрограмм для базисных операций линейной алгебры, скажем, таких, как скалярное произведение. Этот набор постоянно расширяется за счет включения операций все более высо-

кого уровня, например матрично-векторного умножения. Более подробную информацию по этому поводу и указания на литературу можно найти в статье [Dongarra, DuCroz et al., 1986], а сведения о времени работы BLAS-подпрограмм для машины CYBER 205 — в статье [Louter-Nool, 1987]. Использование BLAS-подпрограмм облегчает конструирование переносимого программного обеспечения. Еще один возможный подход к проблеме переносимости изложен в работе [Dongarra, Sorensen, 1987].

14. В статье [Dave, Duff, 1987] приводятся данные о времени матрично-векторного и матрично-матричного умножения для компьютера CRAY-2. Сообщается, что при перемножении двух матриц порядка 500 достигается производительность 432 мегафлопа на одном процессоре.

Глава 2

Прямые методы решения линейных систем

Рассмотрим теперь вопрос о том, как организовать решение систем линейных уравнений прямыми методами. В § 2.1 и 2.2 предполагается, что матрица коэффициентов заполнена, и изучаются гауссово исключение, разложение Холесского и методы ортогонального приведения Гивенса и Хаусхолдера. В § 2.1 рассматриваются только векторные компьютеры, а затем в § 2.2 те же самые основные алгоритмы исследуются применительно к параллельным компьютерам. В § 2.3 эти и некоторые другие алгоритмы применяются к ленточным системам.

2.1. Прямые методы для векторных компьютеров

Рассмотрим систему линейных уравнений

$$A\mathbf{x} = \mathbf{b} \tag{2.1.1}$$

с невырожденной матрицей A размера $n \times n$. В этом параграфе мы считаем A заполненной матрицей и рассматриваем только векторные компьютеры.

Гауссово исключение и LU-разложение

Наиболее известной формой гауссова исключения является та, в которой система (2.1.1) приводится к верхнетреугольному виду путем вычитания одних уравнений, умноженных на подходящие числа, из других уравнений; полученная треугольная система решается с помощью обратной подстановки. Математически все это эквивалентно тому, что вначале строится разложение $A = LU$, где L — нижнетреугольная матрица с единицами на главной диагонали, а U — верхнетреугольная матрица. Затем решаются треугольные системы

$$L\mathbf{y} = \mathbf{b}, \quad U\mathbf{x} = \mathbf{y}. \tag{2.1.2}$$

Процесс их решения называется *прямой* и *обратной подстановками.*

Мы сосредоточимся вначале на LU-разложении, поглощающем бо́льшую часть времени всего процесса, а затем вернемся к решению треугольных систем. Псевдокод разложения приведен на рис. 2.1.1.

В цикле j на рис. 2.1.1 кратные k-й строки текущей матрицы A вычитаются из расположенных ниже строк. Эти операции представляют собой триады, в которых векторами являются строки матрицы A. Поэтому в случае использования векторных компьютеров, требующих, чтобы векторы располагались в последовательно адресуемых ячейках памяти, предполагается, что A хранится по строкам. Посредством триад осуществляется *модификация* (или *пересчет*) строк матрицы A; на это приходится основная часть работы в LU-разложении. На k-м шаге алгоритма выполняется $n-k$ векторных операций и длины векторов равны $n-k$. Следовательно, средняя длина векторов при модификации равна

$$\frac{(n-1)(n-1)+(n-2)(n-2)+\ldots+1}{n-1+n-2+\ldots+1}=O(2n/3). \quad (2.1.3)$$

Проверка этого равенства составляет содержание упражнения 2.1.1. (Здесь и далее мы пользуемся более или менее общепринятым обозначением $O(cn^p)$, указывающим член наивысшего порядка в выражении; во многих случаях именно константа c играет главную роль.) Таким образом, средняя степень векторизации векторных команд равна $O(2n/3)$. Скалярные деления при формировании множителей l_{ik} на каждом шаге несколько уменьшают эту среднюю характеристику. Для некоторых векторных компьютеров может оказаться выгодным переход с векторной на скалярную арифметику при снижении длин векторов до некоторого рубежного значения, хотя с таким переходом связано усложнение программы.

Если требуется выбор главного элемента, то это еще сильнее уменьшает скорость. При использовании стратегии частичного выбора мы должны на первом шаге просмотреть первый столбец в поисках максимального по модулю элемента. Хотя в некоторых векторных компьютерах имеются векторные команды, облегчающие такой поиск, в данном случае элементы первого столбца находятся не в соседних ячейках памяти. Поэтому при поиске нужно применять скалярные операции или операцию сборки, сопровождаемую векторной операцией взятия максимума. Как только положение максимального элемента определено, соответствующую строку можно переставить с первой строкой с помощью векторной операции или изменения индексации. Как именно реализуется стратегия выбора главного элемента, зависит от конкретного векторного компьютера.

Правая часть **b** системы (2.1.1) также может обрабатываться в ходе приведения к треугольному виду, благодаря чему осуществляется этап прямой подстановки в равенствах (2.1.2). Если A хранится по строкам, то полезно присоединить b_i к i-й строке, если это позволяет память. В таком случае в цикле j на рис. 2.1.1 верхняя граница увеличивается до $n+1$, и длины векторов возрастают на единицу.

Если A хранится по столбцам, то мы изменим алгоритм LU-разложения, как это показано на рис. 2.1.2. На k-м шаге измененного алгоритма сначала формируется k-й столбец матрицы L; это достигается векторной операцией деления. В самом внутреннем цикле (цикле по i) k-й столбец L, умноженный на число, вычитается из j-го столбца текущей матрицы A; длина

```
Для k = 1 до n − 1
    Для i = k + 1 до n
        l_ik = a_ik/a_kk
        Для j = k + 1 до n
            a_ij = a_ij − l_ik a_kj
```

Рис. 2.1.1. Строчно ориентированная схема LU-разложения

```
Для k = 1 до n − 1
    Для s = k + 1 до n
        l_sk = a_sk/a_kk
    Для j = k + 1 до n
        Для i = k + 1 до n
            a_ij = a_ij − l_ik a_kj
```

Рис. 2.1.2. Столбцово ориентированная схема LU-разложения

столбцов равна $n-k$. Таким образом, основной векторной операцией снова является триада, но теперь в качестве векторов выступают столбцы матрицы L и текущей матрицы A. Правая часть **b** может обрабатываться таким же образом, как столбцы в A; при этом осуществляется прямая подстановка (см. (2.1.2)).

Если не считать формирования вектора множителей и обработки правой части, то на k-м шаге мы снова имеем $n-k$ триад с векторами длины $n-k$. Поэтому (2.1.3) и теперь дает среднюю длину векторов, и средняя степень векторизации попрежнему равна $O(2n/3)$. Только более детальный анализ для конкретной машины (см., например, упражнение 2.1.2) может показать, будет ли строчная форма алгоритма эффективней столбцовой, однако то обстоятельство, что множители формируются посредством векторной операции, способно склонить выбор в пользу столбцовой формы. Решающим фактором здесь является способ хранения матрицы A.

В то же время внедрить выбор главного элемента в столбцовую форму более сложно. Для машин, допускающих выбор

максимума в векторе, поиск главного элемента можно осуществить с помощью векторных операций, однако перестановка строк становится проблемой. Элементы строк можно переставить с помощью векторных операций, имеющих шаг n, или операций сборки, сопровождаемых векторными операциями, или, наконец, скалярных операций. Следовательно, если требуется выбор главного элемента и A уже хранится по строкам, то строчно ориентированный алгоритм, вероятно, будет предпочтительным.

Машины типа «регистр — регистр»

Проведенное только что обсуждение применимо и к векторным машинам типа «память — память», и к машинам типа «регистр — регистр». Однако для машин последнего типа необходимо некоторое дополнение. Кроме того, для таких машин желателен иной подход к организации LU-разложения.

Исследуем вначале характер обменов во внутреннем цикле столбцового алгоритма на рис. 2.1.2. Для простоты предположим, что столбец матрицы A полностью вкладывается в векторный регистр (см. упражнение 2.1.4), и начнем с рассмотрения случая, когда на фоне вычислений может выполняться только загрузка или только запись в память. Несколько первых операций указаны в следующем списке:

Сформировать первый столбец матрицы L. (2.1.4a)

Загрузить второй столбец матрицы A. (2.1.4b)

Модифицировать второй столбец матрицы A;

загрузить третий столбец матрицы A. (2.1.4c)

Записать в память модифицированный второй столбец. (2.1.4d)

Модифицировать третий столбец;

загрузить четвертый столбец матрицы A. (2.1.4e)

. . . . . .

Согласно (2.1.4c), загрузка следующего столбца матрицы A совмещается с модификацией текущего столбца. Но затем возникает задержка при записи модифицированного второго столбца из регистра в память.

Мы можем модифицировать алгоритм, чтобы устранить задержку, вызванную записью в память (2.1.4d). Новый алгоритм схож с алгоритмом средних произведений для матричного умножения; идея состоит в том, чтобы выполнить всю необходимую обработку для j-го столбца, прежде чем перейти к

$(j+1)$-му столбцу. Таким образом, обработка каждого из остальных столбцов матрицы A откладывается до тех пор, пока не наступит время придать этому столбцу окончательный вид. Псевдокод данного алгоритма приведен на рис. 2.1.3. Опишем несколько первых операций j-го шага вычислений, которые показывают характер обменов с памятью:

Загрузить первый столбец матрицы L. (2.1.5a)

Загрузить j-й столбец матрицы A. (2.1.5b)

Модифицировать j-й столбец матрицы A;
загрузить второй столбец матрицы L. (2.1.5c)

Модифицировать j-й столбец матрицы A;
загрузить третий столбец матрицы L. (2.1.5d)

. . . . . .

Заметим, что в алгоритме (2.1.5) не производится записей в память, пока вся работа с j-м столбцом матрицы A не завершена. Столбцы матрицы L все время должны загружаться в регистры, но эти загрузки идут на фоне вычислений. Только в начале и в конце каждого шага происходят задержки для

```
Для j = 2 до n
    Для s = j до n
        l_{s,j-1} = a_{s,j-1} / a_{j-1,j-1}
    Для k = 1 до j - 1
        Для i = k + 1 до n
            a_{ij} = a_{ij} - l_{ik} a_{kj}
```

Рис. 2.1.3. Столбцово ориентированная схема LU-разложения с отложенными модификациями

загрузок и/или записей. Вполне вероятно, как отмечалось и в § 1.3, что транслятор не сумеет распознать возможности оставить текущий j-й столбец в регистре; тогда результат, требуемый от алгоритма на рис. 2.1.3, либо достигается переходом к программированию на языке ассемблера, либо аппроксимируется путем развертывания циклов. Еще одна потенциальная проблема при реализации данного алгоритма заключается в том, что длины векторов при модификациях непостоянны: на j-м шаге мы модифицируем j-й столбец, используя $n-1$ последних элементов столбца 1, $n-2$ последних элементов столбца 2 и т. д.

Алгоритм с отложенными модификациями не столь нужен для тех машин типа «регистр — регистр», в которых допускается совмещение с арифметикой и загрузок и записей в память. В этом случае операцию (2.1.4d) можно было бы удалить, а операцию (2.1.4c) заменить операцией

Модифицировать третий столбец A;
загрузить четвертый столбец A;
записать в память второй столбец A.

Таким образом, запись в память второго столбца матрицы A происходит одновременно с загрузкой четвертого столбца. Заметим, что и для машин типа «память — память» алгоритм с отложенными модификациями не обязателен, хотя даже здесь он может иметь некоторые преимущества.

LU-разложение и его *ijk*-формы

Полезно представить рассмотренные выше алгоритмы в виде троек циклов Для (For), как это было сделано в § 1.3 для задачи матричного умножения:

$$\begin{aligned} &\text{Для} \quad \_\_\_\_\_ \\ &\quad \text{Для} \quad \_\_\_\_\_ \\ &\quad\quad \text{Для} \quad \_\_\_\_\_ \\ &\quad\quad\quad a_{ij} = a_{ij} - l_{ik}a_{kj} \end{aligned} \tag{2.1.6}$$

Эта запись отражает основную арифметическую операцию, одну и ту же для всех форм (мы опускаем формирование множителей l_{ik}), и три цикла Для, где пропуски нужно заполнить некоторыми перестановками индексов i, j, k. Например, последовательность

$$\begin{aligned} &\text{Для } k = 1 \text{ до } n-1 \\ &\quad \text{Для } i = k+1 \text{ до } n \\ &\quad\quad \text{Для } j = k+1 \text{ до } n \end{aligned} \tag{2.1.7}$$

соответствует алгоритму на рис. 2.1.1, тогда как последовательность

$$\begin{aligned} &\text{Для } j = 2 \text{ до } n \\ &\quad \text{Для } k = 1 \text{ до } j-1 \\ &\quad\quad \text{Для } i = k+1 \text{ до } n \end{aligned} \tag{2.1.8}$$

— алгоритму на рис. 2.1.3. Алгоритм, приведенный на рис. 2.1.2, соответствует форме kji. Отметим, что алгоритмы на рис. 2.1.1—

2.1.3 в действительности не так просты, как можно заключить, рассматривая схему (2.1.6). Это тем более верно в отношении трех остальных алгоритмов, подробно разбираемых в приложении 1.

Сформулируем теперь некоторые свойства *ijk*-форм, а за дальнейшими подробностями отошлем читателя к приложению 1. Первое важное свойство состоит в том, что базисные векторные операции, определяемые самыми внутренними циклами, различны для разных форм. Для форм *kji* и *kij* векторной

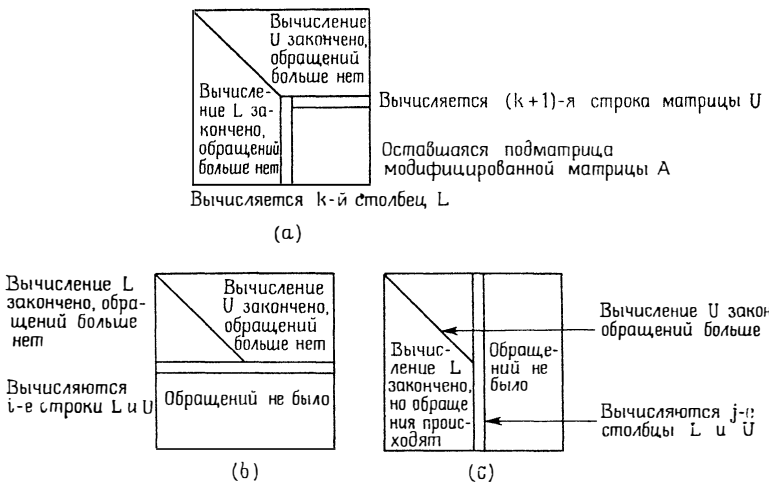


Рис. 2.1.4. Способ доступа к данным для *ijk*-форм *LU*-разложения: a) формы *kij* и *kji*; b) формы *ikj* и *ijk*, c) формы *kij* и *jik*

операцией является триада. В этих формах строки (или столбцы) модифицируются сразу после того, как подготовлены множители, и поэтому мы назовем также алгоритмы *алгоритмами безотлагательной модификации*. Для формы *jki* (см. рис. 2.1.3) столбцы модифицируются непосредственно перед аннулированием элементов в них; форма *ikj* представляет собой соответствующую строчно ориентированную версию. Эти алгоритмы будем называть *алгоритмами отложенной модификации*. В качестве векторной операции снова выступает триада, но теперь она выполняется многократно при формировании линейной комбинации. Формы *ijk* и *jik* также соответствуют алгоритмам отложенной модификации, но векторной операцией здесь будет скалярное произведение.

Шесть различных форм схемы (2.1.6) различаются и способом доступа к элементам матриц A и L. Формы *kji* и *jki*

столбцово ориентированы: они обращаются к столбцам и в A, и в L. В формах kij и ikj происходят обращения к строкам матрицы A и лишь к отдельным элементам в L. Формам ijk и jik свойствен смешанный доступ: к столбцам в A и строкам в L. Поскольку предполагается, что L замещает нижнюю треугольную часть матрицы A, то при такой смешанной стратегии требуется доступ к векторам с шагом, бо́льшим 1. Способ выборки элементов показан на рис. 2.1.4; отмечено, к каким частям матрицы нужен доступ на данном шаге.

Таблица 2.1.1. LU-разложение: ijk-формы

Форма	Операция	Модификация	Способ доступа к A	Способ доступа к L
kij	триада	безотлагательная	строка	отдельные элементы
kji	триада	безотлагательная	столбец	столбец
ikj	линейная комбинация	отложенная	строка	отдельные элементы
jki	линейная комбинация	отложенная	столбец	столбец
ijk	скалярное произведение	отложенная	столбец	строка
jik	скалярное произведение	отложенная	столбец	строка

В таблице 2.1.1 суммированы основные свойства ijk-форм. Едва ли последние две формы, ijk и jik, представляют интерес для векторных компьютеров, но все четыре остальные при подходящих обстоятельствах играют свою роль.

Треугольные системы

По окончании этапа приведения в гауссовом исключении мы должны решить треугольную систему уравнений

$$\begin{bmatrix} u_{11} & & \cdots & u_{1n} \\ & u_{22} & & \vdots \\ & & \ddots & \vdots \\ & & & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ \vdots \\ c_n \end{bmatrix}. \qquad (2.1.9)$$

Обычный алгоритм обратной подстановки описывается формулами

$$x_i = (c_i - u_{i,\,i+1}x_{i+1} - \ldots - u_{in}x_n)/u_{ii},\ i = n,\ \ldots,\ 1. \quad (2.1.10)$$

Рассмотрим, как он может быть реализован в векторных операциях. Если U хранится по строкам (так будет, если на этапе приведения A хранилась по строкам), то формулы (2.1.10) задают скалярные произведения с длинами векторов, меняющимися от 1 до $n-1$, и n скалярных делений. Если игнорировать деления, то средняя степень векторизации равна $O(n/2)$.

```
Для j = n до 1 с шагом −1
    x_j = c_j/u_jj
    Для i = j − 1 до 1 с шагом −1
        c_i = c_i − x_j u_ij
```

Рис. 2.1.5. Столбцовый алгоритм

Альтернативный алгоритм, полезный, если U хранится по столбцам, записан в виде псевдокода на рис. 2.1.5. Он называется *столбцовым алгоритмом* (или *алгоритмом векторных сумм*). Как только найдено x_n, вычисляются и вычитаются из соответствующих элементов c_i величины $x_n u_{in}$ $(i = 1,\ \ldots\ \ldots,\ n-1)$; таким образом, вклад, вносимый x_n в прочие компоненты решения, реализуется до перехода к следующему шагу. Шаг с номером i состоит из скалярного деления, сопровождаемого триадой длины $i-1$ (подразумевается, что шаги нумеруются в обратном порядке: $n,\ n-1,\ \ldots,\ 2,\ 1$. — *Перев.*). Поэтому средняя степень векторизации снова равна $O(n/2)$, только векторными операциями теперь являются триады. Какой из двух алгоритмов выбрать, диктуется способом хранения матрицы U, если он был определен LU-разложением.

И алгоритм скалярных произведений, и столбцовый алгоритм легко переформулировать на случай нижнетреугольных систем (см. упражнение 2.1.5).

Матрично-векторные формы *LU*-разложения

Хотя *ijk*-формы дают шесть различных способов организации LU-разложения, имеются и другие способы, потенциально полезные для векторных компьютеров. Даже тогда, когда та или иная *ijk*-форма теоретически пригодна для конкретной векторной машины, при ее реализации могут возникнуть

проблемы, особенно если применяется язык высокого уровня. Разбираемые ниже способы организации вычислений основаны на операциях с подматрицами; потенциально они проще реализуются и облегчают написание переносимых программ.

В основе простейшего из этих способов организации лежит идея *окаймления*. Обозначим через A_j ведущую главную подматрицу порядка j в матрице A, и пусть известно разложение

```
Для j = 2 до n                          Для i = 2 до n
    Для k = 1 до j − 2                      Для j = 2 до i
        Для i = k + 1 до j − 1              l_{i,j−1} = a_{i,j−1} / a_{j−1,j−1}
            a_ij = a_ij − l_ik a_kj         Для k = 1 до j − 1
    Для k = 1 до j − 1                          a_ij = a_ij − l_ik a_kj
        l_jk = a_jk / a_kk                  Для j = 2 до i − 1
        Для i = k + 1 до j                      Для k = 1 до j − 1
            a_ji = a_ji − l_jk a_ki                 a_ji = a_ji − l_jk a_ki
```

Рис. 2.1.6. Алгоритмы окаймления для LU-разложения: a) столбцовый алгоритм; b) алгоритм скалярных произведений

$L_{j-1}U_{j-1}$ подматрицы A_{j-1}. Тогда для разложения подматрицы A_j имеем

$$\begin{bmatrix} L_{j-1} & 0 \\ \mathbf{l}_j & 1 \end{bmatrix} \begin{bmatrix} U_{j-1} & \mathbf{u}_j \\ 0 & u_{jj} \end{bmatrix} = \begin{bmatrix} A_{j-1} & \mathbf{a}_j \\ \hat{\mathbf{a}}_j & a_{jj} \end{bmatrix},$$

где $\mathbf{l}_j$ и $\hat{\mathbf{a}}_j$ — векторы-строки, а $\mathbf{u}_j$ и $\mathbf{a}_j$ — векторы-столбцы. Это приводит к соотношениям

$$L_{j-1}\mathbf{u}_j = \mathbf{a}_j, \quad \mathbf{l}_j U_{j-1} = \hat{\mathbf{a}}_j, \quad \mathbf{l}_j \mathbf{u}_j + u_{jj} = a_{jj}. \tag{2.1.11}$$

Таким образом, векторы $\mathbf{u}_j$ и $\mathbf{l}_j$ можно вычислить, решая нижнетреугольные системы

$$L_{j-1}\mathbf{u}_j = \mathbf{a}_j, \quad U_{j-1}^{\mathrm{T}}\mathbf{l}_j^{\mathrm{T}} = \hat{\mathbf{a}}_j^{\mathrm{T}}, \tag{2.1.12}$$

после чего u_{jj} определяется из третьего равенства (2.1.11).

Существуют два естественных способа реализации окаймления в LU-разложении. В первом варианте треугольные системы решаются с помощью столбцового алгоритма, во втором — с помощью алгоритма скалярных произведений. Псевдокоды этих двух вариантов приведены на рис. 2.1.6. В первом цикле по i на рис. 2.1.6a выполняется модификация j-го столбца

матрицы A и тем самым вычисляется j-й столбец матрицы U. Во втором цикле по i модифицируется j-я строка матрицы A и вычисляется j-я строка матрицы L. Заметим, что при $i = j$ во втором цикле по i пересчитывается элемент (j, j) матрицы A; в результате, согласно (2.1.11), получается элемент u_{jj}.

Во второй форме алгоритма окаймления (см. рис. 2.1.6b) первый цикл по j, k вычисляет i-ю строку матрицы L, для чего из элементов a_{ij} вычитаются скалярные произведения строк L со столбцами U. Вместе с делениями это эквивалентно решению системы $U_{i-1}^{\mathrm{T}}\mathbf{l}_i^{\mathrm{T}} = \widehat{\mathbf{a}}_i^{\mathrm{T}}$. Отметим, что при $j = i$ модифицируется элемент (i, i) матрицы A, и это относится уже к вычис-

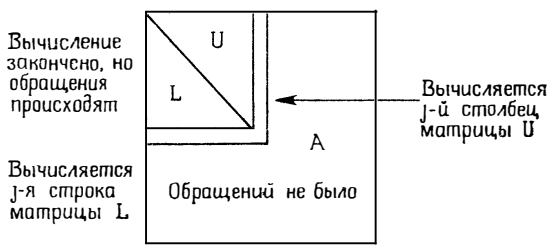


Рис. 2.1.7. Доступ к данным в алгоритмах окаймления

лению i-го столбца матрицы U. Во втором цикле по j, k модифицируется i-й столбец A — опять-таки путем вычитания скалярных произведений строк L и столбцов U. Это эквивалентно решению треугольной системы $L_{i-1}\mathbf{u}_i = \mathbf{a}_i$ относительно i-го столбца матрицы U.

В обеих формах окаймления обращения к данным производятся одинаково, что показано на рис. 2.1.7. Обратим внимание, что в обоих случаях требуется доступ и к строкам, и к столбцам матрицы A. Поэтому алгоритмы будут неэффективны для векторных компьютеров, требующих, чтобы элементы вектора находились в смежных позициях памяти.

Основная работа в алгоритмах окаймления приходится на решение треугольных систем (2.1.12). Это матрично-векторные операции, которые можно реализовать в виде подпрограмм, добиваясь в них максимальной для данной машины эффективности. Еще один способ организации вычислений, который мы назовем *алгоритмом Донгарры — Айзенштата*, имеет то преимущество, что его основной операцией является матрично-векторное умножение. Математически алгоритм можно описать следующим образом. Пусть A, L и U разбиты на блоки

в соответствии с равенством

$$\begin{bmatrix} A_{11} & \mathbf{a}_{12} & A_{13} \\ \mathbf{a}_{21} & a_{22} & \mathbf{a}_{23} \\ A_{31} & \mathbf{a}_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} L_{11} & & \\ \mathbf{l}_{21} & 1 & \\ L_{31} & \mathbf{l}_{32} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & \mathbf{u}_{12} & U_{13} \\ & u_{22} & \mathbf{u}_{23} \\ & & U_{33} \end{bmatrix} \quad (2.1.13)$$

Здесь $\mathbf{a}_{21}$, $\mathbf{a}_{23}$, $\mathbf{l}_{21}$ и $\mathbf{u}_{23}$ — векторы-строки, а $\mathbf{a}_{12}$, $\mathbf{a}_{32}$, $\mathbf{l}_{32}$ и $\mathbf{u}_{12}$ — векторы-столбцы. Предположим, что матрица A_{11} разложена в произведение $L_{11}U_{11}$ и что $\mathbf{l}_{21}$, L_{31}, $\mathbf{u}_{12}$ и U_{13} известны. На очередном шаге мы хотим вычислить $\mathbf{l}_{32}$, следующий столбец матрицы L, а также u_{22} и $\mathbf{u}_{23}$, составляющие следующую строку в U. Приравнивая в (2.1.13) одинаково расположенные элементы, получаем

$$u_{22} = a_{22} - \mathbf{l}_{21}\mathbf{u}_{12}, \quad \mathbf{u}_{23} = \mathbf{a}_{23} - \mathbf{l}_{21}U_{13}, \quad \mathbf{l}_{32} = (\mathbf{a}_{32} - L_{31}\mathbf{u}_{12})/u_{22}. \quad (2.1.14)$$

Характер доступа к данным при таком вычислении показан на рис. 2.1.8.

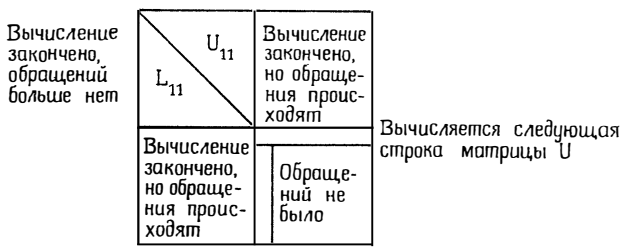


Рис. 2.1.8. Доступ к данным в алгоритме Донгарры — Айзенштата

Основной операцией в (2.1.14) является умножение вектора на прямоугольную матрицу. Мы можем реализовать такие умножения посредством скалярных произведений или линейных комбинаций, что приводит к двум различным формам алгоритма, показанным на рис. 2.1.9. Первый цикл по k, j на рис. 2.1.9а производит последовательные модификации i-й строки матрицы A, которая по окончании цикла k превращается в i-ю строку матрицы U. Эти модификации можно рассматривать как вычисление векторно-матричного произведения $\mathbf{l}_{21}U_{13}$ в (2.1.14) с помощью линейных комбинаций строк U. Заметим, что при $j = i$ результатом модификаций будет первая величина в (2.1.14). Во втором цикле по k, j выполняются модификации i-го столбца матрицы A, что можно интерпретировать как вычисление матрично-векторного произведения $L_{31}\mathbf{u}_{12}$ в (2.1.14) посредством линейных комбинаций столбцов матрицы L. Об-

ратим внимание на то, что, в отличие от алгоритма отложенных модификаций (рис. 2.1.3), теперь длины векторов, участвующих в линейных комбинациях, одинаковы. Второй оператор с индексом k можно удалить, и программа по-прежнему будет верна; мы вставили этот оператор, чтобы подчеркнуть наличие линейных комбинаций. Отметим еще, что в первом цикле по k, j происходят обращения к строкам матрицы A, а во втором цикле по k, j — к ее столбцам. Следовательно, эта форма неэффективна для векторных компьютеров, требующих размещения элементов вектора в смежных ячейках памяти.

```
Для i = 1 до n
    Для k = 1 до i − 1
        Для j = i до n
            a_ij = a_ij − l_ik a_kj
    Для k = 1 до i − 1
        Для j = i + 1 до n
            a_ji = a_ji − l_jk a_ki
    Для s = i + 1 до n
        l_si = a_si / a_ii
```

```
Для j = 1 до n
    Для i = j + 1 до n
        Для k = 1 до j − 1
            a_ij = a_ij − l_ik a_kj
    Для i = j до n
        Для k = 1 до j − 1
            a_ji = a_ji − l_jk a_ki
    Для s = j + 1 до n
        l_sj = a_sj / a_jj
```

Рис. 2.1.9. Алгоритмы LU-разложения Донгарры — Айзенштата: a) алгоритм линейных комбинаций; b) алгоритм скалярных произведений

Алгоритм на рис. 2.1.9b использует скалярные произведения. На j-м шаге первый цикл по i, k вычисляет, с точностью до финального деления, j-й столбец матрицы L; с этой целью j-й столбец в A модифицируется посредством скалярных произведений строк L и j-го столбца U. Во втором цикле по i, k вычисляется j-я строка матрицы U, для чего j-я строка в A модифицируется посредством скалярных произведений j-й строки L и столбцов U. Снова требуется доступ к строкам и столбцам матрицы A. Потенциальное преимущество алгоритма Донгарры — Айзенштата заключается в том, что в некоторых векторных компьютерах матрично-векторные умножения выполняются весьма эффективно.

Разложение Холесского

Если A — симметричная положительно определенная матрица, то часто используемой альтернативой гауссову исключению является разложение Холесского

$$A = LL^{\mathrm{T}}, \tag{2.1.15}$$

где L — нижнетреугольная матрица. Чтобы завершить решение линейной системы $Ax = b$, нужно выполнить прямую и обратную подстановки

$$L\mathbf{y} = \mathbf{b}, \quad L^{\mathrm{T}}\mathbf{x} = \mathbf{y}. \tag{2.1.16}$$

Одна из возможных реализаций разложения (2.1.15) показана на рис. 2.1.10. Поскольку A — симметричная матрица, достаточно хранить лишь нижнюю (или верхнюю) ее треугольную часть. В самом внутреннем цикле (цикле по i) на рис. 2.1.10 модифицируются столбцы матрицы A путем вычитания из них

$l_{11} = a_{11}^{1/2}$
Для $j = 2$ до n
    Для $s = j$ до n
        $l_{s,\,j-1} = a_{s,\,j-1}/l_{j-1,\,j-1}$
    Для $k = 1$ до $j-1$
        Для $i = j$ до n
            $a_{ij} = a_{ij} - l_{ik}l_{jk}$
    $l_{jj} = a_{jj}^{1/2}$

Рис. 2.1.10. Столбцово ориентированная схема разложения Холесского

столбцов L, умноженных на соответствующие числа. Таким образом, основной векторной операцией снова оказывается триада.

На j-м шаге выполняется $j-1$ триад с векторами длины $n-j+1$. Поэтому средняя длина векторов (см. упражнение 2.1.6) равна

$$\frac{n-1+2(n-2)+\ldots+(n-1)\cdot 1}{1+2+\ldots+n-1} = O\left(\frac{n}{3}\right), \tag{2.1.17}$$

что составляет лишь половину средней длины векторов при LU-разложении. Этого следовало ожидать, поскольку в алгоритме Холесского симметрия матрицы коэффициентов используется для сокращения последовательной арифметической работы вдвое. Для векторных машин с большим временем запуска преимущество в скорости алгоритма Холесского по сравнению с LU-разложением гораздо меньше, чем для последовательных компьютеров. Например, из результатов упражнений 2.1.2 и 2.1.8 вытекает, что отношение времени работы обоих алгоритмов составляет 0.8 при $n = 100$ и 0.7 при $n = 500$.

Только для очень больших n это отношение приближается к значению 0.5, соответствующему последовательному компьютеру.

ijk-формы разложения Холесского

Для рис. 2.1.10 был использован только один из возможных способов организации алгоритма Холесского. По аналогии со случаем LU-разложения, существуют и другие способы; все они описываются тройками вложенных циклов

Для _____

Для _____

Для _____

Шесть перестановок индексов i, j и k дают шесть различных вариантов организации алгоритма. Мы сводим некоторые основные аспекты этих вариантов в таблицу 2.1.2, а детальный разбор выносим в приложение 1.

Таблица 2.1.2. Разложение Холесского: *ijk*-формы

Форма	Операция	Модификация	Способ доступа к A	Способ доступа к L
kij	триада	безотлагательная	строка	столбец
kji	триада	безотлагательная	столбец	столбец
ikj	линейная комбинация	отложенная	строка	столбец
jki	линейная комбинация	отложенная	столбец	столбец
ijk	скалярное произведение	отложенная	столбец	строка
jik	скалярное произведение	отложенная	отдельные элементы	строка

Заметим, что некоторые формы, перечисленные в таблице 2.1.2, требуют смешанного доступа к данным, т. е. к строкам матрицы A и столбцам матрицы L или наоборот. Поскольку по предположению матрица L должна замещать A или по крайней мере храниться таким же способом, как A, то смешанный доступ не подходит для векторных компьютеров с векторами, хранящимися только в смежных ячейках памяти.

Алгоритм на рис. 2.1.10 — это форма *jki*; в ней обращения производятся к столбцам A и L. Этой форме в случае LU-раз-

ложения соответствует рис. 2.1.3, однако теперь длины всех векторов, участвующих в модификациях j-го шага, одинаковы. В форме *kji* также нужен доступ к столбцам A и L, поэтому она очень эффективна для векторных компьютеров. На

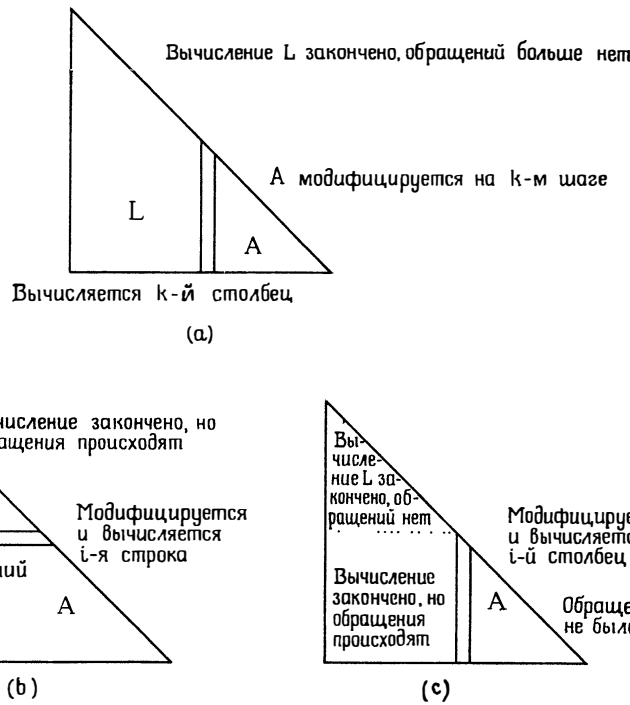


Рис. 2.1.11. Доступ к данным для *ijk*-форм разложения Холесского· а) *kij*, *kji*, b) *ikj*, *ijk*, c) *jki*, *jik*

рис. 2.1.11 показан характер доступа к данным для шести форм алгоритма Холесского из таблицы 2.1.2.

Несколько правых частей

В некоторых задачах требуется решать серию систем уравнений

$$A\mathbf{x}_i = \mathbf{b}_i, \quad i = 1, \ldots, q, \tag{2.1.18}$$

с различными правыми частями $\mathbf{b}_i$, но с одной и той же матрицей коэффициентов A. Так обстоит дело, например, в задачах строительной механики, где заданная конструкция испытывается для нескольких различных конфигураций нагрузок, пред-

ставляемых векторами $\mathbf{b}_i$. То же можно сказать о вычислении обратной матрицы, где $q = n$, а $\mathbf{b}_i$ — вектор с единицей в i-й позиции и нулями в остальных.

Мы можем записать серию систем (2.1.18) эквивалентным образом в виде уравнения $AX = B$, где X и B — матрицы $n \times q$. Столбцово ориентированный алгоритм на рис. 2.1.2 можно модифицировать так, чтобы обрабатывать все столбцы матрицы B, рассматривая их как приписанные к матрице A. На j-м шаге длины векторов остаются равными $n - j$, но число векторных операций теперь возрастает до $n - j + q$. Поэтому для средней длины векторов получаем (см. упражнение 2.1.6)

$$\frac{(n-1+q)(n-1)+\ldots+(1+q)}{(n-1+q)+\ldots+(1+q)} = O\left(\frac{2n^2+3qn}{3n+6q}\right). \quad (2.1.19)$$

Это выражение медленно убывает с ростом q и при $q = n$ составляет $O(5n/9)$; напомним, что при $q = 1$ средняя длина равна $O(2n/3)$.

С другой стороны, мы можем модифицировать и строчно ориентированный алгоритм на рис. 2.1.1, дополняя строки матрицы A строками B. Тогда длины векторов на j-м шаге увеличатся до $n - j + q$. Поскольку число векторных операций попрежнему равно $n - j$, то для средней длины вектора имеем (см. упражнение 2.1.6)

$$\frac{(n-1)(n-1+q)+\ldots+(1+q)}{(n-1)+\ldots+1} = O\left(\frac{2}{3}n+q\right). \quad (2.1.20)$$

Эта функция от q возрастает и при $q = n$ достигает значения $O(5n/3)$, что более чем вдвое превышает значение при $q = 1$. Таким образом, хотя строчно ориентированный алгоритм несколько уступает при $q = 1$ алгоритму столбцовой ориентации, его эффективность с ростом q увеличивается.

Остается этап обратной подстановки, и если правых частей достаточно много, то может оказаться выгодной следующая схема. Запишем системы в виде уравнения $UX = C$, где X и C — матрицы $n \times q$; через $\mathbf{x}_i$ и $\mathbf{c}_i$ обозначим i-е строки этих матриц. В алгоритме

$$\mathbf{x}_i = (\mathbf{c}_i - u_{i,i+1}\mathbf{x}_{i+1} - \ldots - u_{in}\mathbf{x}_n)/u_{ii}, \quad i = n, n-1, \ldots, 1, \quad (2.1.21)$$

строки $\mathbf{x}_n$, $\mathbf{x}_{n-1}$, ... вычисляются посредством триад с векторами длины q. Это в точности алгоритм скалярных произведений (2.1.10), применяемый для одновременного вычисления i-х компонент сразу всех векторов-решений. Итак, основной

векторной операцией является триада, а не скалярное произведение. Будет ли этот подход предпочтительным по сравнению с алгоритмом скалярных произведений, используемым для каждой правой части по отдельности, зависит от величины параметров q и n, равно как и от конкретного компьютера (см. упражнение 2.1.7).

Ортогональное приведение

Альтернативой LU-разложению является представление

$$A = QR, \tag{2.1.22}$$

где Q — ортогональная, а R — верхнетреугольная матрицы. Существуют два распространенных подхода к вычислению разложения (2.1.22): преобразования Хаусхолдера и преобразования Гивенса. Для последовательных компьютеров требуется $O(4n^3/3)$ операций, чтобы найти разложение (2.1.22) с помощью преобразований Хаусхолдера, и $O(2n^3)$ — при использовании преобразований Гивенса. (В число операций включаются как умножения/деления, так и сложения/вычитания. — *Перев.*) Таким образом, эти методы приблизительно вдвое и втрое медленней, чем LU-разложение. Известно, что они численно устойчивы без какого-либо переупорядочения строк, однако это не перевешивает преимущества, которое имеет LU-разложение (даже с выбором главного элемента) в отношении числа операций. Поэтому ортогональные методы редко используются на последовательных компьютерах для решения невырожденных систем уравнений. Однако они широко применяются при вычислении собственных значений, при использовании метода наименьших квадратов, при ортогонализации векторов и для других целей. Во многих из этих приложений A является прямоугольной матрицей, но для простоты мы ограничимся рассмотрением матриц $n \times n$.

Преобразованием Хаусхолдера называется матрица вида $I - \mathbf{w}\mathbf{w}^T$, где $\mathbf{w}$ — вещественный вектор-столбец, для которого $\mathbf{w}^T\mathbf{w} = 2$. Легко показать (упражнение 2.1.11), что матрица Хаусхолдера симметрична и ортогональна. Для получения разложения (2.1.22) мы применим эти матрицы следующим образом. Пусть $\mathbf{a}_1$ — первый столбец матрицы A; положим

$$\mathbf{u}^T = (a_{11} - s,\ a_{21},\ \ldots,\ a_{n1}), \quad \mathbf{w} = \mu\mathbf{u}, \tag{2.1.23}$$

где

$$s = \pm(\mathbf{a}_1^T\mathbf{a}_1)^{1/2}, \quad \gamma = (s^2 - a_{11}s)^{-1}, \quad \mu = \gamma^{1/2} \tag{2.1.24}$$

и знак перед s из соображений численной устойчивости выбран противоположным знаку числа a_{11}. Тогда

$$\mathbf{w}^{\mathrm{T}}\mathbf{w} = \mu^2\left[(a_{11}-s)^2 + \sum_{i=2}^{n} a_{i1}^2\right] = \mu^2\left(\mathbf{a}_1^{\mathrm{T}}\mathbf{a}_1 - 2a_{11}s + s^2\right) = 2,$$

что означает, что $I - \mathbf{w}\mathbf{w}^{\mathrm{T}}$ есть матрица Хаусхолдера. Кроме того,

$$\mathbf{w}^{\mathrm{T}}\mathbf{a}_1 = \mu\left[(a_{11}-s)a_{11} + \sum_{i=2}^{n} a_{i1}^2\right] = \mu\left(s^2 - a_{11}s\right) = \frac{1}{\mu},$$

поэтому

$$a_{11} - w_1\mathbf{w}^{\mathrm{T}}\mathbf{a}_1 = a_{11} - \frac{\mu(a_{11}-s)}{\mu} = s$$

и

$$a_{i1} - w_i\mathbf{w}^{\mathrm{T}}\mathbf{a}_1 = a_{i1} - \frac{a_{i1}\mu}{\mu} = 0, \quad i = 2, \ldots, n.$$

Таким образом,

$$A_1 = P_1A = \begin{bmatrix} s & a'_{12} & \ldots & a'_{1n} \\ 0 & a'_{22} & \ldots & a'_{2n} \\ \cdot & \cdot & \cdot & \cdot \\ 0 & a'_{n2} & \ldots & a'_{nn} \end{bmatrix}, \quad P_1 = I - \mathbf{w}\mathbf{w}^{\mathrm{T}}, \quad (2.1.25)$$

и результатом применения к матрице A преобразования Хаусхолдера является новая матрица с нулями в поддиагональных позициях первого столбца. Мы повторим описанный процесс с вектором $\mathbf{w}_2$, содержащим нуль в первой позиции, а в остальном определяемым аналогично (2.1.23) и (2.1.24), на этот раз с использованием последних $n-1$ элементов второго столбца матрицы A_1. Если $P_2 = I - \mathbf{w}_2\mathbf{w}_2^{\mathrm{T}}$, то матрица P_2A_1 сохранит нулевые элементы в первом столбце и приобретет новые нули в последних $n-2$ позициях второго столбца. Продолжая действовать таким образом, мы последовательно обратим в нуль все поддиагональные элементы с помощью матриц Хаусхолдера $P_i = I - \mathbf{w}_i\mathbf{w}_i^{\mathrm{T}}$. Следовательно, матрица R в равенстве

$$P_{n-1}\ldots P_1A = R$$

является верхнетреугольной. Все матрицы P_i ортогональны, поэтому ортогональными будут матрицы $P = P_{n-1}\ldots P_1$ и P^{-1}. Матрица $Q = P^{-1}$ и есть ортогональный сомножитель разложения (2.1.22).

Основная работа при приведении к треугольному виду посредством преобразования Хаусхолдера связана с модифика-

цией на каждом шаге ненулевых столбцов матрицы A. Приглядимся поэтому внимательней к соотношению (2.1.25). Если $\mathbf{a}_1, \ldots, \mathbf{a}_n$ — столбцы A, то

$$A_1 = (I - \mathbf{w}\mathbf{w}^T)A = A - \mathbf{w}\mathbf{w}^T A = A - \mathbf{w}(\mathbf{w}^T\mathbf{a}_1,\ \mathbf{w}^T\mathbf{a}_2,\ \ldots,\ \mathbf{w}^T\mathbf{a}_n). \tag{2.1.26}$$

Следовательно, i-й столбец матрицы A_1 равен

$$\mathbf{a}_i - \mathbf{w}^T\mathbf{a}_i\mathbf{w} = \mathbf{a}_i - \gamma\mathbf{u}^T\mathbf{a}_i\mathbf{u}. \tag{2.1.27}$$

Теперь мы можем описать алгоритм Хаусхолдера векторным псевдокодом (см. рис. 2.1.12).

Для $k = 1$ до $n - 1$

$$s_k = -\operatorname{sgn}(a_{kk})\left(\sum_{l=k}^{n} a_{lk}^2\right)^{1/2}, \quad \gamma_k = (s_k^2 - s_k a_{kk})^{-1}$$

$$\mathbf{u}_k^T = (0, \ldots, 0, a_{kk} - s_k, a_{k+1,k}, \ldots, a_{nk})$$

$$a_{kk} = s_k$$

Для $j = k + 1$ до n

$$\alpha_j = \gamma_k \mathbf{u}_k^T \mathbf{a}_j$$

$$\mathbf{a}_j = \mathbf{a}_j - \alpha_j \mathbf{u}_k$$

Рис. 2.1.12. Столбцово ориентированная схема метода Хаусхолдера. Алгоритм скалярных произведений

На k-м шаге процесса вычисление s_k требует скалярного произведения длины $n - k + 1$, затем с помощью скалярных операций определяются γ_k и $a_{kk} - s_k$. Внутренний цикл составляет скалярное произведение, сопровождаемое триадой; длины векторов равны $n - k + 1$. Таким образом, средняя длина векторов та же, что и в LU-разложении, т. е. $O(2n/3)$, и во внутреннем цикле выполняется триада того же вида. Главное отличие состоит, разумеется, в том, что теперь во внутреннем цикле требуется еще вычислять скалярные произведения $\mathbf{u}_k^T\mathbf{a}_j$.

Для машин типа «регистр — регистр» важно так организовать вычисления, указанные на рис. 2.1.12, чтобы избежать ненужных обращений к памяти. Рассмотрим первый шаг ($k = 1$) и предположим для начала, что n не превосходит длины векторных регистров. Чтобы начать модификацию, нужно загрузить в векторные регистры $\mathbf{u}_1$ и $\mathbf{a}_2$, после чего вычисляется α_2. Поскольку $\mathbf{u}_1$ и $\mathbf{a}_2$ по-прежнему находятся в регистрах, то можно модифицировать $\mathbf{a}_2$ без дополнительных загрузок. Для $j = 3$

нужна лишь загрузка вектора $\mathbf{a}_3$, так как $\mathbf{u}_1$ остается в регистре, и т. д. Итак, для первого шага ($k = 1$) необходимы только n векторных загрузок.

Если, что вполне вероятно, n больше длины векторного регистра, то ситуация будет совершенно иной. Для вычисления α_2, которое предшествует модификации столбца $\mathbf{a}_2$, требуется частями загрузить полные векторы $\mathbf{u}_1$ и $\mathbf{a}_2$, затем эти же векторы нужно загрузить еще раз, чтобы модифицировать $\mathbf{a}_2$. Если таким же образом поступить со столбцами $\mathbf{a}_3, \ldots, \mathbf{a}_n$, то в общей сложности понадобится $2(n-1)$ загрузок вектора $\mathbf{u}_1$ и по две загрузки для каждого вектора $\mathbf{a}_2, \ldots, \mathbf{a}_n$. Общее число загрузок полных векторов на первом шаге равно $4(n-1)$.

Более удачной стратегией может быть вычисление всех коэффициентов α_i до начала модификаций. В этом случае мы загружаем в регистр первую часть вектора $\mathbf{u}_1$ и вычисляем все частичные скалярные произведения с соответствующими фрагментами векторов $\mathbf{u}_1, \mathbf{a}_2, \mathbf{a}_3, \ldots, \mathbf{a}_n$. Затем загружаются вторые отрезки векторов $\mathbf{u}_1$ и $\mathbf{a}_i$ и вычисление скалярных произведений продолжается, и т. д. При модификациях снова загружается первая часть вектора $\mathbf{u}_1$ и модифицируются соответствующие части в $\mathbf{a}_2, \ldots, \mathbf{a}_n$. Далее загружается вторая часть в $\mathbf{u}_1$, модифицируются соответствующие части в $\mathbf{a}_2, \ldots, \mathbf{a}_n$, и т. д. Преимущество такого подхода заключается в том, что $\mathbf{u}_1$ нужно загружать только дважды, поэтому общее число загрузок полных векторов на первом шаге теперь составляет $2n$. Недостатком является необходимость дополнительной памяти для хранения чисел α_i.

Рассмотрим теперь альтернативную форму метода Хаусхолдера. Пусть $\mathbf{a}_i$ обозначает на этот раз i-ю строку матрицы A; тогда формулу модификации (2.1.26) можно переписать в виде

$$A_1 = A - \mathbf{w}\mathbf{z}^T, \quad \mathbf{z}^T = \mathbf{w}^T A = \mu \sum_{i=1}^{n} u_i \mathbf{a}_i = \mu \mathbf{v}^T, \qquad (2.1.28a)$$

так что новая i-я строка равна

$$\mathbf{a}_i - w_i \mathbf{z}^T = \mathbf{a}_i - \gamma u_i \mathbf{v}^T. \qquad (2.1.28b)$$

Это представление иногда называют формулой *модификации ранга 1*, поскольку ранг матрицы $\mathbf{w}\mathbf{z}^T$ равен 1. Модификацию можно полностью реализовать с помощью триад или линейных комбинаций. Алгоритм в целом приведен на рис. 2.1.13.

Средняя длина векторов в этом алгоритме снова равна $O(2n/3)$, а все векторные операции теперь представляют собой триады. Заметим, однако, что если A хранится по строкам, то

вычисление величины s_k становится менее эффективным, поскольку элементы столбца матрицы A не находятся в смежных позициях памяти. Число обращений к основному массиву по существу такое же, как и в альтернативном алгоритме скалярных произведений. При вычислении $\mathbf{v}_1$ на первом шаге можно использовать все $\mathbf{a}_i$, пока вектор $\mathbf{v}_1$ или его части хранятся в векторном регистре. Таким образом, на первом шаге потребуется $O(2n)$ загрузок полных векторов.

Псевдокод на рис. 2.1.13 основан на неявном предположении, что A хранится по строкам. Может оказаться желательным

Для $k = 1$ до $n - 1$

$$s_k = -\operatorname{sgn}(a_{kk})\left(\sum_{l=k}^{n} a_{lk}^2\right)^{1/2}, \quad \gamma_k = (s_k^2 - s_k a_{kk})^{-1}$$

$$\mathbf{u}_k^{\mathrm{T}} = (0, \ldots, 0, a_{kk} - s_k, a_{k+1,k}, \ldots, a_{nk})$$

$$\mathbf{v}_k^{\mathrm{T}} = \sum_{l=k}^{n} u_{lk}\mathbf{a}_l, \quad \hat{\mathbf{v}}_k^{\mathrm{T}} = \gamma_k \mathbf{v}_k^{\mathrm{T}}$$

Для $j = k$ до n

$$\mathbf{a}_j = \mathbf{a}_j - u_{jk}\hat{\mathbf{v}}_k^{\mathrm{T}}$$

Рис. 2.1.13. Строчно ориентированная схема метода Хаусхолдера. Алгоритм модификаций ранга 1

применять модификации ранга 1 и в том случае, если A хранится по столбцам. Для векторных компьютеров, требующих размещения векторов в смежных ячейках памяти, целесообразно в таких обстоятельствах использовать правосторонние преобразования Хаусхолдера

$$A(I - \mathbf{w}\mathbf{w}^{\mathrm{T}}). \tag{2.1.29}$$

Первое преобразование позволяет получить нули в наддиагональных позициях последнего столбца; последующие преобразования дают нули в столбцах $n - 1$, $n - 2$, и т. д. Таким образом, вместо QR-разложения матрицы A последовательность преобразований $AP_1P_2 \ldots P_{n-1} = L$ приводит к LQ-разложению, где L — нижнетреугольная, а Q — ортогональная матрицы. Это эквивалентно вычислению QR-разложения для A^{T}, но без явного транспонирования матрицы A. Детали LQ-алгоритма, аналогичного алгоритму на рис. 2.1.13, вынесены в упражнение 2.1.10.

Метод Гивенса

Преобразованием Гивенса называется матрица вращения вида

$$P_{ij} = \begin{bmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & \cos\theta_{ij} & & & & \sin\theta_{ij} & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & -\sin\theta_{ij} & & & & \cos\theta_{ij} & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{bmatrix}. \tag{2.1.30}$$

Синусы и косинусы располагаются здесь на пересечении i-й и j-й строк и i-го и j-го столбцов. Легко показать (упражнение 2.1.12), что любая матрица вращения ортогональна. Рассматриваемая как линейное преобразование, матрица P_{ij} задает поворот на угол θ_{ij} в плоскости (i, j).

Чтобы получить QR-разложение (2.1.22), мы воспользуемся матрицами P_{ij} следующим образом. Рассмотрим произведение

$$A_1 = P_{12}A = \begin{bmatrix} c_{12}\mathbf{a}_1 + s_{12}\mathbf{a}_2 \\ -s_{12}\mathbf{a}_1 + c_{12}\mathbf{a}_2 \\ \mathbf{a}_3 \\ \cdots \\ \mathbf{a}_n \end{bmatrix}, \tag{2.1.31}$$

где $c_{12} = \cos\theta_{12}$, $s_{12} = \sin\theta_{12}$, а через $\mathbf{a}_i$ обозначена i-я строка матрицы A. Выберем θ_{12} так, чтобы

$$-a_{11}s_{12} + a_{21}c_{12} = 0. \tag{2.1.32}$$

Таким образом матрица A_1 приобретает нуль в позиции (2,1), и элементы двух ее верхних строк в общем случае отличаются от соответствующих элементов матрицы A, в то время как остальные строки совпадают. Далее мы вычисляем произведение $A_2 = P_{13}A_1$, в котором модифицируются первая и третья строки в A_1, а прочие строки не изменяются; в частности, нуль, полученный на первом шаге в позиции (2,1), сохраняется. Угол θ_{13} теперь подбирается так, чтобы обратить в нуль элемент (3,1) матрицы A_2. Продолжая таким образом, мы один за другим аннулируем оставшиеся (поддиагональные.— *Перев.*) элементы первого столбца, затем аннулируем элементы второго столбца в порядке (3,2), (4,2), ..., $(n, 2)$, и т. д. Всего мы используем $(n-1)+(n-2)+\ldots+1$ матриц вращений. В результате получится верхнетреугольная матрица

$$PA \equiv P_{n-1,\,n} \ldots P_{12}A = R. \qquad (2.1.33)$$

Все матрицы P_{ij} ортогональны, поэтому P и P^{-1} также ортогональны. Полагая $Q = P^{-1}$, видим, что (2.1.33) и есть искомое QR-разложение (2.1.22).

Заметим, что при аннулировании элементов мы не вычисляем сами углы вращений θ_{ij}, а только их синусы и косинусы. Так, для (2.1.32)

$$c_{12} = a_{11}\left(a_{11}^2 + a_{21}^2\right)^{-1/2}, \quad s_{12} = a_{21}\left(a_{11}^2 + a_{21}^2\right)^{-1/2}. \qquad (2.1.34)$$

Метод Гивенса обладает естественным параллелизмом при модификациях строк на каждом шаге. После того как с помощью скалярных операций на первом шаге вычислены c_{12} и s_{12}, две первые строки модифицируются в соответствии с формулами (2.1.31). Это требует умножения вектора на число, а затем выполняется триада. Обратим внимание, что длины векторов равны $n-1$, поскольку, как мы знаем, новый элемент (2,1) имеет нулевое значение. Так как евклидова длина первого столбца при умножении на P_{12} не должна измениться, то мы также знаем, что новым значением элемента (1,1) будет уже вычисленная величина $\left(a_{11}^2 + a_{21}^2\right)^{1/2}$. Итак, аннулирование элементов первого столбца требует $4(n-1)$ векторных операций с векторами длины $n-1$. Аналогично аннулирование элементов j-го столбца потребует $4(n-j)$ векторных операций с векторами длины $n-j$. Поэтому средняя длина векторов для процесса приведения в целом равна (упражнение 2.1.6)

$$\frac{4(n-1)(n-1)+4(n-2)(n-2)+\ldots+4}{4(n-1)+4(n-2)+\ldots+4} = O(2n/3), \qquad (2.1.35)$$

т. е. такая же, как для LU-разложения и метода Хаусхолдера.

Псевдокод рассмотренной схемы алгоритма Гивенса приведен на рис. 2.1.14. Заметим, что при модификации вектора $\mathbf{a}_i$ используется текущее значение вектора $\mathbf{a}_k$, а не только что вычисленное значение, которое обозначено через $\hat{\mathbf{a}}_k$.

В алгоритме Гивенса вдвое больше векторных операций, чем в алгоритмах Хаусхолдера. Однако для машин типа «регистр — регистр» эти операции можно реализовать с исполь-

```
Для k = 1 до n − 1
    Для i = k + 1 до n
        Вычислить s_ki, c_ki
        â_k = c_ki a_k + s_ki a_i
        a_i = − s_ki a_k + c_ki a_i
```

Рис. 2.1.14. Строчно ориентированная схема метода Гивенса

зованием вдвое меньшего числа обращений к памяти по сравнению с алгоритмами Хаусхолдера. Считая теперь первым шагом алгоритма аннулирование элементов первого столбца A, рассмотрим этот шаг более внимательно. Векторы $\mathbf{a}_1$ и $\mathbf{a}_2$ (или их части) загружаются в векторные регистры, а затем выполняется необходимая арифметика. Модифицированный вектор $\mathbf{a}_1$ остается в регистре, поэтому аннулирование элемента (3,1) требует загрузки лишь вектора $\mathbf{a}_3$. Таким образом, для аннулирования $n-1$ элементов первого столбца понадобится лишь по одной загрузке каждой строки матрицы A, и в общей сложности будет n загрузок полных векторов. Это число нужно сопоставить с $O(2n)$ загрузками для алгоритмов Хаусхолдера. Следовательно, для векторных машин алгоритм Гивенса будет более конкурентоспособен, чем показывает простой подсчет количества операций.

Схема алгоритма Гивенса на рис. 2.1.14 ориентирована на обработку по строкам. Столбцово ориентированный алгоритм записан в виде псевдокода на рис. 2.1.15. В начале k-го шага вычисляются все пары синусов и косинусов для k-го столбца. Это требует многократного пересчета элемента (k, k), вследствие чего такой подготовительный цикл l целиком состоит из скалярной арифметики. Первый арифметический оператор в цикле по ji перевычисляет элемент (k, j), который изменяется при каждом новом значении i. Верхний индекс служит для того, чтобы отличать текущее значение элемента (k, j) от его значения для предыдущего i. В терминах векторных операций

внутренний цикл по i можно представить так:

$$\mathbf{p} = \mathbf{s}_k \mathbf{a}_j, \quad \mathbf{r} = \mathbf{c}_k \mathbf{a}_j$$

Для $i = k + 1$ до n

$$a_{kj}^{(i+1)} = c_{ki} a_{kj}^{(i)} + p_i$$

$$q_i = s_{ki} a_{kj}^{(i)}$$

$$\mathbf{a}_j = \mathbf{r} - \mathbf{q}.$$

При формировании $\mathbf{p}$ и $\mathbf{r}$ используется операция покомпонентного умножения векторов. Таким образом, внутренний цикл

Для $k = 1$ до $n - 1$
  Для $l = k + 1$ до n
    Вычислить s_{kl}, c_{kl}
    $a_{kk} = c_{kl} a_{kk} + s_{kl} a_{lk}$
  Для $j = k + 1$ до n
    Для $i = k + 1$ до n
      $a_{kj}^{(i+1)} = c_{ki} a_{kj}^{(i)} + s_{ki} a_{ij}$
      $a_{ij} = -s_{ki} a_{kj}^{(i)} + c_{ki} a_{ij}$

Рис. 2.1.15. Столбцово ориентированная схема метода Гивенса

можно записать с помощью трех векторных операций (два покомпонентных векторных умножения и одно сложение векторов) и скалярного цикла. Ясно, что столбцово ориентированный алгоритм векторизуется не так хорошо, как строчно-ориентированная схема на рис. 2.1.14.

Упражнения к параграфу 2.1

1. Проверить по индукции формулы суммирования

$$\sum_{i=1}^{n} i = \frac{1}{2} n (n + 1), \quad \sum_{i=1}^{n} i^2 = \frac{1}{6} n (n + 1)(2n + 1).$$

С их помощью показать, что отношение в (2.1.3) равно

$$\frac{1}{3} \frac{n (n - 1)(2n - 1)}{n (n - 1)} = \frac{1}{3} (2n - 1) = O(2n/3).$$

2. Пусть для векторного компьютера затраты времени составляют $(100 + 10m)$ нс при сложении, $(1600 + 10m)$ нс для триады, $(1600 + 70m)$ нс

при делении векторов длины m и 100 нс для скалярных операций. Показать, что для системы порядка n общее время схем LU-разложения, изображенных на рис. 2.1.1 и 2.1.2, составляет соответственно

$$T_R = \left(\frac{10}{3}n^3 + 850n^2\right) \text{ нс} \quad \text{и} \quad T_C = \left(\frac{10}{3}n^3 + 835n^2\right) \text{ нс}.$$

3. Пусть векторный компьютер имеет те же характеристики, что и в упражнении 2, и вдобавок может вычислять скалярные произведения с затратами времени $(2000 + 20m)$ нс, где m — длина векторов. Выяснить, в зависимости от порядка n системы, какой из двух алгоритмов решения треугольных систем эффективней: алгоритм (2.1.10) или алгоритм на рис. 2.1.5.

4. Предположим, что в машине типа «регистр-регистр» объем регистра равен 64 словам. Обсудить модификации алгоритмов (2.1.4) и (2.1.5) с учетом этого обстоятельства.

5. Сформулировать алгоритм скалярных произведений и столбцовый алгоритм для решения нижнетреугольной системы уравнений $L\mathbf{x} = \mathbf{b}$.

6. Использовать формулы суммирования из упражнения 1 для проверки равенств (2.1.17), (2.1.19), (2.1.20) и (2.1.35).

7. Пусть векторный компьютер имеет характеристики, указанные в упражнениях 2 и 3. Для системы с матрицей $n \times n$ и q правыми частями выяснить, при каких значениях q и n алгоритм обратной подстановки (2.1.21) более эффективен, чем обратная подстановка (2.1.10), применяемая последовательно к каждой правой части.

8. Пусть векторный компьютер имеет те же характеристики, что и в упражнении 2, и пусть для вычисления квадратного корня требуется 500 нс. Показать, что общее время выполнения алгоритма Холесского (рис. 2.1.10) составляет

$$T = \left(\frac{5}{3}n^3 + 835n^2\right) \text{ нс}.$$

9. Изменить равенство (2.1.13) так, чтобы оно соответствовало разложению $A = LL^t$, а затем дать формулировки типа формулировки Донгарры—Айзенштата для разложения Холесского.

10. Построить алгоритм модификаций ранга 1, аналогичный алгоритму на рис. 2.1.13, для LQ-разложения матрицы A. Использовать при этом правосторонние преобразования Хаусхолдера вида (2.1.29).

11. Пусть $P = I - \mathbf{w}\mathbf{w}^t$, где $\mathbf{w}$ — вещественный вектор-столбец. Показать, что: 1) P — симметричная матрица; 2) матрица P ортогональна тогда и только тогда, когда $\mathbf{w}^t\mathbf{w} = 2$.

12. Показать, что матрица вращения (2.1.30) ортогональна.

13. Распространить алгоритм Донгарры — Айзенштата на блочный случай, заменив (2.1.13) равенством

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & U_{33} \end{bmatrix}$$

Теперь A_{22}, L_{22} и U_{22} — матрицы, причем в L_{22} все диагональные элементы равны 1. Приравнивая соответствующие элементы этой блочной факторизации, получить формулы типа (2.1.14), определяющие L_{22}, L_{32}, U_{22} и U_{23} при заданных L_{11}, L_{21}, L_{31}, U_{11}, U_{12}, U_{13}.

Литература и дополнения к параграфу 2.1

1. В статье [Dongarra, Gustavson, Karp, 1984] довольно подробно обсуждаются способы обработки векторов, длина которых превышает длину векторных регистров, а также важные вопросы зацепления и совмещения. В этой статье, кроме того, систематическим образом вводятся *ijk*-формы *LU*-разложения. Уже в работе [Moler, 1972] пропагандировалось применение формы *kji* для последовательных компьютеров с виртуальной памятью, а в отчете [Fong, Jordan, 1977] отмечалось, что форма *jki* очень удобна для машины CRAY-1. Более детально *ijk*-формы для *LU*-разложения и разложения Холесского рассматриваются в статье [Ortega, 1987b]; там же дана более общая формулировка, включающая в себя алгоритмы окаймления и Доннгарры — Айзенштата. Приложение 1 отчасти базируется на этой статье.

2. Алгоритм Доннгарры — Айзенштата описывается в статье [Dongarra, Eisenstat, 1984], где, кроме того, обсуждаются алгоритмы окаймления. Схема скалярных произведений алгоритма Доннгарры — Айзенштата по существу совпадает с *методом Дулитла* (см. его описание, например, в книге [Stewart, 1973]). Мы предпочитаем называть этот метод алгоритмом Доннгарры - Айзенштата, учитывая акцент, который эти авторы сделали на операциях векторно-матричного умножения, входящих в алгоритм.

3. В статье [Calahan, 1986] алгоритм Доннгарры — Айзенштата обобщается на блочный случай (см. упражнение 2.1.13), основной операцией теперь является матрично-матричное, а не матрично-векторное умножение. Приводятся результаты, полученные автором для одного процессора машины CRAY-2. В техническом отчете [Gallivan et al., 1987] обсуждаются другие блочные алгоритмы *LU*-разложения, предназначенные главным образом для системы Alliant FX/8. Результаты этих авторов внечатляют: создается ощущение, что для машин с иерархическим устройством памяти к наилучшим алгоритмам приводит, по всей вероятности, блочный подход.

4. В статье [Dongarra, Hewitt, 1986] обсуждается комбинация форм *jki* и *kji* *LU*-разложения, в которой вычисляются сразу несколько столбцов (скажем, четыре) матрицы *L*, после чего они используются для модификации остальных столбцов. Сообщается о великолепных результатах, полученных для четырехпроцессорной машины CRAY X-MP. Данные о времени работы похожего алгоритма для компьютера CRAY-2 приведены в [Dave, Duff, 1987]. Среди других статей (с ориентацией на машину CRAY X-MP) можно назвать работы [Calahan, 1985] и [Dongarra, Hinds, 1985], в последней приведены данные о времени *LU*-разложения и разложения Холесского не только для CRAY X-MP, но и для двух японских машин.

5. Существует множество различных точек зрения на проблему перестановок в гауссовом исключении. В диссертации [Lambiotte, 1975] для машин CDC рекомендуется выполнять перестановки с помощью скалярных операций, если матрица *A* хранится по столбцам. Для тех же машин автор работы [Kascic, 1979], напротив, советует использовать при перестановках операции сборки и рассылки, если *A* хранится по столбцам, и операцию сборки, а затем векторную операцию максимума при поиске главного элемента, если *A* хранится по строкам. В статье [Dongarra, Gustavson, Karp, 1984] авторы рекомендуют явные перестановки строк для машин CRAY. В техническом отчете [Hay, Gladwell, 1985] отмечается, что неявное выполнение перестановок приводит к косвенной адресации, а это плохо для векторных машин.

6. Вариантом гауссова исключения является алгоритм Гаусса — Жордана, где матрица *A* приводится к диагональному, а не к верхнетреугольному виду. На *k*-м шаге текущая матрица имеет форму, показанную на рис. 2.1.16. То обстоятельство, что элементы могли изменить свои первона-

чальные значения, указано с помощью символа ^. Считая, что $\hat{a}_{kk} \neq 0$, вычитаем надлежащие кратные k-й строки из остальных строк с тем, чтобы исключить все элементы k-го столбца, кроме диагонального При модификациях основной операцией будет триада $\mathbf{a}_j - a_{kj}\mathbf{m}_k$, где $\mathbf{a}_j$ — столбец текущей матрицы A, а $\mathbf{m}_k$ — вектор, составленный из множителей a_{ik}/a_{kk} и имеющий нулевую k-ю компоненту Потенциальное преимущество алгоритма Гаусса — Жордана в случае использования векторных машин заключается в том, что в ходе приведения длины векторов не уменьшаются, а остаются равными n. Однако число последовательных операций в алгоритме составляет $O(n^3)$ по сравнению с $O(2n^3/3)$ операциями в гауссовом исключении, и эти добавочные

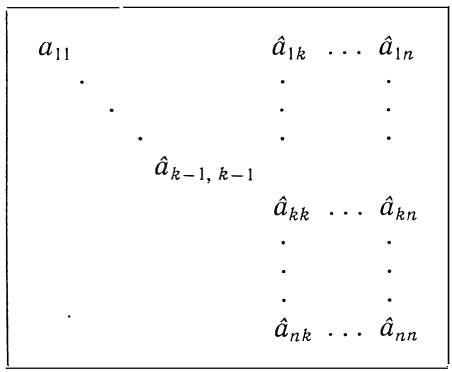


Рис. 2.1.16.

операции, скорей всего, сведут на нет выигрыш, полученный от большей длины векторов Кроме того, могут возникнуть проблемы с численной устойчивостью. В целом, несмотря на кажущуюся привлекательность работы с более длинными векторами, по-видимому, мало оснований для того, чтобы предпочесть этот алгоритм гауссову исключению.

7. В статье [Dongarra, Kaufman, Hammarling, 1986] приводятся данные о времени решения на машине CRAY-1 треугольной системы с q правыми частями посредством столбцового алгоритма (CS), столбцового алгоритма (UCS) с циклами, развернутыми на глубину 2, и алгоритма скалярных произведений (2.1 21) (IP). В приводимой ниже таблице указаны некоторые из этих результатов; единицей времени является 1 млс. Обратите внимание, что столбцовый алгоритм с развертыванием циклов во всех случаях примерно на 50 % быстрее первоначальной версии

n	q	CS	UCS	IP
100	25	12.5	8.32	6.92
	100	49.9	33.3	14.4
200	25	38.6	25.5	24.1
	200	308	202	93.7
300	25	78.5	51.1	51.8
	300	940	613	290

8. Прекрасное изложение способов реализации метода Холесского для последовательных компьютеров дано в книге [George, Liu, 1981] Наши рисунки, показывающие характер доступа к данным, выполнены по образцу рисунков из этой книги Обзор реализаций метода Холесского для векторных компьютеров первого поколения, таких, как TI-ASC, CRAY-1 и CDC STAR-100, можно найти в статье [Voigt, 1977]

9. Алгоритмы данного параграфа не используют явно наличие нулей в матрице коэффициентов A; этот вопрос будет рассмотрен в § 2.3 Здесь мы отметим только, что нули в A вполне могут превратиться в ненулевые элементы при LU-разложении или каком-либо ином алгоритме; это явление называется *заполнением.* Один из простейших примеров того, как может произойти очень сильное заполнение, дает матрица вида

$$\begin{bmatrix} * & * & . & . & . & * \\ * & . & & & & \\ . & & . & & & \\ . & & & . & & \\ . & & & & & \\ * & & & & & * \end{bmatrix}$$

В ней все элементы равны нулю, кроме элементов первой строки и первого столбца, а также диагональных элементов. В ходе LU-разложения все нулевые позиции нижнего треугольника, вообще говоря, будут заполнены. С другой стороны, в матрице

$$\begin{bmatrix} * & & & * \\ & . & & . \\ & & . & . \\ & & & * \\ * & \dots & * & * \end{bmatrix}$$

заполнения не происходит, а множители L и U выглядят так же, как и соответствующие части исходной матрицы.

10 В статье [Mattingly, Meyer, Ortega, 1987] обсуждаются (по аналогии с LU-разложением и разложением Холесского) ijk-формы методов Хаусхолдера и Гивенса. Строчно ориентированная схема алгоритма Гивенса, приведенная в основном тексте параграфа, соответствует форме kij. Имеются также естественные формы kji, jki и ikj, однако две другие формы не приводят к эффективным векторным алгоритмам Для метода Хаусхолдера алгоритм модификаций ранга 1 (рис. 2.1.13) соответствует форме kij, а алгоритм скалярных произведений (рис. 2 1.12) — форме kji Форма jki описывает алгоритм отложенных модификаций, но остальные три ijk-формы бесполезны.

11. Чтобы уменьшить ошибки округлений, вычисление синусов и косинусов в методе Гивенса можно проводить не по формулам (2 1.34), а с помощью следующего алгоритма:

$$\text{если } |a_{21}| \geqslant |a_{11}|, \quad \text{то} \quad r = a_{11}/a_{21}, \quad s_{12} = (1 + r^2)^{-1/2}, \quad c_{12} = s_{12} r;$$

$$\text{если } |a_{21}| < |a_{11}|, \quad \text{то} \quad r = a_{21}/a_{11}, \quad c_{12} = (1 + r^2)^{-1/2}, \quad s_{12} = c_{12} r.$$

В методе Хаусхолдера также возможно переполнение или получение машинного нуля при вычислении величины s_k (см. рис. 2.1.12). Эту проблему можно устранить путем масштабирования; см по этому поводу, например, [Stewart, 1973]

12 В работе [Dongarra, Kaufman, Hammarling, 1986] было предложено объединить в одном шаге два преобразования Хаусхолдера, чтобы сократить число обращений к памяти. Обобщение этой идеи привело к построению в статье [Bischof, van Loan, 1987] блочных методов Хаусхолдера, для которых основной операцией является матрично-матричное умножение Как и в случае LU-разложения, подобные блочные методы, вероятно, будут наилучшими для машин с иерархической организацией памяти.

2.2. Прямые методы для параллельных компьютеров

Теперь мы рассмотрим вопрос о реализации алгоритмов из § 2.1, а также некоторых других алгоритмов для параллельных компьютеров. Как и в предыдущем параграфе, будем считать матрицу коэффициентов A размера $n \times n$ заполненной.

LU-разложение

Предположим вначале, что мы располагаем системой с локальной памятью и числом процессоров $p = n$. Тогда один из возможных вариантов организации LU-разложения выглядит

Шаг 1: $\mathbf{a}_1 \to P_i$, $i = 2, \ldots, n$
В P_i $(i = 2, \ldots, n)$ вычисляются l_{i1} и новые элементы a_{ij} $(j = 2, \ldots, n)$

Шаг 2: $\mathbf{a}_2 \to P_i$, $i = 3, \ldots, n$
В P_i $(i = 3, \ldots, n)$ вычисляются l_{i2} и новые элементы a_{ij} $(j = 3, \ldots, n)$

Рис. 2.2.1. LU-разложение в системе из n процессоров

так. Пусть i-я строка матрицы A хранится в процессоре i. На первом шаге первая строка рассылается всем процессорам, после чего вычисления

$$l_{i1} = a_{i1}/a_{11}, \quad a_{ij} = a_{ij} - l_{i1}a_{1j}, \quad j = 2, \ldots, n, \qquad (2.2.1)$$

могут выполняться параллельно процессорами $P_2, \ldots, P_n$. На втором шаге вторая строка приведенной матрицы рассылается из процессора P_2 процессорам $P_3, \ldots, P_n$, а затем проводятся

параллельные вычисления, и т. д. Первые два шага показаны на рис. 2.2.1; через $\mathbf{a}_i$ обозначена i-я строка текущей матрицы A. Отметим два главных недостатка этого подхода: значительный объем обмена данными между каждыми двумя шагами и уменьшение числа активных процессоров на 1 на каждом шаге.

Альтернативой хранению по строкам является вариант, в котором i-й столбец матрицы A хранится в процессоре i. В этом случае на первом шаге все множители l_{i1} вычисляются в процессоре 1 и рассылаются остальным процессорам. Затем процессорами 2, ..., n параллельно производятся модификации

$$a_{ij} = a_{ij} - l_{i1}a_{1j}, \quad j = 2, \ldots, n. \tag{2.2.2}$$

Вычислив множители l_{i1}, процессор 1 прекращает работу; вообще, с каждым шагом число простаивающих процессоров увеличивается на единицу. Возникает та же, что и в строчно ориентированном алгоритме, проблема балансировки нагрузки.

Слоистая схема хранения

В более реалистической ситуации, когда $p \leqslant n$, проблема балансировки нагрузки в известной степени смягчается. Предположим, что $n = kp$ и применяется хранение по строкам. Поместим первые k строк матрицы A в память процессора 1, следующие k строк в память процессора 2, и т. д. Этот способ хранения назовем *блочной схемой*. Снова первая строка рассылается из процессора 1 остальным процессорам, а затем выполняются вычисления (2.2.1), однако теперь это делается блоками по k наборов операций в каждом процессоре. Как и прежде, в ходе приведения все большее число процессоров становятся бездействующими, однако отношение общего времени вычислений к времени обменов и времени простоев является возрастающей функцией от k.

Более привлекательна схема хранения, в которой строки, распределенные в разные процессоры, как бы прослаивают друг друга. Будем по-прежнему считать, что $n = kp$, и пусть строки $1, p+1, 2p+1, \ldots$ хранятся в процессоре 1, строки $2, p+2, 2p+2, \ldots$ — в процессоре 2, и т. д. (см. рис. 2.2.2). Такой способ хранения мы будем называть *циклической слоистой схемой*. Проблема простоя процессоров для этой схемы теряет остроту; например, процессор 1 будет работать почти до самого конца приведения, а именно до тех пор, пока не закончится обработка строки $(k-1)p+1$. Разумеется, некоторая неравномерность загрузки процессоров сохранится. Так, после первого шага строка 1 станет не нужна, и на следующем

шаге процессору 1 придется обрабатывать на одну строку меньше, чем остальным процессорам. Неравномерность сходного типа возникает и в том вполне вероятном случае, когда n не кратно числу процессоров.

Тот же принцип прослаивания можно использовать при хранении по столбцам: теперь столбцы $1, p+1, \ldots, (k-1)p+1$ закреплены за процессором 1, столбцы $2, p+2, \ldots, (k-1)p+2$ — за процессором 2, и т. д. Снова все процессоры (при небольшом дисбалансе нагрузки) будут работать почти до конца приведения. Сравнение двух схем хранения проводится в упражнениях 2.2.1 и 2.2.3.

строки	строки		строки
$1, p+1, \ldots, (k-1)p+1$	$2, p+2, \ldots, (k-1)p+2$	...	$p, 2p, \ldots, kp$
процессор 1	процессор 2	...	процессор p

Рис. 2.2.2. Циклическая слоистая строчная схема хранения

Вариантом строчной (или столбцовой) циклической схемы является *слоистая схема с отражениями*. Здесь строки распределяются так, как показано на рис. 2.2.3 для случая четырех процессоров и шестнадцати строк. В общем случае первые p строк распределяются между p процессорами в естественном порядке, следующие p строк распределяются в обращенном порядке, и т. д. Циклическая схема и схема с отражениями имеют ряд общих свойств, но у циклической схемы все же есть некоторые преимущества. В дальнейшем мы будем рассматривать только ее, и термины «циклическая», «слоистая» и «циклическая слоистая» будут использоваться как синонимы.

Принцип слоистого хранения оправдан главным образом для систем с локальной памятью. Но его можно применять и в системах с глобальной памятью как средство распределения заданий между процессорами. Это значит (если принять для определенности строчную слоистую схему), что процессор 1 будет выполнять модификации (2.2.2) для строк $p+1, 2p+1$, и т. д. Однако в системах с глобальной памятью динамическая балансировка нагрузки может осуществляться и с помощью банка заданий. Для систем с локальной памятью этот способ динамической балансировки не так хорош, поскольку требует многократного перераспределения данных между процессорами.

Опережающая рассылка и опережающее вычисление

Обсудим теперь более подробно LU-разложение с использованием циклической слоистой схемы хранения. Начнем с систем с локальной памятью. На первом шаге аннулируются элементы первого столбца. Если процесс ведется обычным последовательным образом, то вычисляются множители l_{i1}, модифицируются соответствующие строки, а затем начинается второй шаг. В начале этого шага процессор, хранящий вторую строку, должен переслать ее остальным процессорам. Следовательно, возникает задержка на время этой пересылки. Очевидным выходом из положения является немедленная рассылка процессором 2 модифицированной второй строки, как только эта строка приняла окончательный вид. Такую стратегию мы назовем *опережающей рассылкой*. Если можно совместить вычисления и обмены, то все процессоры будут заняты модификациями первого шага, пока идет рассылка. То же самое будет происходить и на других шагах: на k-м шаге $(k+1)$-я строка рассылается сразу после того, как окончится ее модификация. Насколько выгодна эта стратегия для конкретной машины, зависит от топологии ее межпроцессорных связей.

Стратегию опережающей рассылки можно использовать и в системах с разделяемой памятью. Здесь при прямолинейной реализации потребовалась бы синхронизация после каждого шага с тем, чтобы ни один процессор не начал следующий шаг, пока не все процессоры закончили текущий. На k-м шаге процессор, которому приписана $(k+1)$-я строка, имеет не меньше работы, чем любой другой процессор, поскольку обрабатывает по крайней мере столько же строк. Поэтому другим процессорам, возможно, придется ждать, пока не завершит работу данный процессор, прежде чем можно будет начать следующий шаг. Стратегия опережающей рассылки, которую теперь следует называть стратегией *опережающего вычисления*, меняет порядок действий, маркируя $(k+1)$-ю строку, как только ее модификация закончена, признаком «готова». Теперь другие процессоры, завершив свою работу на k-м шаге, могут немедленно приступать к $(k+1)$-му шагу, если $(k+1)$-я строка отмечена признаком «готова». Маркировка помогает осуществлять необходимую синхронизацию без неоправданных задержек. Заметим, что опережающую рассылку и опережающее вычисление называют еще *конвейеризацией*.

Частичный выбор главного элемента

Использование схемы частичного выбора для обеспечения численной устойчивости процесса создает дополнительный

аспект проблемы хранения информации. Предположим вначале, что хранение A организовано по столбцовой циклической схеме. Тогда поиск главного элемента на каждом шаге происходит в каком-то одном процессоре. Схема хороша своей простотой, но чревата опасностью простоя остальных процессоров во время поиска. Эту опасность в принципе можно уменьшить, применяя стратегию опережающего вычисления и рассылки: на k-м шаге поиск главного элемента в $(k+1)$-столбце начинается сразу по окончании модификации этого столбца. В любом случае после того, как определена ведущая строка, эту информацию следует передать другим процессорам. Вслед за тем все процессоры, работая параллельно, могут выполнить перестановку строк (перестановка может быть выполнена и неявно, посредством индексирования).

Для строчной циклической схемы хранения ситуация совершенно иная. Теперь в поиск максимального элемента в текущем (скажем, k-м) столбце вовлечены все процессоры. Для организации поиска можно применить механизм сдваивания, описанный в § 1.2 (см. упражнение 2.2.3). По окончании поиска номер следующей ведущей строки будет известен только одному процессору, и эту информацию нужно будет разослать остальным процессорам. Снова мы должны принять решение, выполнять ли перестановку строк физически. Если принято положительное решение, то в перестановке участвуют лишь два процессора, а остальные в это время простаивают. С другой стороны, если перестановка производится неявно, то в дальнейших вычислениях мы отходим от строчной циклической схемы. В самом деле, с ростом числа перестановок, не выполняемых физически, схема хранения матрицы все больше напоминает случайное распределение строк между процессорами.

ijk-формы и параллельно-векторные компьютеры

Обсуждавшиеся в предыдущем параграфе (и более детально рассматриваемые в приложении 1) *ijk*-формы LU-разложения указывают ряд других вариантов организации его вычисления, которые могут быть полезны для параллельных систем. Еще важнее, что они дают альтернативные варианты организации для параллельно-векторных систем (где в качестве процессоров выступают векторные компьютеры). Подробный анализ *ijk*-форм для параллельных и параллельно-векторных систем проводится в приложении 1; в таблице 2.2.1 мы суммируем основные выводы этого приложения.

В таблице 2.2.1 приняты следующие обозначения: *kij*, *r* — это форма *kij* с использованием циклической слоистой строчной

Таблица 2.2.1. *LU*-разложение в случае параллельно-векторных систем: *ijk*-формы

kij, *r*:	Минимальные задержки	Полные векторные длины
kij, *c*:	Задержки на вычисление множителей и рассылку	Частичные векторные длины
kji, *r*:	Минимальные задержки	Частичные векторные длины
kji, *c*.	Минимальные задержки	Полные векторные длины при наличии задержек
ikj, *r*:	Сильный дисбаланс нагрузки	
ikj, *c*:	Задержки на вычисление множителей и рассылку	Частичные векторные длины
jki, *r*:	Большие расходы на обмены	Частичные векторные длины
jki, *c*:	Задержки на вычисление множителей и рассылку	Полные векторные длины при наличии задержек
ijk, *r*:	Большие расходы на обмены	
ijk, *c*:	То же	
jik, *r*.	»	
jik, *c*:	»	

схемы хранения; *kij*, *c* — форма *kij* с циклической слоистой столбцовой схемой; аналогично для других форм. Для каждого метода указываются его главная особенность или основные недостатки. Запись «Полные векторные длины» означает, что можно работать с векторами максимальной длины, допустимой на данном шаге *LU*-разложения; например, для *k*-го шага максимальная длина векторов равна $O(n-k)$. Запись «Частичные векторные длины» указывает на то, что соответствующие данные распределяются между процессорами и, следовательно, длины векторов делятся на число процессоров. Как видно из таблицы, для схем *kji*, *c* и *jki*, *c* можно добиться полных векторных длин, но ценой задержек при получении элементов вектора. Формы *ijk* и *jik* характеризуются очень плохим распределением данных и, по-видимому, не пригодны для любой параллельной системы. Наиболее многообещающими кажутся формы *kij*, *r*, *kji*, *r* и *kji*, *c*. В частности, форма *kij*, *r* соответствует алгоритму, описанному выше под названием «опережающее вычисление и рассылка».

Мелкозернистые алгоритмы и организация потока данных

До сих пор мы рассматривали параллельные реализации *LU*-разложения, которые используют в качестве основной единицы строки (или столбцы); назовем их *среднезернистыми* ал-

горитмами. По существу они повторяют векторные алгоритмы предыдущего параграфа. Однако в разложении заключено больше внутреннего параллелизма, и при наличии достаточного числа процессоров этим обстоятельством можно воспользоваться. Обсудим некоторые алгоритмы такого рода; будем называть их *мелкозернистыми*.

строки	строки	строки	строки
1, 8, 9, 16	2, 7, 10, 15	3, 6, 11, 14	4, 5, 12, 13
процессор 1	процессор 2	процессор 3	процессор 4

Рис. 2.2.3. Слоистая строчная схема с отражениями

Заметим прежде всего, что LU-разложение в принципе можно организовать так, как показано на рис. 2.2.4; на каждом шаге соответствующие вычисления проделываются параллельно.

Шаг 1. Вычислить первый столбец множителей l_{i1}, $i = 2, \ldots, n$.

Шаг 2. Вычислить модифицированные элементы $a^1_{ij} = a_{ij} - l_{i1}a_{1j}$, $i = 2, \ldots, n$, $j = 2, \ldots, n$.

Шаг 3. Вычислить второй столбец множителей l_{i2}, $i = 3, \ldots, n$.

Шаг 4. Вычислить модифицированные элементы $a^2_{ij} = a^1_{ij} - l_{i2}a^1_{2j}$, $i = 3, \ldots, n$, $j = 3, \ldots, n$.

. . . . . .

Шаг $2n - 3$. Вычислить последний множитель $l_{n,n-1}$.

Шаг $2n - 2$. Модифицировать элемент (n, n).

Рис. 2.2.4. LU-разложение с максимальным параллелизмом

Поскольку для модификации любого элемента требуются две операции (умножение и сложение), то процесс завершится за $3(n-1)$ временны́х шагов. (Мы исходим из обычного нереалистического предположения, что деление можно выполнить за такое же время, как умножение или сложение.)

В этой схеме предполагается, что в системе имеется по крайней мере $(n-1)^2$ процессоров, т. е. можно провести параллельно все вычисления шага 2. Однако обойден вниманием важный вопрос об обменах. Считается, что на каждом шаге

любому процессору доступны необходимые данные (множители и/или пересчитанные матричные элементы). Чтобы это было справедливо для системы с локальной памятью, необходимо переслать данные из процессоров, которые их вычислили, в те процессоры, которым они нужны на следующем шаге. Сейчас мы обсудим один из способов организации такой пересылки.

Предположим, что имеющиеся $(n-1)^2$ процессоров пронумерованы таким же образом, как матричные элементы, которые они пересчитывают, т. е. процессор P_{ij} пересчитывает элемент a_{ij}. Тогда в схему на рис. 2.2.4 нужно внести следующие дополнения, учитывающие пересылки:

Шаг 1. Вычислить l_{i1} в процессоре P_{i1}, $i=2, \ldots, n$.

Шаг 1а. Переслать l_{i1} в процессоры P_{ij}, $i=2, \ldots, n$, $j=2, \ldots, n$.

Шаг 2. Вычислить a_{ij}^1 в процессоре P_{ij}, $i=2, \ldots, n$, $j=2, \ldots, n$.

Шаг 2а. Переслать a_{2j}^1 в P_{ij}, $i=3, \ldots, n$, $j=2, \ldots, n$.

. . . . . .

Если считать, что на каждом шаге все необходимые пересылки могут быть сделаны за единицу времени, то общее количество временны́х шагов увеличивается до $O(5n)$. Однако возможность столь быстрой пересылки совершенно нереалистична.

Рассмотрим теперь другой способ организации, так называемый *алгоритм потока данных*. Пусть n^2 процессоров расположены в виде квадратной решетки и каждый соединен с четырьмя соседями (см. рис. 1.1.14). Будем считать, во-первых, что любой процессор может переслать одно машинное слово за такое же время, какого требует одна операция над числами с плавающей точкой, и, во-вторых, что пересылка и вычисления могут происходить одновременно. (Это очень оптимистические предположения.) Начальные шаги алгоритма, к обсуждению которого мы приступаем, расписаны на рис. 2.2.5.

Исследуем этот процесс более подробно. На рис. 2.2.6 показана ситуация в конце шага 4. Только что в процессоре P_{22} было вычислено произведение $l_{21}a_{12}$. В процессоры P_{23} и P_{32} были переданы соответствующие множители, но вычисления, использующие эти множители, еще не начинались. В процессоре P_{41} вычислен множитель l_{41}, и первая строка переслана всем процессорам пятой строки. Чтобы читатель мог лучше представить себе развитие процесса, на рис. 2.2.7 указаны вычисления и пересылки, происходящие *в течение* шага 5.

Шаг 1. Переслать a_{1j} из P_{1j} в P_{2j}, $j = 1, \ldots, n$.
Шаг 2. Переслать a_{1j} из P_{2j} в P_{3j}, $j = 1, \ldots, n$.
Вычислить l_{21}.
Шаг 3. Переслать a_{1j} из P_{3j} в P_{4j}, $j = 1, \ldots, n$.
Переслать l_{21} в P_{22}. Вычислить l_{31}.
Шаг 4. Переслать a_{1j} из P_{4j} в P_{5j}, $j = 1, \ldots, n$.
Переслать l_{21} в P_{23}. Переслать l_{31} в P_{32}.
Вычислить l_{41}. Вычислить $l_{21}a_{12}$ в P_{22}.
Шаг 5. Переслать a_{1j} из P_{5j} в P_{6j}, $j = 1, \ldots, n$.
Переслать l_{21} в P_{24}. Переслать l_{31} в P_{33}.
Переслать l_{41} в P_{42}. Вычислить l_{51}.
Вычислить $a^1_{22} = a_{22} - l_{21}a_{12}$ в P_{22}.
Вычислить $l_{21}a_{13}$ в P_{23}.
Вычислить $l_{31}a_{12}$ в P_{32}.

Рис. 2.2.5. LU-разложение, организованное по принципу потока данных

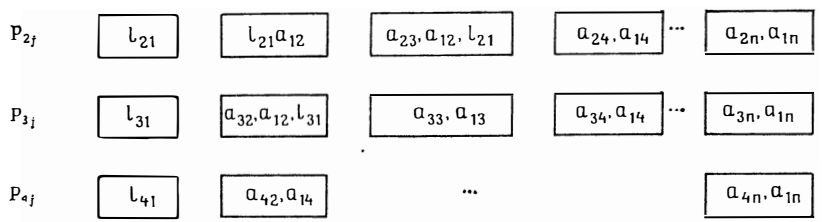


Рис. 2.2.6. Конец шага 4 алгоритма потока данных

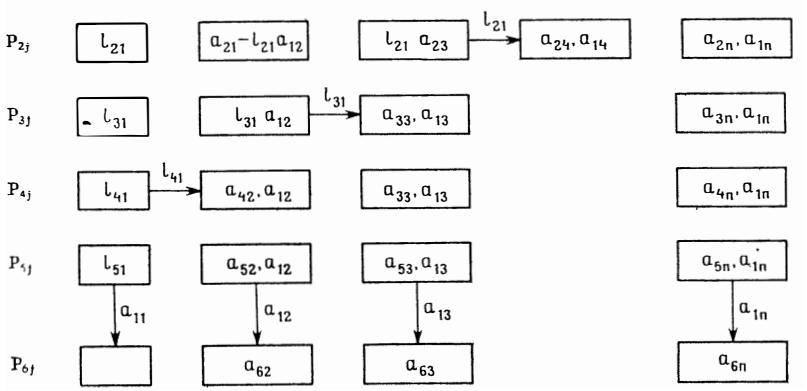


Рис. 2.2.7. Шаг 5 алгоритма потока данных

Суть процесса должна быть теперь ясна. На каждом шаге вычисляется очередной множитель, а все множители, вычисленные ранее, передаются направо в соседние процессоры. Одновременно процессоры, имеющие необходимые данные, пересчитывают матричные элементы. Вычислительный процесс, таким образом, как бы движется сквозь матрицу (или, что эквивалентно, через массив процессоров) на манер диагонального волнового фронта. Поэтому такой способ организации иногда называют LU-разложением *в форме волнового фронта*. Заметим, что как только модифицированное значение элемента (2, 2) определено, его можно переслать процессору P_{32}, и сразу после пересчета элемента (3, 2) можно вычислить множитель l_{32}. Это порождает новый волновой фронт, в котором производятся модификации второго шага LU-разложения. Через некоторое время сквозь матрицу будут двигаться сразу несколько волновых фронтов.

Выясним теперь, какое количество временны́х шагов необходимо для завершения факторизации. Рассмотрим вторую строку. В конце пятого временно́го шага вычислен модифицированный элемент a_{22}^{1}, множитель l_{21} находится в P_{24}, а в P_{23} вычислено произведение $l_{21}a_{13}$. На каждом из последующих временны́х шагов завершается модификация очередного элемента, поэтому обработка второй строки будет закончена после $5+n-2=n+3$ временны́х шагов. В течение первого этапа пересчет третьей строки на один временно́й шаг отстает от пересчета второй, а второй этап разложения начинается через четыре временны́х шага после первого. Чтобы убедиться в этом, проследим за элементом (3, 3). В тот момент, когда вычислен элемент (2, 3), одновременно вычислен и элемент (3, 2), а в процессоре P_{32} получен элемент (2, 2). Для того чтобы найти модифицированное значение элемента (3, 3), требуются следующие действия: вычислить l_{32}; переслать l_{32}; вычислить $l_{32}a_{23}^{1}$; вычислить $a_{33}^{1}=a_{33}-l_{32}a_{23}^{1}$. Следовательно, обработка третьей строки завершится после $n+7$ временны́х шагов. Продолжая действовать таким образом, мы увидим, что весь процесс факторизации потребует $n+3+4(n-2)=5n-5$ временны́х шагов.

Из проведенного подсчета числа временны́х шагов вытекает, что в LU-разложении, организованном по принципу потока данных, достигается ускорение

$$S_p=\frac{O\left(\frac{2}{3}n^3\right)}{O(5n)}=O(n^2).$$

Таким образом, для больших n и при наличии очень большого числа процессоров этот алгоритм дает очень высокое ускорение. Порядки линейных систем, для которых целесообразно применять параллельные компьютеры, по-видимому, должны находиться в пределах от 10^3 до 10^6. Даже при $n = 10^3$ потребуется 10^6 процессоров. Поэтому описанный алгоритм представляет главным образом теоретический интерес. Однако комбинация принципа потока данных с обсуждавшимися выше среднезернистыми алгоритмами может оказаться полезной. Например, в каждый процессор могут быть распределены несколько матричных элементов, а не один. Алгоритм подобного рода мог бы найти применение в таких системах, как Connection Machine (см. § 1.1), которая состоит сейчас примерно из 65 000 процессоров.

Разложение Холесского

Если A — симметричная положительно определенная матрица, то можно рассмотреть разложение Холесского $A = LL^T$; будем считать при этом, что хранится только нижняя треугольная часть матрицы A. Как и в случае LU-разложения, воспользуемся слоистой циклической схемой, строчной или столбцовой. Несколько возможных вариантов организации процесса дают ijk-формы разложения Холесского, которые довольно подробно обсуждаются в приложении 1 как для параллельных, так и для параллельно-векторных компьютеров. В таблице 2.2.2

Таблица 2.2.2. Разложение Холесского в случае параллельно-векторных систем ijk-формы

kij, *r*.	Большие расходы на обмены	Полные векторные длины при наличии задержек
kij, *c*	Неконкурентоспособен	
kji, *r*:	Большие расходы на обмены	Частичные векторные длины
kji, *c*:	Минимальные задержки	Полные векторные длины при наличии задержек
ikj, *r*·	Неконкурентоспособен	
ikj, *c*:	Неконкурентоспособен	
jki, *r*:	Очень большие расходы на обмены	Частичные векторные длины
jki, *c*:	Неконкурентоспособен	
ijk, *r*:	Дисбаланс нагрузки процессоров	
ijk, *c*:	Большие расходы на обмены	
jik, *r*:	Задержки на рассылку	Полные векторные длины
jik, *c*:	Неконкурентоспособен	

суммированы основные выводы этого приложения. Как и прежде, *kij*, *r* и *kij*, *c* обозначают форму *kij* соответственно со строчной и столбцовой слоистой циклической схемой хранения. Аналогичным образом обозначаются другие алгоритмы.

Наиболее привлекательными для параллельных систем кажутся формы *kji*, *c* и *jki*, *r*. Из них только форма *kji*, *c* допускает в случае параллельно-векторных систем длинные векторы, но это достигается ценой задержек для аккумулирования векторных данных.

Решение треугольных систем

Как мы знаем из предыдущего параграфа, после выполнения LU-разложения или разложения Холесского нужно решать треугольные системы уравнений. Мы рассмотрим только верхнетреугольную систему $U\mathbf{x} = \mathbf{c}$; решение нижнетреугольной системы организуется аналогичным образом (упражнение 2.2.8). Основными методами снова будут обсуждавшиеся в § 2.1 столбцовый алгоритм и алгоритм скалярных произведений. Их псевдокоды приведены на рис. 2.2.8.

Для $j = n$ с шагом -1 до 1	Для $i = n$ с шагом -1 до 1
$x_j = c_j / u_{jj}$	Для $j = i + 1$ до n
Для $i = 1$ до $j - 1$	$c_i = c_i - u_{ij} x_j$
$c_i = c_i - x_j u_{ij}$	$x_i = c_i / u_{ii}$

Рис. 2.2.8. Решение треугольной системы $U\mathbf{x} = \mathbf{c}$: a) столбцовый алгоритм; b) алгоритм скалярных произведений

Начнем со случая систем с локальной памятью. Если считать, что треугольная система является результатом проведенного ранее LU-разложения или разложения Холесского, то способ хранения U уже предопределен схемой хранения, использовавшейся при разложении. Так, если использовалась строчная циклическая слоистая схема (рис. 2.2.2), то таким же слоистым образом по строкам будет храниться матрица U. Предположим, что и правая часть $\mathbf{c}$ распределена по слоям, так что система в целом хранится, как показано на рис. 2.2.9, где $\mathbf{u}_i$ обозначает i-ю строку матрицы U. Одна из возможных реализаций столбцового алгоритма представлена псевдокодом на рис. 2.2.10. На первом шаге c_n и u_{nn} находятся в одном и том же процессоре. После того как значение x_n переслано остальным процессорам, каждый процессор пересчитывает хра-

нящиеся в нем компоненты правой части; затем в процессоре, содержащем $(n-1)$-ю строку, вычисляется x_{n-1}. В результате повторения этого процесса вычисляется x_{n-2}, потом x_{n-3} и т. д. При перевычислении величин c_i процессоры загружены равномерно до тех пор, пока треугольная система не редуцируется до малого размера; на заключительных же шагах все большее число процессоров прекращают работу.

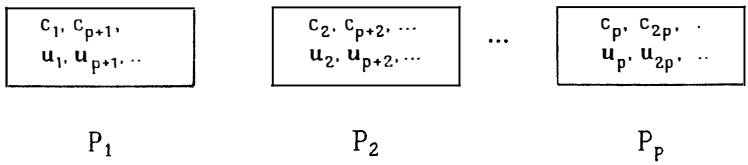


Рис. 2.2.9. Циклическая слоистая строчная схема хранения для системы $U\mathbf{x} = \mathbf{c}$

Недостаток столбцового алгоритма на рис. 2.2.10 заключается в том, что когда какой-то процессор вычисляет x_i, остальные процессоры простаивают. Можно попытаться совместить вычисление последующих значений x_i, применяя стратегию опережения: процессор, содержащий $(n-1)$-ю строку, модифицирует c_{n-1}, а затем, прежде чем модифицировать другие c_i, вычисляет и рассылает x_{n-1}. Следовательно, значение x_{n-1} будет

Вычислить $x_n = c_n/u_{nn}$ в $P(n)$.
Разослать x_n всем процессорам.
Вычислить $c_i = c_i - u_{in}x_n$, $i = 1, \ldots, n-1$.
Вычислить $x_{n-1} = c_{n-1}/u_{n-1,\,n-1}$ в $P(n-1)$.
Разослать x_{n-1} всем процессорам.
Вычислить $c_i = c_i - u_{i,\,n-1}x_{n-1}$, $i = 1, \ldots, n-2$.
. . . . . .

Рис. 2.2.10. Параллельный столбцовый алгоритм для матрицы U, хранимой по строкам

доступно всякому процессору в тот момент, когда он готов приступить к следующей модификации. Процессор, вычисливший x_{n-1}, закончит свои модификации первого шага позже других процессоров, но это отставание постепенно будет ликвидировано, поскольку при вычислении следующих значений x_i каждый процессор по очереди претерпит аналогичную задержку.

Если U хранится в соответствии со столбцовой циклической слоистой схемой, то прямолинейная реализация столбцового

алгоритма приводит по существу к последовательной программе. Предположим, что в процессоре $P(n)$, содержащем последний столбец матрицы U, хранится также и вектор **c**. Тогда $P(n)$ вычисляет x_n, пересчитывает **c** и пересылает новый вектор **c** процессору $P(n-1)$. Тот вычисляет x_{n-1}, модифицирует вектор **c**, пересылает его процессору $P(n-2)$, и т. д. Таким образом, в каждый момент времени вычисления производит лишь один процессор. Можно в какой-то мере поправить положение с помощью различных стратегий опережения (см. раздел «Литература и дополнения»). Однако мы перейдем к рассмотрению алгоритма скалярных произведений, поскольку при хранении U по столбцам он потенциально более привлекателен.

В алгоритме скалярных произведений на рис. 2.2.8b вычисление значения x_i включает в себя скалярное произведение

Для $i = n$ с шагом -1 до 1
Все процессоры вычисляют свою порцию i-го скалярного произведения; после сложения частичных скалярных произведений методом сдваивания сумма находится в $P(i)$
$P(i)$ вычисляет x_i

Рис. 2.2.11. Параллельный алгоритм скалярных произведений

i-й строки матрицы U, исключая диагональный элемент, и вектора с компонентами $x_{i+1}, \ldots, x_n$. Будем считать, что эти компоненты уже вычислены и $x_j \in P(j)$. Тогда процессор $P(j)$ может вычислить часть суммы $\sum u_{ij}x_j$, отвечающую хранящимся в нем значениям x_j, поскольку соответствующие u_{ij} также находятся в $P(j)$. После того как эти частичные скалярные произведения вычислены всеми процессорами параллельно, их можно просуммировать методом сдваивания, а результат поместить в $P(i)$. Предположим, что правая часть хранится таким образом, что $c_i \in P(i)$; в таком случае $P(i)$ вычисляет x_i. Этот алгоритм скалярных произведений представлен на рис. 2.2.11.

При больших n и i на этапе вычисления частичных скалярных произведений параллелизм почти идеален. Этап сдваивания не так хорош: он характеризуется обменами и возрастающим числом процессоров, прекративших работу. Кроме того, на первых шагах, когда i близко к n, параллелизм невелик и при вычислении частичных скалярных произведений. Некоторые из названных проблем в принципе можно смягчить, применяя стратегии опережающего вычисления. Например, на i-м шаге процессор $P(j)$, закончив свои вычисления, предусмотренные этим

шагом, может сразу начинать накапливание частичного скалярного произведения для следующего шага.

Заметим, что в системах с локальной памятью задаче решения треугольных систем внутренне присуща трудность, связанная с тем, что расходы на обмены амортизируются на сравнительно небольшом количестве $O(n^2)$ арифметических операций (ср. с $O(n^3)$ операциями при разложении).

Закончив разложение Холесского, мы должны решать треугольные системы $L\mathbf{y} = \mathbf{b}$ и $L^T\mathbf{x} = \mathbf{y}$. Если A хранится по строкам, то таким же образом будет храниться L. Для системы $L\mathbf{y} = \mathbf{b}$ можно предложить столбцовый алгоритм, аналогичный алгоритму на рис. 2.2.10; единственное отличие будет в том,

```
Если (строка n приписана данному процессору)
    x_n = c_n/u_nn
    маркировать x_n признаком «готово»
Для j = n − 1 с шагом −1 до 1
    ждать, пока не будет маркировано x_{j+1}
    Если (строка j приписана данному процессору)
        c_j = c_j − u_{j,j+1} x_{j+1}
        x_j = c_j/u_jj
    маркировать x_j признаком «готово»
Для i = 1 до j − 1
    Если (строка i приписана данному процессору)
        c_i = c_i − u_{i,j+1} x_{j+1}
```

Рис. 2.2.12. Столбцовый алгоритм для $U\mathbf{x} = \mathbf{c}$ в системе с разделяемой памятью

что неизвестные теперь определяются в порядке $y_1, \ldots, y_n$ (см. упражнение 2.2.8). При решении системы $L^T\mathbf{x} = \mathbf{y}$ хранению матрицы L по строкам соответствует хранение L^T по столбцам. Если же A хранилась по столбцам, то L будет храниться по столбцам, а L^T — по строкам. Итак, в любом случае потребуется решать треугольные системы и со строчной, и со столбцовой схемой хранения.

При применении гауссова исключения прямую подстановку (т. е. решение системы $L\mathbf{y} = \mathbf{b}$) обычно выполняют как часть процесса факторизации. Если используется строчное хранение, то элементы вектора $\mathbf{b}$ могут быть дописаны к строкам A; при хранении по столбцам $\mathbf{b}$ может рассматриваться как дополнительный столбец матрицы A.

При решении треугольных систем в машинах с разделяемой памятью может использоваться статическое или динамическое распределение заданий. Можно применить, например, строчную или столбцовую циклическую слоистую схему, согласно которой процессору i предназначается работа, связанная со строками (или столбцами) i, $i+p$, $i+2p$, ... Для строчной версии псевдокод, описывающий действия процессора, приведен на рис. 2.2.12.

Необходимая синхронизация достигается на рис. 2.2.12 посредством маркировки неизвестных x_i признаком «готово» по мере того, как эти неизвестные вычисляются. Процессоры ожидают, пока не будут готовы и, таким образом, доступны x_i, требующиеся для очередного вычисления. В зависимости от конкретной системы маркировка и ожидание могут быть реализованы другими способами. Чтобы x_i стало как можно скорее доступно другим процессорам, оно вычисляется прежде, чем будет завершен текущий пересчет правой части.

Метод Хаусхолдера

В предыдущем параграфе мы обсудили ортогональные методы Гивенса и Хаусхолдера применительно к векторным компьютерам. Теперь мы хотим изучить способы их реализации

Для $k=1$ до $n-1$

$$s_k = -\operatorname{sgn}(a_{kk})\left(\sum_{l=k}^{n} a_{lk}^2\right)^{1/2}, \quad \gamma_k = \left(s_k^2 - s_k a_{kk}\right)^{-1}$$

$$\mathbf{u}_k^{\mathrm{T}} = (0, \ldots, 0, a_{kk} - s_k, a_{k+1,\,k}, \ldots, a_{nk})$$

$$a_{kk} = s_k$$

Для $j=k+1$ до n

$$\alpha_j = \gamma_k \mathbf{u}_k^{\mathrm{T}} \mathbf{a}_j, \quad \mathbf{a}_j = \mathbf{a}_j - \alpha_j \mathbf{u}_k$$

Рис. 2.2.13. Метод Хаусхолдера в форме алгоритма скалярных произведений

для параллельных систем. Сначала рассмотрим метод Хаусхолдера. Пусть A хранится в соответствии с циклической слоистой столбцовой схемой. Будем исходить из векторной программы на рис. 2.1.12, которая для удобства читателя воспроизведена на рис. 2.2.13 (напомним, что $\mathbf{a}_i$ обозначает i-й столбец текущей матрицы A).

Для системы с локальной памятью первый шаг параллельной программы можно реализовать в соответствии с рис. 2.2.14.

Подшаги на этом рисунке организованы таким образом, чтобы как можно более равномерно загрузить процессоры. Например, пока процессор 1 вычисляет s_1, прочие процессоры могут начать вычисление скалярных произведений $\mathbf{u}_1^T\mathbf{a}_j$, хотя, разумеется, они не смогут завершить это вычисление раньше, чем им станет доступна величина s_1.

1. Послать первый столбец матрицы A всем процессорам.
2. Вычислить s_1 в процессоре 1. Начать вычисление $\mathbf{u}_1^T\mathbf{a}_j$ в других процессорах. Послать s_1 всем процессорам.
3. Вычислить γ_1 и $a_{11} - s_1$ во всех процессорах. Закончить вычисление $\mathbf{u}_1^T\mathbf{a}_j$. Заменить a_{11} на s_1 в процессоре 1.
4. Вычислить $\alpha_i = \gamma_1\mathbf{u}_1^T\mathbf{a}_i$ во всех процессорах.
5. Вычислить $\mathbf{a}_i - \alpha_i\mathbf{u}_1$ во всех процессорах.

Рис. 2.2.14. Первый шаг параллельного метода Хаусхолдера (алгоритм скалярных произведений)

Рассмотрим теперь другую форму метода Хаусхолдера, а именно алгоритм модификаций ранга 1; его псевдокод, представленный на рис. 2.1.13, воспроизведен на рис. 2.2.15. При этом явные формулы для s_k, γ_k и $\mathbf{u}_k$ опущены, поскольку они

Для $k = 1$ до $n - 1$
  $s_k,\ \gamma_k,\ \mathbf{u}_k$
  $\mathbf{v}_k^T = \sum_{j=k}^{n} u_{jk}\mathbf{a}_j$
  $\widehat{\mathbf{v}}_k^T = \gamma_k\mathbf{v}_k^T$
  Для $j = k$ до n
    $\mathbf{a}_j = \mathbf{a}_j - u_{jk}\widehat{\mathbf{v}}_k^T$

Рис. 2.2.15. Метод Хаусхолдера в форме алгоритма модификаций ранга 1

те же, что и в алгоритме скалярных произведений. Первый шаг соответствующей параллельной программы показан на рис. 2.2.16.

На рис. 2.2.15 через $\mathbf{a}_i$ обозначены строки матрицы A. Но так как, согласно предположению, A хранится по столбцам, то

вычисления подшага 4 на рис. 2.2.16 совпадают с вычислениями подшага 4 на рис. 2.2.14. Действительно, вычисление $\hat{\mathbf{v}}_1^{\mathrm{T}}$ сводится к умножению γ_1 на скалярные произведения $\mathbf{u}_1$ и столбцов матрицы A, хранимых каждым процессором. Одинаковы также вычисления на подшаге 5 обоих алгоритмов. Таким образом, параллельные формы на рис. 2.2.14 и 2.2.16 идентичны.

Если A хранится в соответствии со слоистой циклической строчной схемой, то параллельные алгоритмы далеко не так хороши, как в случае столбцового хранения. Рассмотрим вначале алгоритм скалярных произведений. Здесь нужны скаляр-

1. Послать первый столбец матрицы A всем процессорам.
2. Вычислить s_1 в процессоре 1. Начать вычисление $\mathbf{v}_1$ в других процессорах. Послать s_1 всем процессорам.
3. Вычислить γ_1 и $a_{11} - s_1$ во всех процессорах. Закончить вычисление $\mathbf{v}_1$ во всех процессорах. Заменить a_{11} на s_1 в процессоре 1.
4. Вычислить $\hat{\mathbf{v}}_1$ во всех процессорах.
5. Вычислить $\mathbf{a}_j - u_j\hat{\mathbf{v}}_1^{\mathrm{T}}$ во всех процессорах.

Рис. 2.2.16. Первый шаг параллельного метода Хаусхолдера (алгоритм модификаций ранга 1)

ные произведения $\mathbf{u}_k^{\mathrm{T}}\mathbf{a}_j$. Прежде всего вычисляются частичные скалярные произведения, получаемые из элементов, уже находящихся в каждом процессоре, но чтобы вычислить скалярное произведение полностью, требуется межпроцессорное суммирование сдваиванием. Затем каждое скалярное произведение нужно разослать всем процессорам. Алгоритм модификаций ранга 1 страдает тем же недостатком. Здесь нужно вычислить вектор $\mathbf{v}_k^{\mathrm{T}}$, представляющий собой линейную комбинацию строк матрицы A. Чтобы сделать это, опять-таки требуется применить межпроцессорный механизм сдваивания, а потом разослать результат всем процессорам. Мы приходим к выводу, что для метода Хаусхолдера столбцовый способ хранения выгодней строчного.

Метод Гивенса

Напомним (см. § 2.1), что на первом шаге метода Гивенса первые две строки матрицы A модифицируются по формулам

$$\hat{\mathbf{a}}_1 = c_{12}\mathbf{a}_1 + s_{12}\mathbf{a}_2, \quad \hat{\mathbf{a}}_2 = -s_{12}\mathbf{a}_1 + c_{12}\mathbf{a}_2, \tag{2.2.3}$$

где s_{12} и c_{12} — синус и косинус, определенные соотношениями (2.1.34). В результате этого шага в позиции (2, 1) матрицы A появляется нуль. На последующих шагах аннулируются остальные поддиагональные элементы первого столбца, затем поддиагональные элементы второго столбца, и т. д.

Предположим, что A хранится в соответствии со столбцовой циклической слоистой схемой. Начальные операции параллельного алгоритма показаны на рис. 2.2.17. Отметим, что в самом начале возникает задержка, связанная с вычислением

1. Вычислить c_{12} и s_{12} в процессоре 1. Разослать всем процессорам.
2. Начать модификацию строк 1 и 2 во всех процессорах.
3. Вычислить c_{13} и s_{13} в процессоре 1. Разослать всем процессорам.
4. Закончить модификацию строк 1 и 2 во всех процессорах.
5. Начать модификацию строк 1 и 3 во всех процессорах.

. . . . . .

Рис. 2.2.17. Столбцовая форма параллельного метода Гивенса

первой пары синус — косинус и пересылкой ее всем процессорам. В последующем мы вычисляем и рассылаем пары синусов — косинусов, как только готовы необходимые данные. Например, процессор 1, закончив модификацию элемента (1, 1) (см. пункт 2 на рисунке), может перейти к пункту 3, пока остальные процессоры продолжают пересчет строк 1 и 2. Во время первого этапа процессор 1 несет дополнительную нагрузку по вычислению синусов-косинусов, поэтому он закончит этап позже прочих процессоров. Однако теперь столбец 1 принял окончательный вид, в процессоре 1 столбцов стало на один меньше, и он постепенно ликвидирует отставание; в то же время на следующем этапе начинает отставать процессор 2, и т. д. В целом данная форма метода Гивенса обладает хорошим параллелизмом.

Этого нельзя сказать в случае строчной циклической слоистой схемы, если прямолинейно подойти к реализации метода. На первом шаге при обработке строк 1 и 2 будут активны только процессоры 1 и 2, на втором шаге активны лишь процессоры 1 и 3, и т. д. Положение улучшается при использовании другого типа параллелизма, внутренне присущего методу Гивенса; к обсуждению этого вопроса мы сейчас перейдем.

До сих пор при описании метода мы рассматривали обычный порядок исключения элементов в нижней треугольной части матрицы A. Этот порядок имеет следующее важное свойство: всякий полученный нуль остается нулем при дальнейших операциях. Однако существуют и другие порядки исключения, также обладающие этим свойством; условимся называть их *схемами аннулирования*. Мы хотим исследовать схемы аннулирования, позволяющие одновременное исключение нескольких элементов. Одна из таких схем иллюстрируется рисунком 2.2.18. На первом шаге комбинируются строки 1 и 2 с тем, чтобы получить нуль в первой позиции строки 2. Одновременно комбинируются строки 3 и 4, строки 5 и 6, и т. д. На втором шаге комбинируются строки 1 и 3, строки 5 и 7 и т. д. Таким образом, на каждом шаге аннулируется половина оставшихся ненулевых элементов первого столбца. Хотя первые шаги характеризуются высоким параллелизмом, в последующем используется все меньшее число процессоров. Однако в тот момент, когда начинаются простои процессоров, мы можем перейти к исключению элементов в столбце 2, оперируя строками, первые позиции которых уже аннулированы.

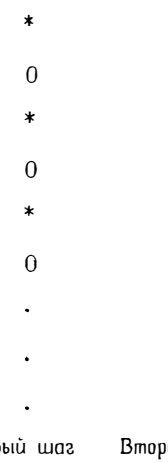


Рис. 2.2.18. Одновременное исключение элементов в методе Гивенса

Один из недостатков рассмотренной схемы заключается в очень большом объеме обмена данными. Так, если комбинирование строк 1 и 2 осуществляется процессором 1, то нужно послать строку 2 из процессора 2 в процессор 1, а после модификации вернуть вторую строку в процессор 2. Можно несколько поправить дело, начав с комбинирования строк, которые уже находятся в процессорах. Например, процессор 1 содержит строки 1 и $p+1$; их можно комбинировать без каких-либо пересылок. На следующем шаге можно комбинировать строки 1 и $2p+1$, и т. д. Аналогичным образом осуществляют комбинирование другие процессоры. Однако в какой-то момент все же придется комбинировать строки из разных процессоров. Даже с указанными улучшениями строчная схема хранения, по-видимому, не может конкурировать со столбцовой.

Рассмотрим теперь совершенно иную схему аннулирования для метода Гивенса. Иллюстрация к ней для матрицы 8×8 дана на рис. 2.2.19. Числами указаны номера шагов, на которых могут быть аннулированы соответствующие элементы из нижней строго треугольной части A, при этом преобразования Гивенса

строка							
2	7						
3	6	8					
4	5	7	9				
5	4	6	8	10			
6	3	5	7	9	11		
7	2	4	6	8	10	12	
8	1	3	5	7	9	11	13

Рис. 2.2.19. Схема аннулирования Самеха — Кука

сохраняют ранее полученные нули. На первом шаге комбинируются строки 7 и 8 с тем, чтобы получить нуль в позиции (8, 1). Затем, комбинируя строки 6 и 7, получаем нуль в позиции (7, 1). На третьем шаге можно получить нуль в позиции

строка							
2	$n-1$						
3	$n-2$	n					
.	.	$n-1$	$n+1$				
.	.		n				
.	.			.			
.	.				.		
.	.					.	
n	1	3	5	.	.	.	$2n-3$

Рис. 2.2.20. Схема аннулирования Самеха — Кука для четного n

(6, 1), комбинируя строки 5 и 6, и одновременно нуль в позиции (8, 2) за счет комбинирования строк 7 и 8, и т. д. Максимальная степень параллелизма достигается на седьмом шаге, когда параллельно получаются четыре нуля. Для произвольного (четного) n максимальная степень параллелизма, равная $n/2$, имеет место на шаге $n-1$ (см. рис. 2.2.20), а общее число

шагов равно $2n - 3$. Последнее верно не только для четного, но и для нечетного n. Однако в последнем случае максимальная степень параллелизма равна $(n-1)/2$ и достигается на двух шагах: $(n-1)$-м и n-м.

При наличии достаточно большого числа процессоров схема аннулирования Самеха — Кука позволяет получить очень высокое ускорение. Имея, например, $p = n^2/2$ процессоров, мы можем распределить между ними матрицу A следующим образом:

$$\begin{aligned} &P_1 - P_n: && \text{строки } 1,\ 2,\ 3 \\ &P_{n+1} - P_{2n}: && \text{строки } 3,\ 4,\ 5 \\ &\cdots\cdots\cdots\cdots\cdots\cdots && \\ &P_{p-n+1} - P_p: && \text{строки } n-1,\ n \end{aligned}$$

При этом процессор $P_i (1 \leqslant i \leqslant n)$ содержит элементы i-го столбца, находящиеся в строках 1, 2 и 3, процессор P_{n+i} — элементы i-го столбца из строк 3, 4, 5, и т. д. Используется схема аннулирования Самеха — Кука, но теперь модификация элементов строк на каждом шаге может производиться параллельно. Если игнорировать затраты времени на обмены и на вычисление синусов и косинусов, то средняя степень параллелизма в процессе равна $O(n^2/4)$.

Несколько правых частей

Закончим этот параграф несколькими замечаниями относительно системы $AX = B$, где X и B являются матрицами $n \times q$. Предположим, что любым из методов данного параграфа построено разложение $A = SU$, где U — верхнетреугольная матрица, а тип S определяется типом выбранного разложения. Тогда задача сводится к решению систем

$$SY = B, \quad UX = Y. \tag{2.2.4}$$

Рассмотрим вначале систему $UX = Y$. Предположим, что имеется q процессоров, матрица U хранится каждым из них и i-й столбец матрицы Y хранится процессором i. Тогда все q систем $U\mathbf{x}_i = \mathbf{y}_i$ могут быть решены одновременно. Аналогичным образом можно организовать решение системы $SY = B$. Итак, вычисления допускают максимальный параллелизм. Обсудим практическую сторону этого подхода.

Как уже говорилось, потребность в решении серии систем с одинаковой матрицей и различными правыми частями воз-

никает, в основном, в двух случаях. В первом случае $B = I$, $q = n$ и $X = A^{-1}$. Второй случай — это анализ поведения конструкций при разных нагрузках; есть и другие ситуации, схожие с данной. В подобных случаях q может принимать как малые, так и очень большие значения. Ясно, что обсуждавшийся выше подход не эффективен, если $q \ll p$ (где p — число процессоров), поскольку $p - q$ процессоров простаивают. Более целесообразно было бы решать системы посредством, например, столбцового алгоритма. Так, если $q = 5$ и $p = 20$, то для решения каждой системы можно было бы выделить по четыре процессора.

С другой стороны, если $p \ll q$, то решение q/p систем каждым процессором кажется вполне привлекательным. Основную трудность представляет следующее обстоятельство: если для приведения к треугольному виду использовались LU-разложение и слоистая строчная схема хранения, то матрица U будет распределена между процессорами по той же слоистой строчной схеме, а перед вычислениями потребуется собрать всю матрицу U в каждом из процессоров. Только детальный анализ для конкретной параллельной системы может показать, будет ли такая организация вычислений более эффективна, чем межпроцессорное решение каждой системы $U\mathbf{x}_i = \mathbf{y}_i$.

Упражнения к параграфу 2.2

1. Предположим, что параллельной системе требуется время t для выполнения арифметических операций и время αt для рассылки машинного слова из одного процессора произвольному числу других. Пусть число процессоров p равно порядку n системы. Подсчитать общее время LU-разложения без выбора главного элемента при использовании: a) строчной схемы хранения; b) столбцовой схемы (см. основной текст параграфа). Придти к заключению (зависящему от α) о том, какой способ хранения более выгоден.

2. Обсудить в деталях процедуру сдваивания для определения максимального из m чисел, распределенных между p процессорами.

3. Пусть параллельная система имеет те же характеристики, что и в упражнении 1, и пусть, кроме того, требуется время t для сравнения абсолютных величин двух чисел. Сопоставить два алгоритма LU-разложения, если проводится частичный выбор главного элемента.

4. Предположим, что параллельная система имеет те же характеристики, что и в упражнении 1, и пусть $n = kp$. Сравнить число временных шагов в LU-разложении без выбора главного элемента, если: a) первые k строк матрицы A хранятся в процессоре 1, следующие k строк — в процессоре 2, и т. д.; b) строки хранятся в соответствии со слоистой схемой на рис. 2.2.2. Повторить анализ, включив в LU-разложение частичный выбор главного элемента.

5. То же задание, что в упражнении 4, но для столбцовых схем хранения матрицы A.

6 Провести подробный разбор алгоритма потока данных (см. рис. 2.2.5) для матрицы

$$A = \begin{bmatrix} 4 & 1 & 1 & 1 \\ 1 & 4 & 1 & 1 \\ 1 & 1 & 4 & 1 \\ 1 & 1 & 1 & 4 \end{bmatrix}.$$

7 Написать псевдокод для формы *kji* разложения Холесского, не заглядывая в приложение 1. Затем сравнить свой псевдокод с тем, что приведен в приложении 1.

8 Обсудить столбцовый алгоритм и алгоритм скалярных произведений, аналогичные тем, что показаны на рис 2 2 8, для нижнетреугольной системы.

9. Показать, что прямолинейная реализация столбцового алгоритма для машины с разделяемой памятью и распределением заданий по слоистой столбцовой схеме приводит к сильному дисбалансу загрузки процессоров Обсудить способы повышения эффективности реализации алгоритма

Литература и дополнения к параграфу 2.2

1. Идея слоистого хранения была независимо высказана в работах [O'Leary, Stewart, 1985; Ipsen et al., 1986, Geist, Heath, 1986] Сам принцип был назван *тороидальным распределением* (torus assignment) в первом случае, *рассеянием* — во втором и *циклическим отображением* — в третьем. В первой статье, кроме того, рассматривается схема с отражениями

2 В ряде работ исследовались вопросы реализации LU-разложения или разложения Холесского для различных реально существующих или гипотетических параллельных систем В техническом отчете [Saad, 1986b] рассматриваются два способа реализации гауссова исключения для гиперкубов. В первом из них ведущая строка рассылается всем процессорам, которые в ней нуждаются, по принципу распространения Во втором способе рассылка ведущей строки организована по конвейерному принципу Когда какой-либо процессор получает ведущую строку, он вначале передает ее другому процессору, а затем продолжает вычисления Автор приходит к выводу, что конвейерная организация может быть более эффективной, даже если при распространении с выгодой используются особенности архитектуры гиперкуба. В техническом отчете [Le Blank, 1986] сообщается об экспериментах с гауссовым исключением, проводившихся для 128-процессорной машины BBN Butterfly. (Процессоры этой машины не имеют аппаратно реализованной вещественной арифметики, она моделировалась посредством целочисленной арифметики) Целью этого исследования было сравнение двух типов архитектуры, которые можно реализовать для Butterfly системы с локальной памятью и передачей сообщений и системы с разделяемой памятью В статье [Davis, 1986] описана реализация LU-разложения для гиперкуба Intel iPSC; она включает в себя выбор главного элемента и стратегии опережения, для хранения используется циклическая столбцовая схема. Джордж, Хит и Лю [George et al, 1986], имея в виду главным образом машину Denelcor HEP, изучают схемы разложения Холесского, максимально приспособленные для систем с разделяемой памятью Несколько схем основаны на ijk-формах метода Холесского, см по этому поводу также технический отчет [Heath, 1985]. В работе [Geist Heath, 1986] проведено детальное исследование формы *kji*, *с* разложения Холесского применительно к гиперкубу Intel iPSC (Авторы называют свою схему формой *jki*, но в окончательном алгоритме они выполняют модификации без задержек.) Сравниваются различные варианты распределения столбцов между процессорами, включая блочную циклическую

схему и случайное рассеяние; на основании экспериментов делается заключение, что циклическая столбцовая схема — наилучшая из проверявшихся. Приводятся, в частности, следующие данные о времени решения системы из 512 уравнений при 32 процессорах: 79.4 с — для столбцовой циклической схемы, 92 1 с — для блочной циклической схемы (в блоке 4 столбца), 174.7 с — для той же схемы при 16 столбцах в блоке, 94.1 с — при случайном рассеянии. Рассматриваются также различные стратегии обмена, например распространение, осуществляемое посылкой к каждому принимающему процессору, и распространение, использующее минимальное остовное дерево гиперкуба. Последняя стратегия имеет то достоинство, что требуется лишь $O(\log p)$ шагов, тогда как в первом случае — $O(p)$ шагов. (Однако при некоторых обстоятельствах более выгодными могут оказаться другие реализации.)

3. Проведен ряд исследований LU-разложения, имеющих целью выяснение принципиальных свойств его параллельных реализаций. В статье [Saad, 1986a] изучается коммуникационная сложность гауссова исключения для нескольких типов архитектуры. Один из главных результатов заключается в следующем: для схемы с общей шиной или кольцевой сети с локальной памятью LU-разложение матрицы $n \times n$ требует времени, равного по меньшей мере $O(n^2)$, независимо от числа процессоров. Сходные результаты приведены в работе [Ipsen et al., 1986]. В диссертации [Romine, 1986] дан подробный анализ ijk-форм LU-разложения. Автор заключает, что при определенных предположениях о параллельной системе форма kij оптимальна среди всех среднезернистых алгоритмов, использующих строчную циклическую схему хранения. См. также отчет [Ortega, Romine, 1987], на котором основывается раздел приложения 1, посвященный параллельным реализациям

4. Технический отчет [Geist, Romine, 1987] представляет собой обширное исследование стратегий выбора главного элемента при реализации LU-разложения для гиперкуба. Сравниваются четыре основных варианта: строчная схема хранения и выбор главного элемента по строке, столбцовая схема и выбор по строке, строчная схема и выбор по столбцу, столбцовая схема и выбор по столбцу. Для каждого основного варианта рассматриваются также различные усовершенствования (например, опережающее вычисление, развертка циклов с целью уменьшения количества индексной арифметики и т. п.). Как отмечалось в основном тексте параграфа, если строки не переставляются фактически, то происходит потеря строчного циклического шаблона хранения. В конечном счете строки будут распределены между процессорами по существу случайным образом. В работе [Geist, Heath, 1986] приводятся некоторые экспериментальные результаты для гиперкуба Intel iPSC, показывающие, что подобное распределение строк снижает скорость на 5—15 % в сравнении с циклическим хранением (см замечание 2 в этой работе). С другой стороны, в статье [Chu, George, 1987] рассматривается вариант с явными перестановками строк Чтобы уменьшить простои процессоров, ведущая строка рассылается до выполнения перестановки; таким образом, другие процессоры получают информацию, позволяющую начать модификации хранимых ими строк Авторы используют также динамическую схему балансировки нагрузки. См., кроме того, работы [Chamberlain, 1987], где применяется выбор главного элемента по строке, а не по столбцу, и [Sorensen, 1985]; в этой последней предложена парная схема, в которой на каждом шаге выбор главного элемента ограничен двумя строками.

5. LU-разложение по принципу потока данных, обсуждавшееся в основном тексте, подсказано аналогичной организацией разложения Холесского в статье [O'Leary, Stewart, 1985] См. также работу [Funderlic, Geist, 1986]. Она может служить примером изучения общего вопроса о том, насколько большого ускорения можно добиться в алгоритме при наличии произвольного или даже бесконечно большого числа процессоров. Большинство ранних

исследований по параллельным алгоритмам выполнялись в том же духе и, как правило, игнорировали время обменов. Хороший обзор этой начальной деятельности в области алгоритмов линейной алгебры дан в статье [Heller, 1978]. Среди более поздних работ, рассматривающих LU-разложение при большом числе процессоров (обычно $O(n)$ или $O(n^2)$ для системы $n \times n$), — [Bojanczyk, Brent, Kung, 1984; Kumar, Kowalik, 1984; Neta, Tai, 1985].

6. Алгоритм потока данных из основного текста параграфа служит, кроме того, примером алгоритма, организованного по принципу волнового фронта; этот принцип использовался для систолических массивов и других параллельных систем, допускающих реализацию посредством сверхбольших интегральных схем. Обзор алгоритмов для систолических массивов дан в работе [H. Kung, 1984], а обзор алгоритмов типа волнового фронта — в статье [S. Kung, 1984]. Из более свежих работ, связанных с этим подходом, отметим статью [Onaga, Takechi, 1986].

7. В статье [Shanehchi, Evans, 1982] анализируется так называемый QIF-метод (QIF — сокращение от Quadrant Interlocking Factorization. — *Перев.*). Он тесно связан с LU-разложением, но задуман именно для параллельных вычислений. Основная идея метода состоит в том, чтобы проводить исключение в матрице как сверху вниз, так и снизу вверх. Это приводит к разложению, форма которого показана на рисунке.

$$A = \begin{bmatrix} * & & O & & o \\ & \ddots & & & * \\ & & * & * & \\ & & & * & \\ * & & & & \\ o & & O & & * \end{bmatrix} \begin{bmatrix} * & & \cdots & & * \\ & \ddots & & \iddots & \\ O & & & & O \\ & & * & & \\ & \iddots & & \ddots & \\ * & & \cdots & & * \end{bmatrix}$$

8. Если матрица коэффициентов A имеет какую-либо специальную структуру, то систему $A\mathbf{x} = \mathbf{b}$ можно подчас решить значительно быстрее с помощью алгоритмов, использующих эту структуру. Например, *тёплицевой* называется матрица, вдоль диагоналей которой элементы сохраняют постоянные значения. Систему с такой матрицей можно решить не за $O(n^3)$, а за $O(n^2)$ операций. (Если не принимать во внимание вопрос о численной устойчивости, то системы с тёплицевыми матрицами можно решать еще быстрее. Существует ряд алгоритмов, называемых *сверхбыстрыми*, в которых для решения тёплицевой системы порядка n затрачивается $O(n \log^2 n)$ операций. — *Перев.*). Обзор параллельных методов для тёплицевых систем (с упором на систолические массивы) дан в работе [Delosme, Ipsen, 1987]. См., кроме того, статьи [Cohberg et al., 1987], где рассматриваются матрицы типа тёплицевых, и [Grear, Sameh, 1981].

9. Долгое время существовало убеждение, что для систем с локальной памятью решение треугольных систем с использованием столбцовой схемы хранения является принципиально трудной задачей. Недавно было замечено [Romine, Ortega, 1988], что алгоритм скалярных произведений потенциально очень хорош, а в статье [Li, Coleman, 1987a] был предложен эффективный способ реализации столбцового алгоритма. Оба этих алгоритма анализируются в отчете [Heath, Romine, 1987] наряду с реализациями по принципу волнового фронта столбцового алгоритма при столбцовой схеме хранения и алгоритма скалярных произведений при строчном хранении. Эти два последних алгоритма максимально используют совмещение операций; кроме того, ча-

стично вычисленные результаты «сегментами» пересылаются каждым процессором своему соседу в кольцевом порядке. «Циклические алгоритмы» также пользуются сегментами, но фиксированного размера $p-1$. Алгоритм Ли — Коулмена представляет собой циклическую реализацию столбцового алгоритма для столбцовой схемы хранения; аналогичным образом предлагается реализовать алгоритм скалярных произведений при строчном хранении в статье [Chamberlain, 1987].

Хит и Ромине провели обширные эксперименты для систем типа гиперкуб и пришли к следующим выводам:

a) Для строчной или столбцовой слоистой схемы хранения и малого ($\leqslant 16$) числа процессоров наилучшим является циклический подход.

b). Для строчной или столбцовой слоистой схемы и большого числа процессоров оптимален принцип волнового фронта.

c). Для случайной столбцовой слоистой схемы наилучшим будет алгоритм скалярных произведений с суммированием по методу сдваивания, организованным посредством минимального остовного дерева гиперкуба.

Еще более поздняя модификация циклического алгоритма [Li, Coleman, 1987b] значительно повысила его быстродействие при большом числе процессоров (см. также работу [Eisenstat et al., 1987]). Во всяком случае, теперь существуют хорошие алгоритмы и для строчного, и для столбцового хранения.

10. В статье [Sameh, 1985] описана версия метода Хаусхолдера для кольца процессоров. Матрица A поначалу находится в процессоре 1 (или же ее столбцы по одному загружаются в процессор 1 из центральной памяти). В процессоре 1 строится первое преобразование Хаусхолдера и производится модификация второго столбца. Новый второй столбец (начиная с диагонального элемента вниз) пересылается затем в процессор 2, этот процессор вычисляет второе преобразование Хаусхолдера. Одновременно процессор 1 продолжает пересчитывать столбцы матрицы A и посылать их (со второй позиции вниз) процессору 2. Тот пересчитывает эти столбцы и посылает их процессору 3, и т. д. В конце приведения процессор i будет хранить строки i, $i+p$, ... верхнетреугольной матрицы U, т. е. U хранится в соответствии со слоистой строчной схемой.

11. Для систем с разделяемой памятью и относительно малыми накладными расходами на синхронизацию в статье [Dongarra, Sameh, Sorensen, 1986] предложен еще один подход к параллельной реализации алгоритма Гивенса, так называемый «конвейерный метод Гивенса». Идея состоит в том, чтобы обрабатывать одновременно несколько строк, каждую с небольшим запаздыванием относительно предыдущей, чтобы сохранить надлежащую последовательность операций. См. также работу [Heath, Sorensen, 1986], где конвейерный метод Гивенса применяется к разреженным матрицам.

12. Метод Гивенса обладает естественным параллелизмом, проявляющимся в том, что одновременно можно исключить несколько элементов. Впервые это было отмечено в работе [Gentleman, 1975]. Конкретная схема аннулирования на рис. 2.1.19 принадлежит Самеху и Куку [Sameh, Kuck, 1978], но известен и ряд других схем. Так, в статье [Lord et al., 1980] предложены «зигзагообразная» схема, требующая $O(n/2)$ процессоров, по одному на каждую пару поддиагоналей матрицы, и «столбцовая» схема для малого числа процессоров. Другие схемы аннулирования и некоторые результаты по оптимальности можно найти в работах [Modi, Clark, 1984; Cosnard, Robert, 1986].

13. Перечислим другие сравнительно недавние работы по параллельным методам Гивенса и Хаусхолдера: [Bowgen, Modi, 1985], где сравниваются реализации обоих методов для машины ICL DAP и делается заключение, что метод Хаусхолдера в 1.5—2.3 раза быстрее метода Гивенса; [Barlow, Ipsen,

1984; Ipsen, 1984], где рассматриваются соответственно «масштабированный» и «быстрый» методы Гивенса; [Golub et al., 1986], где изучаются блочные методы ортогональной факторизации для задачи наименьших квадратов; и, наконец, [Luk, 1986]

2.3. Ленточные системы

В двух предыдущих параграфах предполагалось, что матрица A заполнена. Теперь мы изучим ленточные системы. Для простоты, как и в § 1.3, будем явно рассматривать только системы с симметричной лентой (полуширина ленты всюду обозначается через β), однако бо́льшая часть нашего обсуждения естественным образом распространяется на ленточные системы общего вида.

LU-разложение

Начнем с LU-разложения. На рис. 2.3.1 приведены псевдокоды с рисунков 2.1.1 и 2.1.2, адаптированные к системе с полушириной ленты β. Нетрудно построить адаптации и других ijk-форм LU-разложения из § 2.1 (см. упражнение 2.3.1).

```
Для k = 1 до n − 1
    Для i = k + 1
          до min (k + β, n)
        l_ik = a_ik/a_kk
        Для j = k + 1
              до min (k + β, n)
            a_ij = a_ij − l_ik a_kj
```

```
Для k = 1 до n − 1
    Для s = k + 1
          до min (k + β, n)
        l_sk = a_sk/a_kk
        Для j = k + 1
              до min (k + β, n)
            Для i = k + 1
                  до min (k + β, n)
                a_ij = a_ij − l_ik a_kj
```

Рис. 2.3.1. LU-разложение ленточной матрицы

На рис. 2.3.2 показаны схематически несколько первых шагов алгоритма kij для матрицы с полушириной ленты $\beta = 3$. При $k = 1$ из строк 2, 3, ..., $\beta + 1$ вычитаются кратные первой строки. Поскольку в первой строке и первом столбце только $\beta + 1$ элементов могут быть ненулевыми, то первый шаг затрагивает лишь первые $\beta + 1$ строк и столбцов матрицы A. Соответствующая подматрица на рис. 2.3.2 отмечена цифрой 1. Точно так же, во втором шаге участвуют только элементы 2, ..., $\beta + 2$ строк 2, ..., $\beta + 2$, и т. д. Продолжая процесс,

мы достигнем этапа, когда остается только подматрица $\beta \times \beta$, после чего разложение ведется как для заполненной матрицы.

В случае последовательных компьютеров ленточная матрица обычно хранится по диагоналям, однако этот способ не очень хорош для векторных компьютеров. Например, для алгоритма *kij* на рис. 2.3.1 матрицу естественно хранить по строкам. Однако использовать для этой цели полный двумерный $n \times n \times n$ массив неразумно, поскольку большая его часть не будет востребована, особенно при малом β. Альтернативный вариант состоит в том, чтобы хранить строки матрицы A в одномерном массиве; см. рис. 2.3.3, где через $\mathbf{a}_i$ обозначена i-я строка и рядом со строками указаны их длины. Если $\mathbf{a}_i$ интерпретировать как i-й столбец, то рис. 2.3.3 может служить иллюстрацией и к столбцовой схеме хранения.

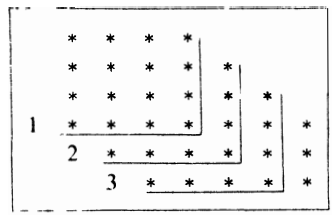


Рис. 2.3.2. Первые шаги LU-разложения для ленточной матрицы

При пересчетах строк или столбцов во внутренних циклах на рис. 2.3.1 могут использоваться векторы длины β до тех пор, пока не будет получена финальная подматрица $\beta \times \beta$, после

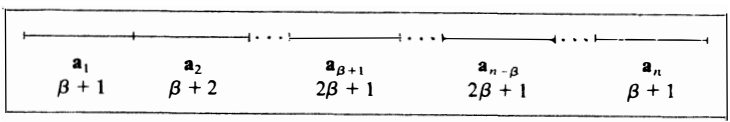


Рис. 2.3.3. Строчная (столбцовая) схема хранения ленточной матрицы

чего длины уменьшаются на 1 с каждым шагом. Таким образом, при достаточно большом β псевдокоды на рис. 2.3.1 могут служить основой потенциально эффективных алгоритмов для векторных компьютеров, однако при уменьшении β эффективность падает во все возрастающей степени. В частности, для трехдиагональных систем, где $\beta = 1$, эти алгоритмы совершенно неудовлетворительны. Позднее в этом параграфе мы рассмотрим другие алгоритмы, предназначенные для систем с малой шириной ленты.

Параллельная реализация

Перейдем теперь к вопросу о параллельной реализации LU-разложения. Как и в предыдущем параграфе, будем считать, что используется строчная или столбцовая циклическая слоистая

схема хранения, показанная на рис. 2.2.2. Соображения общего характера относительно LU-разложения, высказанные в § 2.2, применимы и к ленточному случаю, однако главный интерес сейчас представляет влияние ширины ленты на степень параллелизма.

Для алгоритмов на рис. 2.3.1 степень параллелизма в модификациях внутреннего цикла на протяжении почти всего процесса равна β, т. е. совпадает с длиной векторов. В частности, при слоистом распределении строк между процессорами на первом шаге нужно будет исключать элементы первого столбца в строках $2, \ldots, \beta + 1$. Эти β строк могут обрабатываться одновременно, если речь идет о форме kij; в форме kji одновременно могут обрабатываться соответствующие столбцы. Таким образом, если β меньше числа p процессоров, то на первом шаге будут работать только процессоры $2, \ldots, \beta + 1$. Поэтому для полного использования всех процессоров необходимо, чтобы выполнялось $\beta \geqslant p$. Заметим, что и при выполнении этого условия дисбаланс загрузки процессоров для ленточной системы гораздо более вероятен, чем для системы общего вида. Предположим, например, что имеется 50 процессоров, а $n = 10032$. Тогда 32 процессора получат по 201 строке, а 18 — по 200 строк. Для заполненной системы это различие очень мало. Пусть, однако, $\beta = 75$. В этом случае число строк, хранимых каждым процессором, по-прежнему равно 201 или 200, но при прямолинейной реализации алгоритма kij во второй части шага будет активна только половина процессоров. Очевидно, что наихудшим будет случай $\beta = p + 1$, так как при этом во второй части шага работает лишь один процессор. Проблему неравномерной загруженности процессоров можно в известной мере снять, применив стратегию опережающего вычисления. Например, как только на k-м шаге модифицирована строка $k + 1$, она рассылается остальным процессорам. Тогда, закончив свою работу на k-м шаге, процессор немедленно начинает $(k + 1)$-й шаг.

Разложение Холесского

К алгоритмам разложения Холесского, аналогичным тем, что приведены на рис. 2.3.1, применимы те же соображения. В частности, при $\beta < p$ на всех шагах разложения некоторые процессоры будут простаивать независимо от того, какая слоистая схема используется: строчная или столбцовая. Более детальное обсуждение алгоритмов Холесского вынесено в упражнения 2.3.5 и 2.3.6.

Треугольные системы

То, что сказано в § 2.1 о решении треугольных систем в случае векторных компьютеров, непосредственно переносится на ленточные системы. Пусть в системе $U\mathbf{x} = \mathbf{c}$ матрица U является ленточной с полушириной ленты β:

$$\begin{bmatrix} u_{11} & \dots & u_{1,\beta+1} & & \\ \ddots & & \ddots & & \\ & \ddots & & \ddots & \\ & & \ddots & & u_{n-\beta,n} \\ & & & & \vdots \\ & & & & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}. \qquad (2.3.1)$$

Если лента матрицы U хранится по столбцам, то столбцовый алгоритм с рис. 2.1.5 может быть записан как на рис. 2.3.4. Здесь $\mathbf{u}_i$ содержит элементы i-го столбца от верхней границы

```
Для i = n с шагом −1 до 1
    x_i = c_i/u_ii
    c = c − x_i u_i
```

Рис. 2.3.4. Столбцовый алгоритм

ленты до главной диагонали, но диагональный элемент в $\mathbf{u}_i$ не включается. Векторы будут иметь длину β, пока $i > \beta$. На участке $2 \leqslant i \leqslant \beta$ длины векторов с каждым шагом уменьшаются на 1. Итак, длины векторов в триадах на протяжении большей части процесса равны β и уменьшаются до 1 на заключительных шагах.

Если U хранится по строкам, то более целесообразно применять алгоритм скалярных произведений (2.1.10), который теперь принимает вид

$$x_i = (c_i - u_{i,i+1}x_{i+1} - \dots - u_{i,i+\beta}x_{i+\beta})/u_{ii}, \quad i = n-\beta, \dots, 1. \qquad (2.3.2)$$

Для $i = n, \dots, n-\beta+1$ формулу (2.3.2) нужно изменить с учетом того, что длина строки меньше чем $\beta + 1$. Алгоритм, основанный на (2.3.2), включает в себя скалярные произведения векторов длины β; для $i = n-1, \dots, n-\beta+1$ в

соответствующих скалярных произведениях длины векторов равны 1, ..., $\beta - 1$.

Как и в случае заполненных матриц, столбцовый алгоритм для некоторых векторных компьютеров может быть более привлекательным, поскольку использует триады, тогда как строчный алгоритм — скалярные произведения. Кроме того, для строчно-ориентированного алгоритма *kij* (см. рис. 2.3.1) и схемы хранения на рис. 2.3.3 элемент b_i правой части системы нельзя просто присоединить к i-й строке матрицы A, как это делалось для заполненных систем. Это более веский аргумент в пользу столбцовой схемы хранения, чем в случае заполненных матриц. Однако, как мы увидим ниже, для малых β и нескольких правых частей этот вывод уже несправедлив.

Вычислить $x_n = c_n/u_{nn}$. Разослать x_n всем процессорам.
Вычислить $c_i = c_i - u_{in}x_n$, $i = n - \beta, \ldots, n - 1$.
Вычислить $x_{n-1} = c_{n-1}/u_{n-1,\,n-1}$. Разослать x_{n-1} всем процессорам.
Вычислить $c_i = c_i - u_{i,\,n-1}x_{n-1}$, $i = n - \beta - 1, \ldots, n - 2$.
. . . . . .

Рис. 2.3.5. Параллельный столбцовый алгоритм для ленточных систем

Рассмотрим теперь параллельную машину с локальной памятью, и пусть A хранится в соответствии с циклической слоистой строчной схемой. Параллельный столбцовый алгоритм (см. рис. 2.2.10), адаптированный для ленточной системы, показан на рис. 2.3.5. Ясно, что в этом алгоритме степень параллелизма на начальных шагах равна β. Если $\beta < p$, то $p - \beta$ процессоров не используются. Но и при $\beta > p$ вполне вероятно, что загруженность процессоров не будет одинаковой, если только не прибегнуть к стратегии опережающего вычисления, которую мы обсуждали применительно к этапу LU-разложения. Тем не менее потенциально алгоритм вполне удовлетворителен. Если U хранится согласно циклической слоистой столбцовой схеме, то нетрудно адаптировать для ленточных систем алгоритм скалярных произведений с рис. 2.2.11.

Перестановки

Разложение Холесского и LU-разложение сохраняют ширину ленты, т. е. для верхнетреугольной матрицы U значение β оказывается таким же, как для A. Но перестановки строк, которые нужны при частичном выборе главного элемента, увели-

чивают ширину ленты; последствия этого для векторных и параллельных компьютеров более серьезны, чем для последовательных. Предположим, что на первом шаге были переставлены k-я и первая строки. Это показано на рисунке для матрицы с полушириной ленты 4 и $k = 5$.

```
* * * * * * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * *
```

В результате перестановки теперь при вычислении разложения во второй, третьей и четвертой строках, а также в новой пятой строке возникает заполнение. Таким образом ширина ленты над главной диагональю фактически удваивается; так будет и в дальнейшем, если потребуются дополнительные «наихудшие» перестановки.

Одно из возможных решений этой проблемы состоит в том, чтобы с самого начала отвести больше места для хранения матрицы. Так, если A хранится по строкам, то вместо схемы с

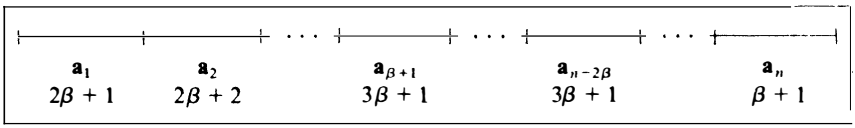


Рис. 2.3.6. Строчная схема хранения при наличии перестановок

рис. 2.3.3 следует применить схему с рис. 2.3.6: каждой строке выделяются дополнительно β ячеек памяти; начиная со строки $n - 2\beta + 1$, длина каждой очередной строки уменьшается на единицу. В LU-разложении с таким способом хранения не хотелось бы работать с длинами векторов, превышающими необходимую. Так, если на первом шаге нет перестановки, то, как и прежде, требуются только векторы длины β. Если же здесь или на дальнейших шагах перестановки нужны, то мы устанавливаем соответствующие длины векторов для последующих операций. Это увеличивает накладные расходы и усложняет программу. Однако, если перестановок немного или же они порождают сравнительно небольшое число дополнительных ненулевых элементов, то, вероятно, эти усложнения следует предпочесть работе с максимальными длинами векторов, указанными на рис. 2.3.6. Для столбцовой схемы хранения справедливы аналогичные соображения.

Ортогональные методы

При приведении матрицы к треугольному виду посредством преобразований Хаусхолдера или Гивенса наддиагональная часть ленты тоже расширяется. Рассмотрим вначале применение к A преобразования Хаусхолдера $I - \mathbf{w}\mathbf{w}^{\mathrm{T}}$, которое должно аннулировать в A поддиагональные элементы первого столбца. Тогда $\mathbf{w}$ имеет вид $\mathbf{w}^{\mathrm{T}} = (*, \ldots, *, 0, \ldots, 0)$, где первые $\beta + 1$ элементов в общем случае ненулевые. Согласно (2.1.27), новый i-й столбец матрицы A равен $\mathbf{a}_i - \mathbf{w}^{\mathrm{T}}\mathbf{a}_i\mathbf{w}$. Воздействие этого

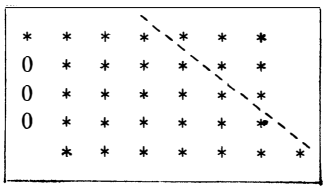


Рис. 2.3.7. Структура матрицы A после первого преобразования Хаусхолдера

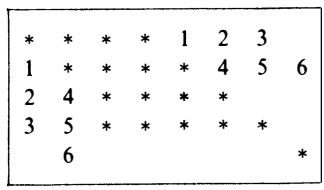


Рис. 2.3.8. Расширение ленты в методе Гивенса

преобразования на ленту показано на рис. 2.3.7 для $\beta = 3$. Вне исходной ленты, вообще говоря, возникают ненулевые элементы. На рисунке они расположены справа от пунктирной линии. Никаких других ненулевых элементов появиться не может, так как при первом преобразовании $\mathbf{w}^{\mathrm{T}}\mathbf{a}_i = 0$, если $i > 2\beta + 1$. При втором преобразовании Хаусхолдера ненулевые элементы будут введены в $(2\beta + 2)$-й столбец начиная со второй его позиции. При каждом последующем преобразовании ненулевые элементы появляются еще в одном столбце, поэтому в конце приведения верхнетреугольная матрица U имеет ленту ширины $2\beta + 1$.

То же самое происходит в методе Гивенса. Когда с помощью вращений аннулируются элементы первого столбца, в первую строку над лентой последовательно вносятся β ненулевых элементов. При исключении элементов второго столбца ненулевые элементы возникают вне ленты во второй строке, и т. д. На рис. 2.3.8 это показано для $\beta = 3$. Числами отмечено соответствие между появлением ненулевых элементов и аннулированием элементов в первых двух столбцах. Хотя порядок заполнения над лентой несколько иной по сравнению с методом Хаусхолдера, конечный результат такой же: ширина ленты верхней треугольной части матрицы A увеличивается до $2\beta + 1$.

Несколько правых частей

Пусть заданы несколько правых частей $B = (\mathbf{b}_1, \ldots, \mathbf{b}_q)$, причем матрицы A и B хранятся по столбцам. Алгоритм с рис. 2.3.1 b можно модифицировать таким образом, чтобы на этапе разложения обрабатывать q правых частей. Это, как и в случае заполненных матриц, добавляет q триад на каждом шаге разложения.

Если A хранится по строкам, то хотелось бы присоединить строки B к строкам A. Для заполненной матрицы A это увеличивает на q длины векторов на этапе разложения (см. § 2.1). Однако в случае ленточной матрицы мы встречаемся с проблемой, которую проиллюстрируем на примере двух первых строк:

$$\begin{array}{cccccccc} a_{11} & \ldots & a_{1,\beta+1} & b_{11} & & \ldots & b_{1q} \\ a_{21} & \ldots & a_{2,\beta+1} & a_{2,\beta+2} & b_{21} & \ldots & b_{2q} \end{array}$$

Из-за того, что длины строк в A неодинаковы, нет удобного доступа к столбцам матрицы B. Если β и q достаточно велики, мы можем отказаться от идеи приписывания строк B и оперировать с A и B раздельно.

Некоторый интерес представляет случай, когда β мало, но q настолько велико, что векторные операции длины q эффективны. Возможно, наиболее оправданным решением в такой ситуации является разложение матрицы A в скалярной арифметике и использование векторных операций для последующей одновременной обработки всех правых частей.

Блочные методы

Рассматривавшиеся до сих пор методы для ленточных систем удовлетворительны, пока полуширина ленты β достаточно велика. Однако для небольших значений β векторные длины слишком малы или, если речь идет о параллельной системе, процессоры используются не полностью. В отличие от обсуждавшейся в § 1.3 задачи матрично-векторного умножения, для этой задачи мы не располагаем эффективным алгоритмом исключения, опирающимся на диагональную схему хранения матрицы A.

Изучим теперь класс методов, предполагающих блочное разбиение матрицы коэффициентов A. Запишем ленточную систему

в блочной форме

$$\begin{bmatrix} A_1 & B_1 & & & \\ C_2 & A_2 & B_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & B_{p-1} \\ & & & C_p & A_p \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_p \end{bmatrix}, \tag{2.3.3}$$

где мы для простоты считаем, что $q = n/p$ — целое число и все матрицы A_i в (2.3.3) имеют размеры $q \times q$. Рис. 2.3.9 более подробно показывает устройство матриц A_i, B_i и C_i для $\beta = 2$. В общем случае B_i — нижнетреугольные, а C_i — верхнетреугольные матрицы.

Предположим, что все матрицы A_i невырожденны и допускают устойчивое LU-разложение. Так будет, в частности, если A_i — симметричные положительно определенные матрицы или матрицы с диагональным преобладанием. Решим системы

$$A_i W_i = B_i, \quad A_i V_i = C_i, \quad A_i \mathbf{d}_i = \mathbf{b}_i, \tag{2.3.4}$$

пользуясь разложениями

$$A_i = L_i U_i. \tag{2.3.5}$$

Если умножить обе части системы (2.3.3) на матрицу $\operatorname{diag}(A_1^{-1}, \ldots, A_p^{-1})$, получим систему

$$\begin{bmatrix} I & W_1 & & & \\ V_2 & I & W_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & W_{p-1} \\ & & & V_p & I \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \vdots \\ \mathbf{d}_p \end{bmatrix}. \tag{2.3.6}$$

Итак, решив системы (2.3.4), мы свели исходную систему (2.3.3) к виду (2.3.6).

В матрице W_i столбец j равен произведению A_i^{-1} на j-й столбец матрицы B_i. Хотя A_i — ленточная матрица, обратная к ней матрица A_i^{-1} в общем случае будет заполненной, поэтому j-й столбец в W_i заполнен, если соответствующий столбец в B_i ненулевой; таким образом, ненулевые столбцы матрицы B_i в W_i заполняются. Точно так же заполняются ненулевые столбцы матрицы C_i. Структура матрицы (2.3.6) для $\beta = 2$ показана на рис. 2.3.10.

В каждой матрице W_i (V_i) только первые (последние) β столбцов ненулевые. Представим эти матрицы в виде

$$W_i = \begin{bmatrix} W_{i1} & 0 \\ W_{i2} & 0 \\ W_{i3} & 0 \end{bmatrix}, \quad V_i = \begin{bmatrix} 0 & V_{i1} \\ 0 & V_{i2} \\ 0 & V_{i3} \end{bmatrix}, \quad \mathbf{x}_i = \begin{bmatrix} \mathbf{x}_{i1} \\ \mathbf{x}_{i2} \\ \mathbf{x}_{i3} \end{bmatrix}, \quad \mathbf{d}_i = \begin{bmatrix} \mathbf{d}_{i1} \\ \mathbf{d}_{i2} \\ \mathbf{d}_{i3} \end{bmatrix},$$

где W_{i1}, W_{i3}, V_{i1} и V_{i3} имеют размеры $\beta \times \beta$, а W_{i2} и V_{i2} — размеры $(q - 2\beta) \times \beta$. (Мы считаем, что $q > 2\beta$.) Аналогичным

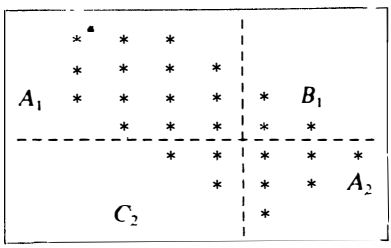


Рис. 2.3.9. Блочное разбиение для $\beta = 2$

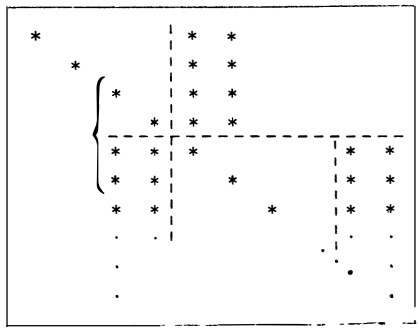


Рис. 2.3.10. Структура редуцированной системы

образом представлены векторы $\mathbf{x}_i$ и $\mathbf{d}_i$ из формул (2.3.6). Пользуясь этими представлениями, можно записать первое блочное уравнение $\mathbf{x}_1 + W_1\mathbf{x}_2 = \mathbf{d}_1$ системы (2.3.6) в виде

$$\begin{bmatrix} \mathbf{x}_{11} \\ \mathbf{x}_{12} \\ \mathbf{x}_{13} \end{bmatrix} + \begin{bmatrix} W_{11} \\ W_{12} \\ W_{13} \end{bmatrix} \mathbf{x}_{21} = \begin{bmatrix} \mathbf{d}_{11} \\ \mathbf{d}_{12} \\ \mathbf{d}_{13} \end{bmatrix}. \tag{2.3.7}$$

Аналогично, второе блочное уравнение $V_2\mathbf{x}_1 + \mathbf{x}_2 + W_2\mathbf{x}_3 = \mathbf{d}_2$ можно записать в виде

$$\begin{bmatrix} V_{21} \\ V_{22} \\ V_{23} \end{bmatrix} \mathbf{x}_{13} + \begin{bmatrix} \mathbf{x}_{21} \\ \mathbf{x}_{22} \\ \mathbf{x}_{23} \end{bmatrix} + \begin{bmatrix} W_{21} \\ W_{22} \\ W_{23} \end{bmatrix} \mathbf{x}_{31} = \begin{bmatrix} \mathbf{d}_{21} \\ \mathbf{d}_{22} \\ \mathbf{d}_{23} \end{bmatrix}. \tag{2.3.8}$$

То же самое делается для остальных блочных уравнений.

Заметим теперь, что уравнения

$$\begin{aligned} \mathbf{x}_{13} + W_{13}\mathbf{x}_{21} &= \mathbf{d}_{13}, \\ V_{21}\mathbf{x}_{13} + \mathbf{x}_{21} + W_{21}\mathbf{x}_{31} &= \mathbf{d}_{21}, \\ V_{23}\mathbf{x}_{13} + \mathbf{x}_{23} + W_{23}\mathbf{x}_{31} &= \mathbf{d}_{23}, \end{aligned} \tag{2.3.9}$$

не зависят от уравнений, содержащих векторы $\mathbf{x}_{i2}$; это значит, что ни в одно из уравнений системы (2.3.9) не входит никакой из векторов $\mathbf{x}_{12}$, $\mathbf{x}_{22}$, $\mathbf{x}_{32}$, ... Следовательно, систему (2.3,9) можно решить независимо от $\mathbf{x}_{i2}$. Мы назовем ее *редуцированной системой*. (В этом месте читателю, возможно, будет полезно проработать упражнение 2.3.9.) Как только система (2.3.9) решена, векторы $\mathbf{x}_{i2}$ можно найти из вторых уравнений в (2.3.7) и (2.3.8):

$$\mathbf{x}_{12} = \mathbf{d}_{12} - W_{12}\mathbf{x}_{21}, \quad \mathbf{x}_{22} = \mathbf{d}_{22} - V_{22}\mathbf{x}_{13} - W_{22}\mathbf{x}_{31}.$$

Вообще,

$$\mathbf{x}_{i2} = \mathbf{d}_{i2} - V_{i2}\mathbf{x}_{i-1,3} - W_{i2}\mathbf{x}_{i+1,1}. \tag{2.3.10}$$

Вектор $\mathbf{x}_{11}$ не присутствует в редуцированной системе; его можно определить из первого уравнения в (2.3.7): $\mathbf{x}_{11} = \mathbf{d}_{11} - W_{11}\mathbf{x}_{21}$. То же самое относится к вектору x_{p3}, который также не входит в редуцированную систему.

Шаг 1. Выполнить LU-разложения (2.3.5) и решить системы (2.3.4).

Шаг 2. Решить систему (2.3.9) относительно векторов $\mathbf{x}_{i1}$ и $\mathbf{x}_{i3}$.

Шаг 3. Вычислить векторы $\mathbf{x}_{i2}$ из равенств (2.3.10), а затем вычислить $\mathbf{x}_{11}$ и $\mathbf{x}_{p3}$.

Рис. 2.3.11. Блочный алгоритм Лори — Самеха

Процедура в целом показана на рис. 2.3.11. Ясно, что шаги 1 и 3 имеют высокую степень параллелизма. Если у нас есть p процессоров, то разложение (2.3.5) и решение уравнений (2.3.4) можно поручить i-му процессору $(1 \leqslant i \leqslant p)$. Аналогичным образом распараллеливается заключительный этап (2.3.10). Потенциально узким местом является решение редуцированной системы (2.3.9), и мы сейчас определим порядок этой системы.

Векторы $\mathbf{x}_{i1}$ и $\mathbf{x}_{i3}$ имеют длину β. Поэтому в систему (2.3.9) входит $2\beta(p-2)+2\beta = 2\beta(p-1)$ неизвестных (упражнение 2.3.10). В таблице 2.3.1 указан порядок редуцированной системы для различных значений β и p. Отметим еще, что матрица коэффициентов системы (2.3.9) имеет блочный пятидиагональный вид, а если ее рассматривать как ленточную, то полуширина ленты равна 3β (упражнение 2.3.10).

Таблица 2.3.1. Порядок редуцированной системы (2.3.9) для различных значений p и β

p	β				
	1	2	10	20	50
2	2	4	20	40	100
10	18	36	180	360	900
20	38	76	380	760	1900
50	98	196	980	1960	4900

Предположим, например, что $n = 10\,000$, $p = 20$, а $\beta = 2$. Тогда $q = 500$ и все матрицы A_i имеют размер 500×500. Порядок редуцированной системы равен 76, а полуширина ленты — 6. С другой стороны, если $p = 50$, а $\beta = 10$, то порядок матриц A_i равен только 200, тогда как порядок редуцированной системы — 980, а полуширина ленты — 30.

Теперь обсудим другой способ разбиения, для которого порядок редуцированной системы вдвое меньше. Будем, как и прежде, считать, что $q = n/p$ — целое число, и представим матрицы A_i из формулы (2.3.3) в виде

$$A_i = \begin{bmatrix} A_{i1} & A_{i2} \\ A_{i3} & A_{i4} \end{bmatrix}. \tag{2.3.11}$$

Размеры матриц A_{i4} и A_{i1} равны соответственно $\beta \times \beta$ и $(q-\beta) \times (q-\beta)$. Представим аналогичным образом матрицы B_i и C_i и векторы $\mathbf{x}_i$ и $\mathbf{b}_i$ в (2.3.3):

$$B_i = \begin{bmatrix} 0 & 0 \\ B_{i1} & B_{i2} \end{bmatrix}, \quad C_i = \begin{bmatrix} 0 & C_{i1} \\ 0 & C_{i2} \end{bmatrix}, \quad \mathbf{x}_i = \begin{bmatrix} \mathbf{x}_{i1} \\ \mathbf{x}_{i2} \end{bmatrix}, \quad \mathbf{b}_i = \begin{bmatrix} \mathbf{b}_{i1} \\ \mathbf{b}_{i2} \end{bmatrix}. \tag{2.3.12}$$

Отметим, что при $2\beta \leqslant q$ блоки B_{i2} и C_{i2} нулевые.

Выполним теперь LU-разложения $A_{i1} = L_{i1}U_{i1}$ и с помощью этих разложений решим системы

$$A_{i1}A_{i2}^1 = A_{i2}, \quad A_{i1}C_{i1}^1 = C_{i1}, \quad A_{i1}\mathbf{b}_{i1}^1 = \mathbf{b}_{i1}. \tag{2.3.13}$$

Решения этих уравнений позволяют осуществить неявное умножение исходной системы на матрицу $\operatorname{diag}(A_{11}^{-1}, I, A_{21}^{-1}, I, \ldots)$, что приводит к новой системе, которую мы проиллюстрируем

для случая $p = 3$ равенством (2.3.14):

$$\begin{bmatrix} I & A_{12}^1 & & & & \\ A_{13} & A_{14} & B_{11} & B_{12} & & \\ & C_{21}^1 & I & A_{22}^1 & & \\ & C_{22} & A_{23} & A_{24} & B_{21} & B_{22} \\ & & & C_{31}^1 & I & A_{32}^1 \\ & & & C_{32} & A_{33} & A_{34} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{11} \\ \mathbf{x}_{12} \\ \mathbf{x}_{21} \\ \mathbf{x}_{22} \\ \mathbf{x}_{31} \\ \mathbf{x}_{32} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{11}^1 \\ \mathbf{b}_{12} \\ \mathbf{b}_{21}^1 \\ \mathbf{b}_{22} \\ \mathbf{b}_{31}^1 \\ \mathbf{b}_{32} \end{bmatrix}. \quad (2.3.14)$$

Умножим первое блочное уравнение в (2.3.14) на A_{13}, а третье блочное уравнение на B_{11} и вычтем их из второго; в результате получим уравнение

$$A_{14}^1 \mathbf{x}_{12} + B_{12}^1 \mathbf{x}_{22} = \mathbf{b}_{12}^1,$$

где

$$A_{14}^1 = A_{14} - A_{13}A_{12}^1 - B_{11}C_{21}^1,$$

$$B_{12}^1 = B_{12} - B_{11}A_{22}^1, \quad \mathbf{b}_{12}^1 = b_{12} - A_{13}\mathbf{b}_{11}^1 - B_{11}\mathbf{b}_{21}^1.$$

Продолжая действовать таким же образом, умножим третье блочное уравнение на A_{23}, а пятое на B_{21}, и вычтем их из четвертого блочного уравнения. Наконец, вычитая из шестого блочного уравнения пятое, умноженное на A_{33}, придем к системе

$$\begin{bmatrix} I & A_{12}^1 & & & & \\ & A_{14}^1 & 0 & B_{12}^1 & & \\ & C_{21}^1 & I & A_{22}^1 & 0 & \\ & C_{22}^1 & 0 & A_{24}^1 & 0 & B_{22}^1 \\ & & & C_{31}^1 & I & A_{32}^1 \\ & & & C_{32}^1 & 0 & A_{34}^1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_{11} \\ \mathbf{x}_{12} \\ \mathbf{x}_{21} \\ \mathbf{x}_{22} \\ \mathbf{x}_{31} \\ \mathbf{x}_{32} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{11}^1 \\ \mathbf{b}_{12}^1 \\ \mathbf{b}_{21}^1 \\ \mathbf{b}_{22}^1 \\ \mathbf{b}_{31}^1 \\ \mathbf{b}_{32}^1 \end{bmatrix}. \quad (2.3.15)$$

Мы видим, что в (2.3.15) блочные уравнения с четными номерами не зависят от остальных уравнений. Это позволяет сформировать редуцированную систему

$$\begin{bmatrix} A_{14}^1 & B_{12}^1 & \\ C_{22}^1 & A_{24}^1 & B_{22}^1 \\ & C_{32}^1 & A_{34}^1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_{12} \\ \mathbf{x}_{22} \\ \mathbf{x}_{32} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{12}^1 \\ \mathbf{b}_{22}^1 \\ \mathbf{b}_{32}^1 \end{bmatrix}. \quad (2.3.16)$$

Как только система (2.3.16) решена, векторы $\mathbf{x}_{i1}$ определяются из уравнений системы (2.3.15) с нечетными номерами:

$$\mathbf{x}_{i1} = \mathbf{b}^{!}_{i1} - A^{!}_{i2}\mathbf{x}_{i2} - C^{!}_{i1}\mathbf{x}_{i-1,2}, \quad i = 1, \ldots, p. \qquad (2.3.17)$$

При $i = 1$ последний член в правой части отсутствует.

Уравнения (2.3.14)—(2.3.16) описывают процесс для $p = 3$, однако способ его обобщения для произвольного p очевиден. В частности, редуцированная система (2.3.16) в общем случае

Шаг 1. Выполнить разложения $A_{i1} = L_{i1}U_{i1}$ и решить системы (2.3.13).
Шаг 2. Решить редуцированную систему (2.3.16) (а в общем случае блочно трехдиагональную систему блочного порядка p).
Шаг 3. Вычислить остальные векторы $\mathbf{x}_{i1}$ по формулам (2.3.17).

Рис. 2.3.12. Блочный алгоритм Джонсона

представляет собой блочно трехдиагональную систему блочного порядка p с блоками размера $\beta \times \beta$; таким образом, полуширина ленты равна $2\beta - 1$. Процедура в целом показана на рис. 2.3.12. Как и в методе Лори — Самеха, шаги 1 и 3 обладают высокой степенью параллелизма, а потенциально узким местом является шаг 2. Однако редуцированная система в методе Джонсона содержит только βp уравнений (ср. с $2\beta(p-1)$ уравнениями в методе Лори — Самеха).

Методы декомпозиции области

Обсудим еще один класс методов, которые в некоторых отношениях схожи с блочными методами. Основную идею отражает трехдиагональная матрица

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & \cdot & \cdot & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & -1 \\ & & & -1 & 2 \end{bmatrix}. \qquad (2.3.18)$$

Такая матрица возникает, например, при численном решении двухточечной краевой задачи для дифференциального уравнения

$$x''(t) = f(t), \quad a \leqslant t \leqslant b, \quad x(a) = \alpha, \quad x(b) = \gamma. \quad (2.3.19)$$

Интервал $[a, b]$ заменяется дискретной сеткой из равноудаленных узлов x_i (см. рис. 2.3.13). Вторая производная в (2.3.19) дискретизуется по обычной центральной разностной формуле.

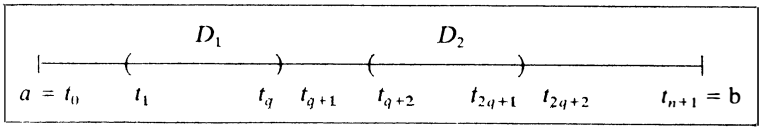


Рис. 2.3.13. Узлы сетки и декомпозиция области

В результате в каждой точке сетки получаем аппроксимирующее уравнение

$$x_{i+1} - 2x_i + x_{i-1} = h^2 f_i, \quad i = 1, \ldots, n, \quad (2.3.20)$$

где h — расстояние между узлами сетки. Поскольку x_0 и x_n — это заданные краевые значения, то (2.3.20) есть система линейных уравнений относительно n неизвестных $x_1, \ldots, x_n$, аппроксимирующих решение задачи (2.3.19) в узлах сетки. Если умножить уравнения (2.3.20) на -1, то (2.3.18) совпадает с матрицей коэффициентов полученной системы.

Разобьем теперь узлы сетки t_i на группы, как это показано на рис. 2.3.13. К группе D_1 отнесем узлы $t_1, \ldots, t_q$, к группе D_2 — узлы $t_{q+2}, \ldots, t_{2q+1}$, и т. д. Для простоты будем считать, что $n = pq + p - 1$. Тогда мы получим p множеств $D_1, \ldots, D_p$ (каждое из них содержит q узлов) и $p - 1$ узлов $t_{q+1}, t_{2q+2}, \ldots$, находящихся между множествами D_i. Эти $p - 1$ узлов составляют *множество-разделитель* S. Перенумеруем неизвестные x_i, присвоив последние номера узлам множества S. Новая нумерация неизвестных, отождествленных с узлами сетки, показана на рис. 2.3.14 для случая $q = 2$, $p = 3$.

Выпишем уравнения (2.3.20) при новом упорядочении неизвестных:

$$\begin{aligned}
x_0 - 2x_1 + x_2 &= h^2 f_1, & x_8 - 2x_5 + x_6 &= h^2 f_5,\\
x_1 - 2x_2 + x_7 &= h^2 f_2, & x_5 - 2x_6 + x_9 &= h^2 f_6,\\
x_7 - 2x_3 + x_4 &= h^2 f_3, & x_2 - 2x_7 + x_3 &= h^2 f_7,\\
x_3 - 2x_4 + x_8 &= h^2 f_4, & x_4 - 2x_8 + x_5 &= h^2 f_8.
\end{aligned} \quad (2.3.21)$$

Отметим, что уравнения, отвечающие узлам разделителя, выписаны последними. В матричной форме система (2.3.21) выглядит так:

$$\begin{bmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \cdot & & \cdot \\ & & & \cdot & \cdot \\ & & & \cdot & \cdot \\ & & & A_p & B_p \\ C_1 & C_2 & \dots & C_p & A_S \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{x}_p \\ \mathbf{x}_S \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{b}_p \\ \mathbf{b}_S \end{bmatrix} \qquad (2.3.22)$$

Здесь $p=3$, $C_i=B_i^{\mathrm{T}}$ $(i=1, 2, 3)$ и

$$A_1=A_2=A_3=\begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}, \quad A_S=\begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix},$$

$$B_1=\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \quad B_2=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad B_3=\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

В общем случае A_i — трехдиагональные матрицы $q \times q$, A_S — диагональная матрица $(p-1)\times(p-1)$, а B_i — матрицы

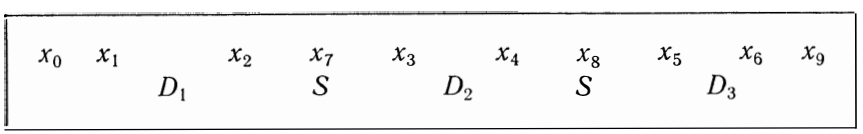


Рис. 2.3.14. Переупорядочение неизвестных

$q \times (p-1)$. Матрицу системы (2.3.22) иногда называют *стреловидной.*

Прежде чем более подробно обсудить свойства системы (2.3.22), вернемся к случаю ленточной матрицы общего вида с полушириной ленты β. Такая матрица может и не соответствовать дискретизации дифференциального уравнения, однако мы применим к ней те же построения, что и выше. Разобьем неизвестные на $p+1$ множеств $D_1, \dots, D_p$ и S так, чтобы в каждом множестве D_i было q неизвестных, а неизвестные из S «разделяли» неизвестные из множеств D_i следующим образом. По аналогии с рис. 2.3.13 можно представить себе, что неизвестные находятся на прямой; тогда расположение множеств должно подчиняться схеме

$$D_1 S_1 D_2 S_2 \dots S_{p-1} D_p. \qquad (2.3.23)$$

Каждое множество S_i в (2.3.23) содержит β неизвестных. В системе $Ax = b$ j-е уравнение имеет вид

$$a_{jj}x_j + \sum_{\substack{k=j-\beta \\ k \neq j}}^{j+\beta} a_{jk}x_k = b_j. \tag{2.3.24}$$

Поэтому, если $x_j \in D_i$, то никакое неизвестное в уравнении (2.3.24) не может принадлежать какому-либо другому множеству D_k. Множества S_i разделяют неизвестные в D_i в том смысле, что любое уравнение системы содержит неизвестные только из одного множества D_i.

Если теперь перенумеровать неизвестные так, чтобы последние номера были присвоены неизвестным из множеств-разделителей, и записать уравнения в соответствующем порядке, то получится новая система вида (2.3.22); в ней $A_1, \ldots, A_p$ имеют порядок q, а A_S — порядок s, где $s = \beta(p-1)$ есть число неизвестных в множестве-разделителе $S = \bigcup S_i$. Матрицы B_i и C_i имеют соответственно размеры $q \times s$ и $s \times q$. Для простоты мы предположили, что $n = pq + s$.

Введем обозначения

$$A_I = \operatorname{diag}(A_1, \ldots, A_p), \quad B^{\mathrm{T}} = (B_1^{\mathrm{T}}, \ldots, B_p^{\mathrm{T}}), \quad C = (C_1, \ldots, C_p). \tag{2.3.25}$$

Тогда систему (2.3.22) можно записать в виде

$$A_I\mathbf{x}_I + B\mathbf{x}_S = \mathbf{b}_I, \tag{2.3.26a}$$

$$C\mathbf{x}_I + A_S\mathbf{x}_S = \mathbf{b}_S, \tag{2.3.26b}$$

где $\mathbf{x}_I^{\mathrm{T}} = (\mathbf{x}_1^{\mathrm{T}}, \ldots, \mathbf{x}_p^{\mathrm{T}})$, $\mathbf{b}_I^{\mathrm{T}} = (\mathbf{b}_1^{\mathrm{T}}, \ldots, \mathbf{b}_p^{\mathrm{T}})$. Предположим, что матрица A_I невырожденна. Если умножить (2.3.26a) на CA_I^{-1} и вычесть из (2.3.26b), то получится уравнение

$$\hat{A}\mathbf{x}_S = \hat{\mathbf{b}}, \quad \hat{A} = A_S - CA_I^{-1}B, \quad \hat{\mathbf{b}} = \mathbf{b}_S - CA_I^{-1}b_I. \tag{2.3.27}$$

Заметим, что именно к такому уравнению приводит блочное гауссово исключение для системы (2.3.26). Матрица $\hat{A}$ называется *преобразованием Гаусса* или *дополнением Шура* подматрицы A_I. Как только система (2.3.27) решена относительно $\mathbf{x}_S$, остальные $\mathbf{x}_i$ могут быть найдены из систем

$$A_i\mathbf{x}_i = \mathbf{b}_i - B_i\mathbf{x}_S, \quad i = 1, \ldots, p. \tag{2.3.28}$$

Обсудим этот процесс более подробно. Будем считать, что матрицы A_i допускают устойчивые LU-разложения

$$A_i = L_iU_i, \quad i = 1, \ldots, p. \tag{2.3.29}$$

Пусть решены системы

$$L_i Y_i = B_i, \quad L_i \mathbf{y}_i = \mathbf{b}_i, \quad i = 1, \ldots, p, \tag{2.3.30}$$

$$U_i Z_i = Y_i, \quad U_i \mathbf{z}_i = \mathbf{y}_i, \quad i = 1, \ldots, p. \tag{2.3.31}$$

Поскольку

$$C_i A_i^{-1} B_i = C_i (L_i U_i)^{-1} B_i = C_i U_i^{-1} Y_i = C_i Z_i$$

и, аналогично, $C_i A_i^{-1} \mathbf{b}_i = C_i \mathbf{z}_i$, то можно написать

$$\hat{A} = A_S - \sum_{i=1}^{p} C_i A_i^{-1} B_i = A_S - \sum_{i=1}^{p} C_i Z_i \tag{2.3.32}$$

и

$$\hat{\mathbf{b}} = \mathbf{b}_S - \sum_{i=1}^{p} C_i A_i^{-1} \mathbf{b}_i = \mathbf{b}_S - \sum_{i=1}^{p} C_i \mathbf{z}_i. \tag{2.3.33}$$

Эти формулы показывают, как можно вычислить $\hat{A}$ и $\hat{\mathbf{b}}$. Общее описание процедуры приводится на рис. 2.3.15.

Шаг 1. Выполнить разложения (2.3.29) и решить системы (2.3.30) и (2.3.31).
Шаг 2. Сформировать произведения $C_i Z_i$ и $C_i \mathbf{z}_i$, $i = 1, \ldots, p$.
Шаг 3. Сформировать $\hat{A}$ и $\hat{\mathbf{b}}$ и решить систему $\hat{A}\mathbf{x}_S = \hat{\mathbf{b}}$.
Шаг 4. Сформировать векторы $\mathbf{c}_i = \mathbf{b}_i - B_i \mathbf{x}_S$, $i = 1, \ldots, p$.
Шаг 5. Решить системы $A_i \mathbf{x}_i = \mathbf{c}_i$, $i = 1, \ldots, p$, пользуясь разложениями (2.3.29).

Рис. 2.3.15. Алгоритм декомпозиции области

Ясно, что шаги 1, 2, 4 и 5 имеют высокую степень параллелизма. Рассмотрим, например, систему с локальной памятью, и пусть A_i, B_i, C_i и $\mathbf{b}_i$ приписаны к процессору i ($i = 1, \ldots, p$). Тогда все вычисления первых двух шагов параллельны и не требуют обмена данными. Потенциально узким местом является шаг 3. Можно решить систему $\hat{A}\mathbf{x}_S = \hat{\mathbf{b}}$ в каком-то одном процессоре, скажем, P_1. В этом случае суммирование в формулах (2.3.32)—(2.3.33) следует организовать с помощью межпроцессорной процедуры сдваивания, причем так, чтобы конечные суммы оказались в P_1. Другая возможность — решать систему, используя гауссово исключение и слоистую схему хранения. Теперь суммирование в (2.3.32) нужно проводить так, чтобы

первая строка матрицы $\hat{A}$ накапливалась в процессоре 1, вторая строка — в процессоре 2 и т. д. В любом случае после того, как получен вектор x_S, нужно разослать его копии всем процессорам. Затем шаги 4 и 5 можно выполнять параллельно, и здесь снова не требуется обмена данными.

Уже отмечалось, что матрица A_S имеет порядок s, где $s = \beta(p-1)$ (ср. с порядком $2\beta(p-1)$ редуцированной системы (2.3.9) блочного метода Лори — Самеха и с порядком βp для метода Джонсона).

Теперь мы укажем, какие изменения нужно внести в алгоритм декомпозиции области в важном частном случае, когда матрица A симметрична и положительно определена. Поскольку система (2.3.22) получена из исходной путем перестановок уравнений и перенумерации неизвестных, то ее матрица, которую мы обозначим через $\bar{A}$, связана с A соотношением $\bar{A} = PAP^{\mathrm{T}}$, где P — матрица перестановки. Поэтому $\bar{A}$ — симметричная положительно определенная матрица, и теми же свойствами обладают матрицы A_i и A_S. На шаге 1 алгоритма (см. рис. 2.3.15) мы теперь скорей всего применим разложение Холесского $A_i = L_i L_i^{\mathrm{T}}$. Так как, по симметрии, $C_i = B_i^{\mathrm{T}}$, то

$$\hat{A} = A_S - \sum_{i=1}^{p} B_i^{\mathrm{T}} A_i^{-1} B_i, \quad \hat{\mathbf{b}} = \mathbf{b}_S - \sum_{i=1}^{p} B_i^{\mathrm{T}} A_i^{-1} b_i.$$

Но

$$B_i^{\mathrm{T}} A_i^{-1} B_i = B_i^{\mathrm{T}} \left(L_i L_i^{\mathrm{T}}\right)^{-1} B_i = Y_i^{\mathrm{T}} Y_i.$$

Полагая $\mathbf{y}_i = L_i^{-1}\mathbf{b}_i$, получаем

$$\hat{A} = A_S - \sum_{i=0}^{p} Y_i^{\mathrm{T}} Y_i, \quad \hat{\mathbf{b}} = \mathbf{b}_S - \sum_{i=1}^{p} Y_i^{\mathrm{T}} y_i. \tag{2.3.34}$$

Важно отметить следующее обстоятельство: если A — симметричная положительно определенная матрица, то это же верно по отношению к матрице $\hat{A}$. Симметричность последней матрицы очевидна. Что касается положительной определенности, то пусть $\mathbf{x}_2$ — произвольный ненулевой вектор размерности s. Положим $\mathbf{x}_1 = -A_I^{-1} B\mathbf{x}_2$. Тогда

$$0 < (\mathbf{x}_1^{\mathrm{T}} \mathbf{x}_2^{\mathrm{T}}) \begin{bmatrix} A_I & B \\ B^{\mathrm{T}} & A_S \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}$$

в силу положительной определенности матрицы A. Расписывая это неравенство более подробно, имеем

$$0 < \mathbf{x}_1^{\mathrm{T}} A_I \mathbf{x}_1 + 2\mathbf{x}_1^{\mathrm{T}} B\mathbf{x}_2 + \mathbf{x}_2^{\mathrm{T}} A_S \mathbf{x}_2 = \mathbf{x}_2^{\mathrm{T}} \hat{A} \mathbf{x}_2,$$

а это означает, что $\hat{A}$ положительно определена. Алгоритм декомпозиции области для случая симметричных положительно определенных матриц представлен на рис. 2.3.16. Как и в ранее рассмотренных алгоритмах, шаги 1 и 3 обладают высокой степенью параллелизма, а шаг 2 потенциально является узким местом.

Между алгоритмами декомпозиции области и блочным методом Джонсона (рис. 2.3.12) существует очень тесная связь.

Шаг 1. Выполнить разложения Холесского $A_i = L_i L_i^T$ и решить системы $L_i Y_i = B_i$, $L_i \mathbf{y}_i = \mathbf{b}_i$, $i = 1, \ldots, p$.

Шаг 2. Сформировать $\hat{A}$ и $\hat{\mathbf{b}}$ по формулам (2.3.34) и решить систему $\hat{A}\hat{\mathbf{x}}_S = \hat{\mathbf{b}}$.

Шаг 3. Сформировать векторы $\mathbf{c}_i = \mathbf{b}_i - B_i \mathbf{x}_S$ и решить системы $A_i \mathbf{x}_i = \mathbf{c}_i$, $i = 1, \ldots, p$.

Рис. 2.3.16. Алгоритм декомпозиции области для случая симметричной положительно определенной матрицы

В самом деле, метод Джонсона можно рассматривать как алгоритм декомпозиции области, в который введен дополнительный разделитель на правом конце (т. е. схема (2.3.23) в случае метода Джонсона должна была бы заканчиваться множеством S_p, а не D_p) и неизвестные не переупорядочиваются. Отсюда следует (см. упражнение 2.3.13), что редуцированная система в методе Джонсона симметрична и положительно определена, если это верно для исходной системы. Метод Лори — Самеха не всегда обладает этим свойством. Заметим еще, что в методе Лори — Самеха нет множеств-разделителей.

Трехдиагональные системы и циклическая редукция

И блочные методы, и методы декомпозиции области в принципе применимы к трехдиагональным системам (см. упражнение 2.3.14). Рассмотрим еще один подход к решению трехдиагональных систем, которые мы теперь будем записывать в виде

$$\begin{aligned} a_1 x_1 + b_1 x_2 &= d_1, \\ c_2 x_1 + a_2 x_2 + b_2 x_3 &= d_2, \\ c_3 x_2 + a_3 x_3 + b_3 x_4 &= d_3, \\ &\cdots\cdots\cdots \\ c_n x_{n-1} + a_n x_n &= d_n. \end{aligned} \tag{2.3.35}$$

Умножим первое уравнение системы (2.3.35) на c_2/a_1 и вычтем его из второго, чтобы аннулировать коэффициент при x_1 во втором уравнении; это обычный шаг гауссова исключения. Но затем мы умножаем третье уравнение на b_2/a_3 и также вычитаем его из второго, чтобы исключить коэффициент при x_3 во втором уравнении. Так получается новое второе уравнение

$$a_2'x_2 + b_2'x_4 = d_2'.$$

Проделаем то же самое с уравнениями 3, 4 и 5: вычтем из уравнения 4 надлежащие кратные уравнений 3 и 5, чтобы получить новое уравнение

$$c_4'x_2 + a_4'x_4 + b_4'x_6 = d_4',$$

в котором по сравнению с исходным четвертым уравнением аннулированы коэффициенты при x_3 и x_5. Продолжим работать таким образом с перекрывающимися тройками уравнений; при этом каждый раз получается новое среднее уравнение, где исключены переменные с нечетными номерами. На рис. 2.3.17

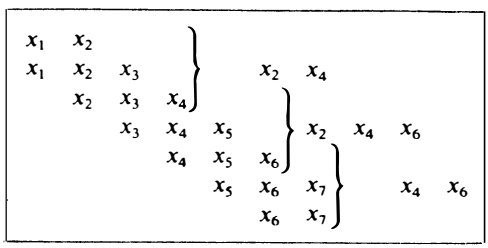


Рис. 2.3.17. Циклическая редукция

дана схематическая иллюстрация этого процесса для $n = 7$. Считая n нечетным числом, мы получаем в конце процесса модифицированную систему

$$\begin{aligned} a_2'x_2 + b_2'x_4 &= d_2', \\ c_4'x_2 + a_4'x_4 + b_4'x_6 &= d_4', \\ &\cdots\cdots\cdots \end{aligned} \tag{2.3.36}$$

содержащую только переменные x_2, x_4, ..., x_{n-1}. (Заметим, что мы уже проводили такой процесс для блочно трехдиагональной матрицы (2.3.3) при построении редуцированной системы (2.3.16) по методу Джонсона.)

Система (2.3.36) — это трехдиагональная система относительно переменных x_2, x_4, ..., и описанный только что процесс

можно повторить. В результате получится новая система, в которую входят лишь переменные x_4, x_8, ... Будем продолжать действовать указанным образом, пока дальнейшая редукция не станет невозможной. В частности, при $n = 2^q - 1$ редукция закончится единственным финальным уравнением. Решив его, начнем обратную подстановку. Она схематически иллюстрируется для $n = 7$ рисунком 2.3.18, который представляет собой продолжение рис. 2.3.17. Три уравнения на рис. 2.3.18 получены после первого шага редукции, показанного на рис. 2.3.17; они комбинируются, порождая финальное уравнение относительно

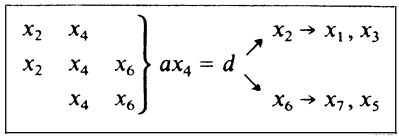


Рис. 2.3.18. Обратная подстановка

единственного неизвестного x_4. Решив его, мы можем затем решить относительно x_2 и x_6 первое и последнее уравнения редуцированной системы. После этого x_1 и x_7 можно определить из первого и последнего уравнений исходной системы. Наконец, x_3 и x_5 можно найти из третьего и пятого исходных уравнений.

Если $n \neq 2^q - 1$, то редукцию можно закончить системой с небольшим числом переменных. Решив ее, начнем обратную подстановку. Альтернативный вариант предусматривает добавление к системе некоторого количества фиктивных уравнений вида $x_i = 1$, чтобы общее число переменных стало равно $2^q - 1$ для некоторого числа q.

Описанный процесс известен под названием *циклической* или *нечетно-четной редукции*. Хотя в те времена, когда он был изобретен, параллельные вычисления вовсе не имелись в виду, этот процесс обладает немалым параллелизмом. Заметим прежде всего, что операции, ведущие к новым уравнениям для x_2, x_4, ..., x_{n-1}, независимы и могут выполняться одновременно. Предположим, что имеются $p = (n - 1)/2$ процессоров и применяется схема хранения, показанная на рис. 2.3.19. Тогда процессоры могут параллельно выполнять операции, порождающие первую редуцированную систему относительно неизвестных с четными номерами. В конце этого шага каждый процессор будет содержать коэффициенты в точности одного нового уравнения, и до начала следующего шага каким-то процессорам должны быть пересланы коэффициенты двух дополнительных уравнений. Один из естественных способов состоит в том, что

каждый «средний» процессор получает такую информацию по схеме

$$P_1 \to P_2 \leftarrow P_3 \to P_4 \leftarrow P_5.$$

Процессоры с нечетными номерами теперь прекращают работу, а процессоры с четными номерами производят следующий шаг исключения. Процесс продолжается, и после каждого шага примерно половина остающихся процессоров переходит в разряд пассивных; заключительное комбинирование трех уравнений, порождающее одно уравнение с единственным неизвестным, выполняется каким-то одним процессором. Отметим, что всего в редукции $q - 1 = \log[(n+1)/2]$ параллельных шагов. При

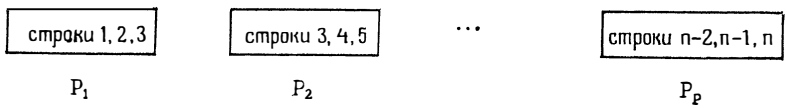


Рис. 2.3.19. Схема хранения для циклической редукции

обратной подстановке порядок действий по существу обращается. Финальное уравнение решается одним процессором, и с каждым новым шагом число активных процессоров возрастает.

В более реалистическом случае $p \ll n$ в каждый процессор до начала редукции будут загружены несколько строк и работа процессоров будет высокоэффективной, за исключением последних этапов, когда потребуются пересылки данных и, как и прежде, все больше процессоров будут становиться пассивными.

Упражнения к параграфу 2.3

1. Адаптировать к ленточным системам другие псевдокоды LU-разложения из § 2.1. Показать, что максимальная длина векторов для любой из этих адаптаций равна β.

2. Предположим, что векторному компьютеру требуется 200 нс для скалярных операций и $(1000 + 10\ n)$ нс для векторных операций с векторами длины n, исключая триады, подчиняющиеся формуле затрат времени $(1600 + 10\ n)$ нс. Подсчитать число операций векторной и скалярной арифметики для LU-разложения (рис. 2.3.1b) и столбцового алгоритма (рис. 2.3.4) в случае ленточной системы с полушириной ленты β, если матрица хранится по столбцам. Выяснить, для каких значений β выгодно пользоваться исключительно скалярной арифметикой.

3. Для векторного компьютера с характеристиками, указанными в упражнении 2, установить, какой из двух алгоритмов, представленных на рис. 2.3.1a и на рис. 2 3.1b, эффективней, считая, что в каждом случае используется оптимальная для данного алгоритма схема хранения. После этого добавить этап решения треугольных систем путем соответственно прямой и обратной подстановки. Использовать в каждом случае наиболее подходящий из двух

алгоритмов: скалярных произведений или столбцовый. Выяснить, какой из алгоритмов решения полной системы лучше в целом.

4. Пусть A — ленточная матрица порядка 1000 с полушириной ленты β. Для значений β, равных 20, 45 и 75, обсудить распределение нагрузки в LU-разложении со строчной циклической слоистой схемой хранения для 10, 50 и 100 процессоров.

5. Пусть A — симметричная положительно определенная матрица с полушириной ленты β. Адаптировать псевдокод метода Холесского (рис. 2.1.10) к ленточным системам. Каковы длины векторов в адаптированном варианте?

6. Для матрицы A из упражнения 5 рассмотреть параллельную реализацию метода Холесского для циклической слоистой схемы хранения (строчной или столбцовой), следуя при этом обсуждению LU-разложения в основном тексте параграфа.

7. Пусть A — матрица порядка 1000 с полушириной ленты $\beta = 100$. Обсудить требования к схеме хранения при решении системы $A\mathbf{x} = \mathbf{b}$ с использованием перестановок.

8. Дополнить рисунки 2.3.9 и 2.3.10, считая, что $p = 3$.

9. Выписать явно системы (2.3.7), (2.3.8) и (2.3.9) в покомпонентной форме для $p = 3$, $\beta = 2$ и $q = 5$.

10. Показать, что порядок редуцированной системы (2.3.9) равен $2\beta(p-1)$. Показать, кроме того, что матрица коэффициентов системы (2.3.9) является блочно пятидиагональной, а если рассматривать ее как ленточную, то имеет полуширину ленты 3β

11. Выписать явно матрицу системы (2.3.14) для $\beta = 2$ и $q = 3$. То же самое сделать для матрицы системы (2.3.16).

12. Выписать явно матрицу системы (2.3.22) для $p = 3$, $\beta = 2$, $q = 4$ и $s = 2$.

13. Переупорядочить систему в методе Джонсона, присваивая последние номера неизвестным, составляющим подвекторы $\mathbf{x}_{i2}$ в (2.3.12) и записывая уравнения в соответствующем порядке. Показать, что матрица коэффициентов переупорядоченной системы имеет форму (2.3.22). Показать далее, что редуцированная система (в случае $p = 3$ ее можно получить из (2.3.16)) соответствует системе $\hat{A}\mathbf{x}_S = \hat{\mathbf{b}}$ метода декомпозиции области. Вывести отсюда, повторяя рассуждения, проведенные в основном тексте для метода декомпозиции, что редуцированная система метода Джонсона симметрична и положительно определена, если такова исходная система

14. Для трехдиагональных систем ($\beta = 1$) сформулировать блочные методы и методы декомпозиции области (см рис. 2.3.11, 2.3.12, 2.3.15 и 2.3.16). В каждом случае выписать явно редуцированные системы и указать их порядки. Обсудить параллельные свойства каждого метода в зависимости от значений p и n.

Литература и дополнения к параграфу 2.3

1. Осознание того обстоятельства, что степень векторизации или параллелизма в LU-разложении и разложении Холесского зависит от ширины ленты матрицы, стимулировало — на раннем этапе развития теории параллельных методов — разработку методов решения систем с узкой лентой, особенно трехдиагональных систем. Стоун [Stone, 1973] предложил алгоритм для трех-

диагональных систем, основанный на идее рекурсивного удвоения (см. § 1.2). В статье [Lambiotte, Voigt, 1975] показано, что алгоритм Стоуна не согласован и указана согласованная версия; то же самое сделано и самим Стоуном [Stone, 1975], однако эти алгоритмы не получили широкого распространения.

2. Алгоритм циклической редукции (или его варианты) был, пожалуй, самым популярным методом решения трехдиагональных систем как для параллельных, так и для векторных компьютеров. Первоначально он был предложен Голубом и Хокни для специальных блочно трехдиагональных систем (см [Hockney, 1965]), но вскоре стало очевидно [Hockney, 1970], что метод можно применять и к трехдиагональным системам общего вида Хеллер [Heller, 1976] показал, что при определенных предположениях внедиагональные элементы в процессе циклической редукции убывают по абсолютной величине относительно диагональных элементов с квадратичной скоростью; это позволяет заканчивать процесс до того, как выполнены все $\log n$ шагов Несколькими авторами было отмечено (см., например, статью [Lambiotte, Voigt, 1975]), что циклическая редукция есть не что иное, как гауссово исключение, примененное к матрице PAP^{T}, где P — некоторая матрица перестановки. Таким образом, если A — симметричная положительно определенная матрица, то такова же матрица PAP^{T}, и процесс циклической редукции численно устойчив. Однако нужно все же со вниманием отнестись к обработке правой части, чтобы сохранить численную устойчивость, см по этому поводу [Golub, van Loan, 1983] Отметим еще, что при применении гауссова исключения к матрице PAP^{T} имеет место заполнение, которого не происходит в первоначальной трехдиагональной системе Отсюда вытекает, что число арифметических операций в циклической редукции примерно вдвое больше числа операций в гауссовом исключении для трехдиагональной системы. Более подробное обсуждение циклической редукции и ее вариантов можно найти в книге [Hockney, Jesshope, 1981]. В недавней работе [Johnson, 1987b] дан анализ циклической редукции и родственных методов в аспекте их реализации для ряда типов параллельной архитектуры Анализ обменов при реализации для гиперкубов см. в статье [Lakshmivarahan, Dhall, 1987].

3. Для трехдиагональных систем предложен и ряд других параллельных методов. Трауб ([Traub, 1974]; см. также [Heller et al., 1976]) построил итерационный метод, превратив три основных рекуррентных соотношения LU-факторизации в итерационные. В статье [Swarztrauber, 1979] рассматривается метод, опирающийся на формулы Крамера В работе [Sameh, Kuck, 1978] для QR-разложения матрицы используются вращения; этот процесс численно устойчив без перестановок. Данный метод, а также блочный метод Уонга [Wang, 1981] являются предшественниками блочных методов, обсуждавшихся в основном тексте. Среди других статей, где предлагаются или анализируются параллельные методы для трехдиагональных систем, можно назвать работы [Gao, 1986] (автор основывает свой алгоритм для параллельных машин на линейных разностных соотношениях); [Opsall, Parkinson, 1986] (рассматриваются почти трехдиагональные системы и ОКМД-машины, такие, как ICL DAP и MPP); [van der Vorst, 1986].

4. При $\beta = 1$ блочный алгоритм Лори — Самеха (см. рис 2 3 11) основан на методе Уонга [Wang, 1981] для трехдиагональных систем Аналогичное разбиение на блоки используется в QR-методе Самеха и Кука [Sameh, Kuck, 1978]. В статье [Lawrie, Sameh, 1984] идея блочного разбиения обобщается, что приводит к алгоритму LU-разложения ленточных систем, представленному на рис. 2.3.11. Похожий метод построен в работе [Meier, 1985] Поскольку решение редуцированной системы (2 3.9) потенциально является узким местом блочного алгоритма, в техническом отчете [Dongarra, Sameh, 1984] рассматривается вопрос о применении к этой системе итерационного метода. Блочный алгоритм на рис. 2.3.12 предложен Джонсоном [Johnson,

1985b], который проанализировал его основные свойства и дал оценки сложности его реализации для ряда топологий связей, включая гиперкубы, двоичные деревья и линейные массивы. В статье [Dongarra, Johnson, 1987] можно найти дальнейший анализ и экспериментальные результаты для блочных методов. Некоторый вариант метода Лори – Самеха использован в работе [Gannon, van Rosendale, 1984] в качестве примера при исследовании коммуникационной сложности.

5 Идея декомпозиции области имеет давнюю историю в инженерном мире, где она была высказана вне связи с параллельными методами и привела к технике расчетов, обычно называемой *методом подконструкций* Идея состояла в том, чтобы разбить большую задачу на меньшие части, которые можно обрабатывать порознь, после чего из решений этих подзадач восстановить решение задачи в целом (см статью [Noor et al, 1978]). В книге [George, Liu, 1981] идея декомпозиции области обсуждается под именем «стратегии параллельных сечений» с точки зрения использования разреженности в матрицах B_i (см. (2.3 22)). Вопрос о приложениях техники подконструкций к параллельным вычислениям рассматривается в работе [Adams, Voigt, 1984].

6 Процесс разбиения можно продолжить, выделяя в подзадачах их собственные подконструкции Так, если исходная система имеет полуширину ленты β, то это же верно для матриц A_i в (2.3 22), и технику подконструкций можно применить к каждой из них. Возможны разбиения и большей глубины. Эта идея была высказана Джорджем в начале 1970-х годов (см. статью [George, 1977] и книгу [George, Liu, 1981]) под названием стратегии *вложенных сечений.* Она особенно полезна в применении к задачам, ведущим свою родословную от дифференциальных уравнений с частными производными. Некоторые свойства техники вложенных сечений при ее реализации для параллельных и векторных компьютеров обсуждаются в работах [George et al., 1978; Gannon, 1980].

7. Среди других работ, касающихся решения ленточных систем в связи с параллельными и векторными компьютерами, можно назвать [Ashcraft, 1985a], [Cleary et al, 1986] (обсуждаются LU-разложение и разложение Холесского для параллельных систем с локальной и разделяемой памятью), [Gannon, 1986] (рассматривается гауссово исключение для параллельно-векторной машины), [Bjorstad, 1987] (описана программа, решающая большие конечно-элементные системы посредством повторного разбиения на подконструкции), [Schreiber, 1986] (предложен алгоритм QR-разложения, предназначенный для систолических массивов), [Kapur, Browne, 1984] (исследуется вопрос о решении блочно трехдиагональных систем применительно к массивам процессоров с изменяемой конфигурацией) и [Saad, Schultz, 1987].

8. В основном тексте отмечено, что перестановки в ходе гауссова исключения расширяют ленту матрицы. Это же верно в отношении метода Гаусса — Жордана даже при отсутствии перестановок. Более точно, после исключения наддиагональных элементов i-го столбца все наддиагональные элементы в столбце $i + \beta$ в общем случае становятся ненулевыми Это значительно увеличивает количество вычислений и делает алгоритм Гаусса — Жордана для ленточных систем еще менее привлекательным, чем для заполненных.

9 Методы из данного параграфа непригодны для разреженных матриц общего вида. В случае использования последовательных компьютеров для таких матриц имеются хорошо выверенные методы (см., например, книгу [George, Liu, 1981]). В техническом отчете [George, Heath et al., 1986] изучается реализация разложения Холесского для параллельных компьютеров в предположении, что необходимая предварительная работа (вычисление

перестановки и символьная факторизация) уже проделана. Описан алгоритм разложения, ориентированный на обработку столбцов; основными операциями в нем являются модификация j-го столбца с помощью k-го, если $l_{jk} \neq 0$ ($j > k$), и деление j-го столбца на число. См. также работы [George, Liu, Ng, 1987; George, Heath et al., 1987a, b]. В статье [Peters, 1984] обсуждаются параллельные алгоритмы, включающие в себя выбор главного элемента. В сообщении [Lewis, Simon, 1986] говорится о реализации гауссова исключения для векторных компьютеров, особенно машин серии CRAY X-MP. Основная операция здесь — триада с разреженными операндами, которую трудно векторизовать. Однако в машинах серии X-MP, в противоположность компьютеру CRAY-1, имеются аппаратно реализованные команды сборки/рассылки. Использование этих команд позволяет — при программировании на языке ассемблера — получить производительность, доходящую до 78 мегафлопов. Среди других работ, посвященных методам для разреженных матриц, можно назвать [Duff, 1984] (реализации для машин CRAY), [Duff, 1986] (предложен многофронтальный метод), [Dave, Duff, 1987] (сообщается о производительности фронтальной программы для разреженных матриц, достигнутой в экспериментах с машиной CRAY-2), [Liu, 1986, 1987] (разреженное разложение Холесского), [Greenbaum, 1986a] (рассматривается решение треугольных систем на машинах с разделяемой памятью), [Wing, Huang, 1977, 1980] и [Alaghband, Jordan, 1985].

Глава 3

Итерационные методы решения линейных уравнений

3.1. Метод Якоби

В этом параграфе мы рассмотрим один из простейших итерационных методов: метод Якоби. Хотя метод Якоби не является приемлемым для большинства задач, он представляет собой удобную отправную точку для обсуждения итерационных методов. Он будет также полезен в дальнейшем как вспомогательный метод.

Пусть A — невырожденная матрица $n \times n$ и нужно решить систему

$$A\mathbf{x} = \mathbf{b}, \tag{3.1.1}$$

где диагональные элементы a_{ii} матрицы A ненулевые. Тогда метод Якоби имеет вид

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(-\sum_{j \neq i} a_{ij} x_j^k + b_i\right), \quad i = 1, \ldots, n, \quad k = 0, 1, \ldots, \tag{3.1.2}$$

где верхние индексы указывают номер итерации и предполагается, как и для всех итерационных методов, что $x_1^0, \ldots, x_n^0$ представляют собой заданную аппроксимацию решения системы (3.1.1).

Для многих целей удобно переписать (3.1.2) в матричной форме. Пусть $D = \operatorname{diag}(a_{11}, \ldots, a_{nn})$ — диагональная матрица, образованная диагональными элементами A, и пусть $B = D - A$, т. е. $A = D - B$ представляет собой разложение матрицы на диагональную и внедиагональную части. Тогда мы можем записать (3.1.2) как

$$x^{k+1} = Hx^k + d, \quad k = 0, 1, \ldots, \quad H = D^{-1}B, \quad d = D^{-1}b. \tag{3.1.3}$$

Конечно, метод Якоби сходится не всегда. Приведем две стандартные теоремы сходимости, доказательства которых даны в приложении 2.

3.1.1. Теорема. *Если матрица A имеет строгое диагональное преобладание или является неприводимой с диагональным*

преобладанием, то итерации метода Якоби сходятся при любом начальном приближении $\mathbf{x}^0$.

3.1.2. Теорема. *Если матрица* $A = D - B$ *симметрична и положительно определена, то итерации метода Якоби сходятся при любом начальном приближении* $\mathbf{x}^0$ *тогда и только тогда, когда матрица* $D + B$ *положительно определена.*

Основная операция, фигурирующая в (3.1.3), — это матрично-векторное умножение, и для ее выполнения можно применить любой из подходов, описанных в параграфе 1.3. Там мы убедились в том, что эффективная параллельная или векторная реализация матрично-векторного умножения существенно зависит от структуры матрицы H. Итерационные методы, вообще говоря, используются только в том случае, когда матрица A, а значит, и H, является большой разреженной матрицей типа тех, которые возникают при дискретизации эллиптических краевых задач с помощью конечных разностей или конечных элементов. Опишем модельную задачу такого типа.

Уравнение Пуассона

Рассмотрим эллиптическое уравнение в частных производных, называемое *уравнением Пуассона*,

$$u_{xx} + u_{yy} = f, \tag{3.1.4}$$

где $(x, y) \in \Omega = [0, 1] \times [0, 1]$ и функция u задана на границе области Ω. Правая часть f представляет собой известную функцию от x и y, и если $f \equiv 0$, то (3.1.4) называют *уравнением Лапласа*. Введем дискретизацию единичного квадрата Ω с шагом по пространству h способом, показанным на рис. 3.1.1. Если обозначить через u_{ij} приближение к решению задачи (3.1.4) в точке (ih, jh), принадлежащей сетке, то, применив для аппроксимации производных в (3.1.4) обычные конечные разности второго порядка, мы получим систему уравнений

$$u_{i+1,\,j} + u_{i-1,\,j} + u_{i,\,j+1} + u_{i,\,j-1} - 4u_{ij} = h^2 f_{ij}, \tag{3.1.5}$$
$$i,\ j = 1,\ \ldots,\ N,$$

где $(N + 1)h = 1$. Согласно нашему предположению о том, что функция u задана на границе области, только N^2 переменных u_{ij}, $i, j = 1, \ldots, N$, во внутренних точках сетки являются неизвестными в уравнениях (3.1.5); таким образом, получается линейная система из $n = N^2$ уравнений, решение которой дает приближения к решению (3.1.4) в точках сетки.

Можно записать полученную систему в виде $A\mathbf{x} = \mathbf{b}$, взяв $x_k = u_{1k}$, $k = 1, \ldots, N$, в качестве неизвестных первой линии

узлов сетки, $x_k = u_{2,\,k-N}$, $k = N+1, \ldots, 2N$, — на второй линии узлов, и т. д. Таким образом,

$$x^{\mathrm{T}} = (u_{11}, \ldots, u_{1N}, u_{21}, \ldots, u_{2N}, \ldots, u_{N1}, \ldots, u_{NN}). \quad (3.1.6)$$

Соответствующую матрицу коэффициентов можно записать в блочном виде

$$A = \begin{bmatrix} T & -I & & & \\ -I & \ddots & \ddots & & \\ & \ddots & \ddots & -I & \\ & & -I & T & \end{bmatrix}, \quad T = \begin{bmatrix} 4 & -1 & & & \\ -1 & \ddots & \ddots & & \\ & \ddots & \ddots & -1 & \\ & & -1 & 4 & \end{bmatrix}, \quad (3.1.7)$$

где I — единичная матрица $N \times N$ и T также имеет размеры $N \times N$. Правая часть $\mathbf{b}$ полученной системы образована величинами $(-h^2 f_{ij})$, скорректированными в приграничных точках сетки при помощи известных значений u_{ij} на границе.

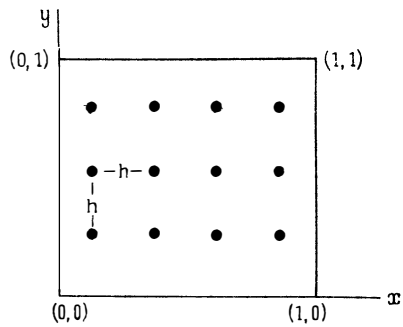


Рис. 3.1.1. Точки сетки на единичном квадрате

Заметим, что (3.1.7) является примером *диагонально разреженной матрицы* (см. § 1.3) и содержит только пять ненулевых диагоналей независимо от величины N. Однако мы вряд ли будем хранить такую матрицу пусть даже в диагональном формате, особенно в случае выполнения вычислений типа итераций Якоби. Здесь лучше применить метод Якоби к уравнениям (3.1.5) следующим образом:

$$u_{ij}^{k+1} = \frac{1}{4}\left(u_{i+1,\,j}^{k} + u_{i-1,\,j}^{k} + u_{i,\,j+1}^{k} + u_{i,\,j-1}^{k} - h^2 f_{ij}\right), \quad (3.1.8)$$

$$i, j = 1, \ldots, N; \quad k = 0, 1, \ldots.$$

Для уравнения Лапласа $f \equiv 0$ соотношения (3.1.8) показывают, что очередная итерация в точке сетки с индексом (i, j) есть не что иное, как среднее арифметическое четырех значений предыдущего приближения в соседних точках: «северной», «южной», «восточной» и «западной».

Распараллеливание метода Якоби

Из (3.1.8) ясно, что в принципе все значения u_{ij}^{k+1} можно вычислять параллельно; именно по этой причине метод Якоби иногда рассматривается в качестве прототипа параллельного метода. Однако при реализации этого метода на векторных или параллельных компьютерах следует обратить внимание на некоторые детали. Рассмотрим простейшие вопросы, связанные с параллельными вычислительными системами, сохраняя пока дискретное уравнение Пуассона (3.1.5) в качестве модельного.

Предположим сначала, что параллельная система образована сетью из $p = q^2$ процессоров P_{ij}, упорядоченных в виде

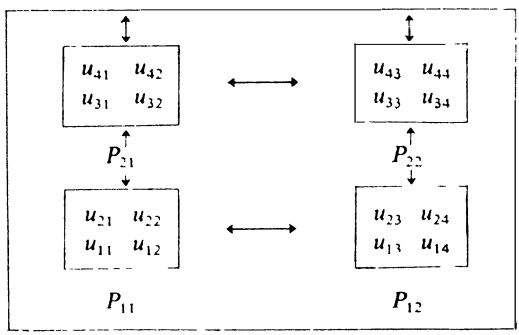


Рис. 3.1.2. Распределение точек сетки по процессорам

двумерной решетки, и каждый процессор связан со своим северным, южным, восточным и западным соседом, как это показано на рис. 3.1.2. Предположим, что $N^2 = mp$. Тогда естественно поручить каждому процессору обработку m неизвестных. Это можно сделать множеством способов. Один из простейших — наложить сеть процессоров на точки сетки, изображенные на рис. 3.1.1, так, как это показано на рис. 3.1.2 для случая, когда на каждый процессор приходится по четыре неизвестных. Мы предполагаем, что те процессоры, которые содержат приграничные внутренние точки сетки, содержат также и соответствующие заданные граничные значения.

На параллельных системах с локальной памятью по завершении каждой итерации новые значения, полученные в некоторых точках сетки, необходимо передать соседним процессорам. Обсудим этот вопрос более детально. В предельном случае описанного выше упорядочения, когда число процессоров составляет $p = N^2$, каждый процессор содержит в точности одно неизвестное. В этом случае вычисления в каждом процессоре,

занимающем положение внутри двумерной решетки, будут выполняться следующим образом:

$$\begin{aligned} &\text{вычисляем } u_{ij}^{k+1} \text{ в процессоре } P_{ij};\\ &\text{посылаем значение } u_{ij}^{k+1} \text{ процессорам}\\ &P_{i+1,j},\ P_{i-1,j},\ P_{i,j+1},\ P_{i,j-1}. \end{aligned} \qquad (3.1.9)$$

Таким образом, на каждом этапе вычислительный шаг, реализуемый по формулам (3.1.8) в случае уравнения Пуассона, сопровождается последующей передачей обновленных значений итерационного приближения тем процессорам, которым эти значения потребуются для выполнения следующей итерации. Хотя такое упорядочение демонстрирует идеальный параллелизм метода Якоби, требование наличия N^2 процессоров представляется, вообще говоря, нереалистическим, и, кроме того, значительная часть времени в этом случае тратится на выполнение операций обмена.

Более обычной является ситуация, когда $N^2 = mp$ и каждый процессор содержит m неизвестных. В случае $m = 4$ иллюстрацией может служить рис. 3.1.2, однако более реальны значения $N = 100$ и $p = 16$, откуда $m = 625$. Каждый процессор будет вычислять итерационные приближения для тех неизвестных, которые он содержит. Например, для ситуации, изображенной на рис. 3.1.2, вычисления, выполняемые процессором P_{22}, будут таковы:

$$\begin{aligned} &\text{вычислить } u_{33}^{k+1},\ u_{34}^{k+1};\ \text{передать их в } P_{12};\\ &\text{вычислить } u_{43}^{k+1};\ \text{передать } u_{33}^{k+1},\ u_{43}^{k+1} \text{ в } P_{21};\\ &\ldots \end{aligned} \qquad (3.1.10)$$

При этом аналогичные вычисления параллельно выполняются и в других процессорах.

Будем называть *внутренними граничными значениями* те значения u_{ij}, которые необходимы другим процессорам на следующей итерации и, таким образом, подлежат пересылке. Соответствующая иллюстрация для одного процессора дана на рис. 3.1.3. Если процессор содержит $m = q^2$ неизвестных, то для всех m необходимо вычислить новые значения, однако переслать нужно не более $4q$ значений. Таким образом, при фиксированном числе процессоров по мере возрастания размера задачи будет увеличиваться также и отношение времени вычислений ко времени обменов. В частности, количество внутренних точек сетки, для которых не требуется обмен данными, возрастает как квадрат N, в то время как число точек сетки,

для которых необходим обмен данными, возрастает по N лишь линейно. Таким образом, для достаточно больших задач время обмена данными становится все более незначительным по сравнению со временем вычислений.

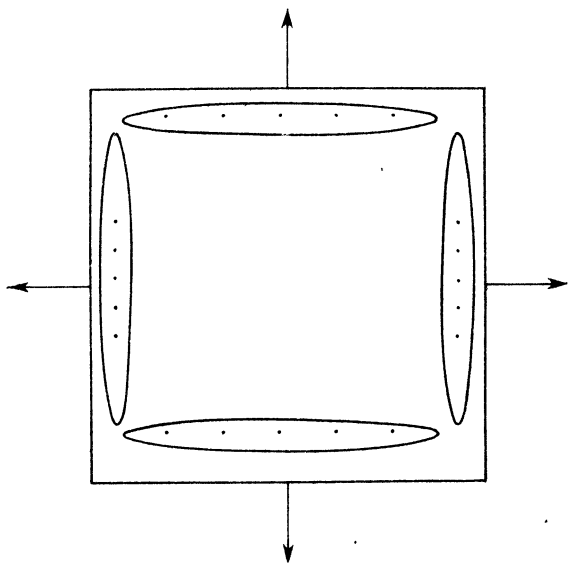


Рис. 3.1.3. Обмен данными, соответствующими внутренним границам

Синхронные и асинхронные методы

Для того чтобы обеспечить корректное выполнение итераций Якоби, в вычислительную схему типа (3.1.10) необходимо ввести механизм синхронизации. Например, вычисление u_{22}^{k+1} не может быть завершено до тех пор, пока u_{32}^{k} и u_{23}^{k} не будут получены от соседних процессоров. Как уже говорилось в § 1.2, время, необходимое для выполнения синхронизации, так же как и время, затрачиваемое процессором на ожидание того, чтобы все необходимые данные стали доступны для продолжения вычислений, вносит вклад в накладные вычислительные затраты.

Для некоторых итерационных методов, таких, как метод Якоби, привлекательной альтернативой служит допущение асинхронной работы процессоров (без какой бы то ни было синхронизации). В этом случае некоторые итерационные приближения могут быть вычислены некорректно. Например, в ситуации,

показанной на рис. 3.1.2, предположим, что в тот момент времени, когда должно быть вычислено значение u_{12}^{k+1}, значение u_{13}^{k} еще не получено от процессора P_{12}. Тогда для вычисления u_{12}^{k+1} будет использовано предыдущее значение u_{13}^{k-1}, и получающееся новое итерационное приближение уже не будет итерационным приближением Якоби, определяемым формулой (3.1.8). Однако это не обязательно нанесет ущерб итерационному процессу в целом, и в некоторых случаях с точки зрения общих затрат может оказаться выгодным применять итерационные методы в асинхронном варианте. Метод Якоби, реализованный указанным образом, будем называть *асинхронным* методом.

Проверка на сходимость

Применяя любой итерационный метод, необходимо проверять сходимость итерационных приближений. Если $\{\mathbf{x}^k\}$ — последовательность итерационных приближений, то часто используются два метода проверки:

$$\|\mathbf{x}^{k+1} - \mathbf{x}^k\| \leqslant \varepsilon \quad \text{или} \quad \|\mathbf{x}^{k+1} - \mathbf{x}^k\| \leqslant \varepsilon \|\mathbf{x}^k\|, \tag{3.1.11}$$

где $\|\cdot\|$ обозначает векторную норму. Довольно часто используют также невязку $\mathbf{r}^k = \mathbf{b} - A\mathbf{x}^k$ на k-й итерации и требуют, чтобы вместо (3.1.11) выполнялось условие

$$\|\mathbf{r}^k\| \leqslant \varepsilon \quad \text{или} \quad \|\mathbf{r}^k\| \leqslant \varepsilon \|\mathbf{r}^0\|. \tag{3.1.12}$$

Рассмотрим теперь реализацию проверки условий (3.1.11) для метода Якоби [1]) на примере ситуации, изображенной на рис. 3.1.2. Обычно используют нормы l_2 и l_∞, определяемые для вектора $\mathbf{x}$ соотношениями

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}, \quad \|\mathbf{x}\|_\infty = \max_{1 \leqslant i \leqslant n} |x_i|. \tag{3.1.13}$$

Использование нормы l_2 требует вычисления скалярного произведения с привлечением всех процессоров (см. § 1.3). С другой стороны, проверка первого условия (3.1.11) для нормы l_2 может быть реализована следующим образом. Каждый процессор на каждой итерации проверяет, удовлетворяют ли его неизвестные условиям

$$|u_{ij}^{k+1} - u_{ij}^{k}| \leqslant \varepsilon.$$

---

[1]) Все же следует предостеречь от применения (3.1.11) и рекомендовать (3.1.12), так как в общем случае из (3.1.11) вовсе не следует требуемая близость $\mathbf{x}^k$ к $\mathbf{x}$. — *Прим. перев.*

Если указанное условие выполнено, то устанавливается «флаг», показывающий, что приближения, содержащиеся в этом процессоре, удовлетворяют условию сходимости; кроме того, необходим механизм проверки флагов всех процессоров. Если все флаги сходимости установлены, то итерации сошлись, и, таким образом, их дальнейшее выполнение прекращается; в противном случае итерационный процесс продолжается. Таким образом, описанная процедура имеет много общего с механизмом синхронизации. Заметим, что в действительности здесь для проверки условия сходимости не вычисляется векторная норма как таковая.

Второй способ проверки сходимости (3.1.11) представляет собой проверку условия малости относительной погрешности и может оказаться полезным, когда примерная величина решения не известна априори (что позволило бы выбрать разумное значение ε). Однако такая проверка сходимости требует вычисления нормы с привлечением всех процессоров и последующей передачей вычисленного значения нормы каждому из процессоров, т. е. бóльших затрат при реализации.

Векторизация метода Якоби

Обратимся теперь к реализации метода Якоби на векторных компьютерах. Сначала мы опять ограничимся рассмотрением дискретного уравнения Пуассона и итерационного процесса

```
Для J от 1 до N
    Для I от 1 до N
        UN(I, J) = 0.25 * (U(I - 1, J) + U(I + 1, J)
                           + U(I, J + 1) + U(I, J - 1) - G(I, J))
```

Рис. 3.1.4. Фрагмент кода для метода Якоби

(3.1.8). В таком случае метод Якоби можно представить псевдокодом, приведенным на рис. 3.1.4. Здесь U и UN представляют собой двумерные массивы размера $(N+2)\times(N+2)$, которые содержат значения итерационных приближений на сетке, а также граничные значения, в то время как G является двумерным массивом размера $N\times N$, содержащим значения h^2f. На рис. 3.1.4 показана только одна итерация; на следующей итерации массивы U и UN меняются ролями. Заметим, что здесь допускаются нулевые индексы, чтобы учесть влияние граничных условий.

Рассмотрим сначала реализацию алгоритма, представленного на рис. 3.1.4, на векторном компьютере, использующем векторные регистры. Предположим для простоты, что значение

```
Загрузить U( , 0) (граничный столбец)
[Загрузить U( , 2) (второй столбец)
[Вычислить UN( , 1) = U( , 0) + U( , 2)
[Загрузить U( , 1) (первый столбец)
[Вычислить UN( , 1) = UN( , 1) + U( , 1)+1
 Вычислить UN( , 1) = UN( , 1) + U( , 1)-1
 Загрузить G( , 1)
 Вычислить UN( , 1) = UN( , 1) - G( , 1)
 Вычислить UN( , 1) = 0.25 * UN( , 1)
 Загрузить U( , 3)
[Записать в память UN ( , 1)
[. . .
  . . .
```

Рис. 3.1.5. Метод Якоби для компьютера, использующего векторные регистры

$N+2$ равно длине регистра; тогда мы можем реализовать вычисления так, как показано на рис. 3.1.5. Схема распреде-

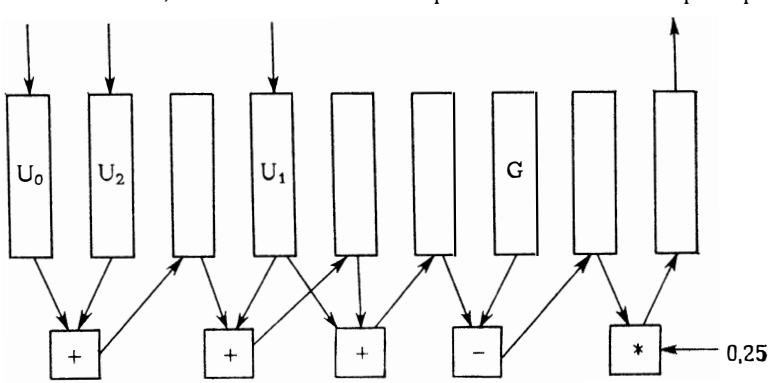


Рис. 3.1.6. Схема использования регистров для метода Якоби

ления регистров показана на рис. 3.1.6. Операторы, приведенные на рис. 3.1.5, вычисляют новые итерационные приближения для первого столбца точек сетки. Через $U(\ ,1)$ обозначен вектор длины N, содержащий текущие итерационные приближения,

отвечающие первому столбцу сетки, а $U(,1)_{\pm 1}$ обозначает тот же вектор, сдвинутый на одну позицию вверх или вниз. Квадратная скобка, объединяющая второй и третий операторы, указывает на то, что на некоторых векторных компьютерах эти две операции могут практически перекрываться с точностью до небольшой задержки. Аналогично, вторая скобка указывает на то, что операция загрузки следующего столбца данных может выполняться одновременно с предыдущим вычислением, а следующая скобка означает, что заключительная операция запоминания может выполняться одновременно с последующим вычислением. Приведенный фрагмент кода является, конечно, только частью всего цикла, осуществляющего индексацию столбцов U и UN. Дополнительное усложнение кода потребуется для того, чтобы эффективно использовать текущие данные, содержащиеся в регистрах, и, конечно, в том случае, когда значение $N+2$ превосходит длину векторных регистров; в последнем случае придется обрабатывать сегменты столбцов подходящей длины. Важным является то обстоятельство, что при соответствующем использовании векторных регистров вычисления хорошо векторизуются.

Использование длинных векторов

Рассмотрим теперь реализацию алгоритма на векторных компьютерах, извлекающих векторные операнды непосредственно из памяти и наиболее эффективных для векторов большой длины. Код, приведенный на рис. 3.1.4, легко реализуется за счет обработки одного столбца за другим, однако длина векторов в этом случае составляет лишь $O(N)$. Однако, оказывается, можно реорганизовать вычисления таким образом, что длина вектора окажется равной $O(N^2)$. Для этого удобно рассмотреть одномерный массив длины $(N+2)^2$, содержащий значения итерационных приближений и граничных значений, хранящихся по строкам, упорядоченных слева направо и снизу вверх. Это иллюстрирует рис. 3.1.7.

Будем обозначать через $U(I;K)$ вектор длины K, начинающийся с I-й позиции вектора U, и пусть $M1=(N+1)(N+2)-1$ и $M2=N(N+2)-2$. Тогда одна итерация метода Якоби может быть выполнена так, как показано на рис. 3.1.8, где T — вспомогательный вектор той же длины, что и U. Первый оператор вычисляет попарные диагональные суммы, как показано на рис. 3.1.9. Пунктирная линия на рис. 3.1.9, соединяющая $U(N+3)$ и $U(2N+4)$, указывает, что соответствующее сложение является избыточным вычислением; к обсуждению связанных с этим вопросов мы еще вернемся. Последней

суммой, вычисляемой первым оператором, будет $U((N+1)(N+2))+U((N+2)^2-1)$. Второй оператор позволяет получить сумму соседних значений для каждой точки сетки, как показано на рис. 3.1.10.

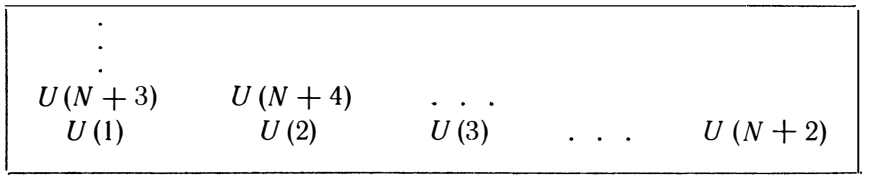


Рис. 3.1.7. Схема хранения в виде одномерного массива

```
T (2; M1) = U (2; M1) + U (N + 3; M1)
U (N + 4; M2) = T (2; M2) + T (N + 5; M2)
U (N + 4; M2) = 0.25 * (U (N + 4; M2) - G (N + 4; M2))
```

Рис. 3.1.8. Код для метода Якоби с использованием длинных векторов

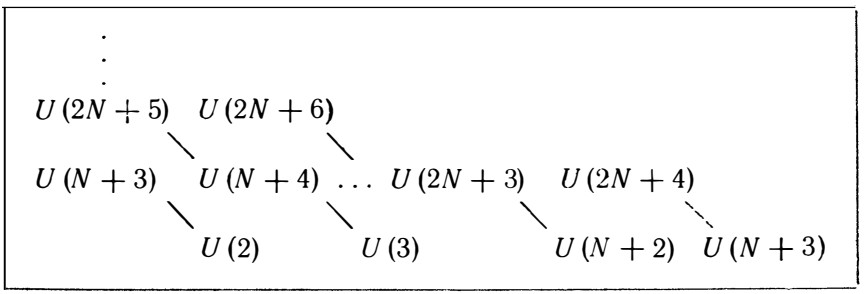


Рис. 3.1.9. Попарные суммы

$$U(2)+U(N+3)+U(N+5)+U(2N+6)$$
$$U(3)+U(N+4)+U(N+6)+U(2N+7)$$
$$\cdots$$

Рис. 3.1.10. Суммы, получаемые при выполнении второго оператора кода

Возможное затруднение, связанное со вторым оператором, представленным на рис. 3.1.8, заключается в том, что он заменяет граничные значения на некие фиктивные результаты. На некоторых векторных компьютерах (например, на CYBER 205) существует возможность подавления операции записи в память

посредством вектора, содержащего логические значения, и здесь эта возможность оказывается очень полезной; увеличения затрат времени при этом не происходит, однако требуется память для хранения $(N+2)^2$ битов. Последний оператор вычитает h^2f и делит на 4. Предполагается, что G также является массивом длины $(N+2)^2$, и здесь также используется подавление записи в память при обработке граничных позиций. Альтернативой подавлению записи в память может служить восстановление граничных значений после каждой итерации, однако это приведет к значительно бо́льшим затратам времени. Отметим также, что код, представленный на рис. 3.1.8, содержит лишь три векторных сложения в отличие от четырех, предусмотренных на рис. 3.1.4. Причиной является получаемый здесь эффект сложения сдваиванием; первый оператор на рис. 3.1.8 вычисляет суммы $U(2)+U(N+3)$ и $U(N+5)+U(2N+6)$, а затем второй оператор эти суммы складывает.

Подытоживая сказанное выше, заметим, что мы можем достичь максимальной длины векторов $O(N^2)$ и исключить одно векторное сложение, однако за это придется расплачиваться усложнением кода, дополнительной памятью для хранения граничных значений как в массиве T, так и в массиве G (а также памятью для хранения битового вектора, если есть возможность для его использования) и избыточными вычислениями, соответствующими граничным значениям. Отметим также, что нельзя получить код типа приведенного на рис. 3.1.8, использующий вектор, образованный только значениями во внутренних точках сетки (см. упражнение 3.1.11).

Векторизация с использованием матричного умножения

Подход, показанный на рис. 3.1.8, имеет весьма ограниченную область применимости. В следующем параграфе мы приведем пример более сложного дифференциального уравнения, для которого этот подход еще остается в силе, однако в общем случае далеко уклониться от уравнения Пуассона не удается. Даже для уравнения Пуассона, когда область не является квадратом или прямоугольником или принято неравномерное распределение точек сетки по направлению x или y, такой подход не работает (см. упражнение 3.1.12). Выгоды подхода, отвечающего рис. 3.1.8, заключаются в следующем: используются векторы большой длины и сокращается объем вычислений за счет уменьшения числа векторных сложений с четырех до трех. Гораздо более общий подход, к описанию которого мы переходим, позволяет также получить большую длину векторов, хотя и не дает экономии числа арифметических операций.

Для многих задач матрица перехода H метода Якоби (3.1.3) оказывается диагонально разреженной, однако ее диагонали не являются «постоянными», как в случае уравнения Пуассона (3.1.5) В качестве иллюстрации рассмотрим *обобщенное уравнение Пуассона*

$$(au_x)_x + (bu_y)_y = f, \tag{3.1.14}$$

где a и b — положительные функции от x и y. Для простоты опять предположим, что (3.1.14) рассматривается на области, представляющей собой единичный квадрат, и на границе области задано решение u. Стандартная конечноразностная дискретизация уравнения (3.1.14) имеет вид

$$a_{i-\frac{1}{2},\,j}u_{i-1,\,j} + a_{i+\frac{1}{2},\,j}u_{i+1,\,j} + b_{i,\,j-\frac{1}{2}}u_{i,\,j-1} + \\ + b_{i,\,j+\frac{1}{2}}u_{i,\,j+1} - c_{ij}u_{ij} = h^2 f_{ij}, \tag{3.1.15}$$

где

$$c_{ij} = a_{i-\frac{1}{2},\,j} + a_{i+\frac{1}{2},\,j} + b_{i,\,j\,\frac{1}{2}} + b_{i,\,j+\frac{1}{2}}. \tag{3.1.16}$$

В (3.1.15) коэффициенты a и b вычисляются в средних точках отрезков, соединяющих соседние точки сетки, например, $a_{i-\frac{1}{2},\,j} = a\left(ih - \frac{h}{2},\, jh\right)$, и аналогично для других коэффициентов. Как мы увидим ниже, такой способ обеспечивает симметрию матрицы коэффициентов. Заметим, что (3.1.15) сводится к (3.1.5) при $a = b = 1$.

Нетрудно убедиться (см. упражнение 3.1.13), что уравнения (3.1.15) можно записать в виде $A\mathbf{x} = \mathbf{b}$, где $\mathbf{x}$ — вектор неизвестных u_{ij}, определенный в (3.1.6), $\mathbf{b}$ — вектор, содержащий граничные значения и значения $-h^2 f_{ij}$, причем

$$A = \begin{bmatrix} T_1 & B_1 & & & \\ & \ddots & \ddots & & \\ B_1 & & \ddots & & \\ & \ddots & & \ddots & B_{N-1} \\ & & & B_{N-1} & T_N \end{bmatrix}, \tag{3.1.17}$$

где

$$T_j = \begin{bmatrix} c_{1j} & -a_{\frac{1}{2},\,j} & & & \\ -a_{\frac{1}{2},\,j} & c_{2j} & -a_{\frac{3}{2},\,j} & & \\ & \ddots & \ddots & \ddots & \\ & & -a_{N-\frac{1}{2},\,j} & c_{Nj} & \end{bmatrix},$$

$$B_j = \begin{bmatrix} -b_{1,\,j+\frac{1}{2}} & & \\ & \ddots & \\ & & -b_{N,\,j+\frac{1}{2}} \end{bmatrix}.$$

Матрица A в (3.1.17) имеет пять ненулевых диагоналей. Матрица перехода метода Якоби имеет ту же структуру, что схематически отражено в формуле

$$H = \begin{bmatrix} 0 & \diagdown & \diagdown & & \\ \diagdown & 0 & \diagdown & \diagdown & \\ & \diagdown & 0 & \diagdown & \ddots \\ & & \diagdown & 0 & \diagdown \\ & & & \diagdown & 0 \end{bmatrix} \qquad (3.1.18)$$

Мы предполагаем, что H хранится в диагональном формате и что операция умножения Hx^k в итерациях Якоби выполняется при помощи процедуры умножения по диагоналям, описанной в § 1.3. Таким образом, длина векторов при выполнении итераций Якоби составляет $O(N^2)$. Более точно, главная диагональ матрицы H имеет длину N^2, ближайшие к ней диагонали имеют длину N^2-1, а более удаленные диагонали — длину N^2-N. Таким образом, для хранения матрицы H требуется $2N^2-N-1$ слов, так как в силу симметрии достаточно хранить только верхнюю треугольную часть матрицы. При этом память в

$N-1$ слов затрачивается на хранение нулей, содержащихся в диагоналях, ближайших к главной, которые показаны в (3.1.18), однако это необходимо для достижения большой длины векторов.

Описанная реализация метода Якоби с использованием «длинных» векторных операций эффективна для тех задач с переменными коэффициентами, в которых матрицы, играющие роль H, в любом случае должны храниться. Однако для уравнения Пуассона явное хранение матрицы H излишне, в чем мы уже убедились. Более того, подход, продемонстрированный на рис. 3.1.8, гораздо более эффективен, если только его удается применить (см. упражнение 3.1.14).

На параллельных машинах метод Якоби для уравнений (3.1.15) реализуется по уравнениям Пуассона, обсуждавшимся ранее, но с тем важным отличием, что коэффициенты a, b и c должны теперь храниться и использоваться. Если снова точки сетки распределены по процессорам так, как показано на рис. 3.1.2, то на вычислительной системе с локальной памятью желательно хранить соответствующие значения a, b и c в памяти того процессора, которому они требуются для вычислений. Например, согласно (3.1.15), текущая итерация Якоби для u_{ij} задается формулой

$$u_{ij}^{k+1} = c_{ij}^{-1}\left(a_{i-\frac{1}{2},\,j}u_{i-1,\,j}^{k} + a_{i+\frac{1}{2},\,j}u_{i+1,\,j}^{k} + b_{i,\,j-\frac{1}{2}}u_{i,\,j-1}^{k} + b_{i,\,j+\frac{1}{2}}u_{i,\,j+1}^{k} - h^2 f_{ij}\right),$$

и желательно, чтобы все указанные коэффициенты, включая f_{ij}, хранились в том процессоре, который выполняет соответствующее вычисление. Заметим, что это потребует хранения некоторых коэффициентов более чем в одном процессоре. Например, в ситуации, продемонстрированной на рис. 3.1.2, как для вычисления u_{33}, так и для вычисления u_{23} необходим коэффициент $a_{\frac{5}{2},\,3}$, который нужно будет хранить в обоих процессорах P_{12} и P_{22}. Аналогично, итерирование u_{33} и u_{32} требует доступа к одному и тому же коэффициенту $b_{3,\,\frac{5}{2}}$.

Блочные методы

Пусть заданная матрица A разбита на подматрицы так, что

$$A = \begin{bmatrix} A_{11} & \dots & A_{1q} \\ \vdots & & \vdots \\ A_{q1} & \dots & A_{qq} \end{bmatrix}, \qquad (3.1.19)$$

причем предполагается, что каждая матрица A_{ii} невырождена. Тогда блочный метод Якоби решения системы $A\mathbf{x} = \mathbf{b}$, отвечающий введенному разбиению, имеет вид

$$A_{ii}\mathbf{x}_i^{k+1} = -\sum_{j \neq i} A_{ij}\mathbf{x}_j^k + \mathbf{b}_i, \quad i = 1, \ldots, q, \quad k = 0, 1, \ldots, \tag{3.1.20}$$

где разбиение векторов $\mathbf{x}$ и $\mathbf{b}$ согласовано с разбиением A. Таким образом, выполнение одной блочной итерации Якоби требует решения q систем вида (3.1.20) с матрицами коэффициентов A_{ii}. Заметим, что в специальном случае, когда каждая матрица A_{ij} имеет размер 1×1, соотношения (3.1.20) сводятся к методу Якоби, рассматривавшемуся выше. Известно, что в некоторых случаях для компьютеров с последовательной обработкой блочные методы оказываются более быстрыми, чем точечные.

В качестве примера блочного метода Якоби снова рассмотрим дискретное уравнение Пуассона (3.1.5) и соответствующее матричное представление (3.1.7). В этом случае $q = N$, $A_{ii} = T$, $i = 1, \ldots, N$, $A_{i,i+1} = A_{i+1,i} = -I$, а все остальные блоки A_{ij} нулевые. Таким образом, соотношения (3.1.20) принимают вид

$$T\mathbf{x}_i^{k+1} = \mathbf{x}_{i+1}^k + \mathbf{x}_{i-1}^k + \mathbf{b}_i, \quad i = 1, \ldots, N, \tag{3.1.21}$$

где векторы $\mathbf{x}_0^k$ и $\mathbf{x}_{N+1}^k$ содержат граничные значения вдоль нижней и верхней границы соответственно, а векторы $\mathbf{b}_i$ содержат

$\mathbf{d}_i^k = \mathbf{x}_{i+1}^k + \mathbf{x}_{i-1}^k + \mathbf{b}_i$,	$i = 1, \ldots, N$
Решить $L\mathbf{y}_i^k = \mathbf{d}_i^k$,	$i = 1, \ldots, N$
Решить $U\mathbf{x}_i^{k+1} = \mathbf{y}_i^k$,	$i = 1, \ldots, N$

Рис. 3.1.11. Полинейный метод Якоби

граничные значения на боковых сторонах области и значения $h^2 f$. Для сетки, изображенной на рис. 3.1.1, результатом применения (3.1.21) является одновременное обновление всех неизвестных каждой строки сеточных узлов с использованием приближений к неизвестным на смежных линиях сетки, полученных на предыдущей итерации. Поэтому метод (3.1.21) известен под названием *полинейного метода Якоби.*

Для реализации (3.1.21) мы раз и навсегда в самом начале выполним разложение Холесского или LU-разложение матрицы T, а затем используем полученные множители при решении систем (3.1.21). Таким образом, если $T = LU$, то вычисле-

ния (3.1.21) выполняются так, как показано на рис. 3.1.11. Системы для $\mathbf{y}_i^k$ независимы и поэтому могут решаться параллельно; то же справедливо и для систем для $\mathbf{x}_i^{k+1}$, если векторы $\mathbf{y}_i^k$ уже известны. Таким образом, если у нас есть, например, $p = N$ процессоров, то можно поручить решение каждой пары систем одному из процессоров, как показано на рис. 3.1.12. На этом же рисунке указано, что после каждой итерации новые итерационные приближения $\mathbf{x}_i^{k+1}$ должны быть переданы каждому из соседних процессоров P_{i-1} и P_{i+1}. Заметим, что рас-

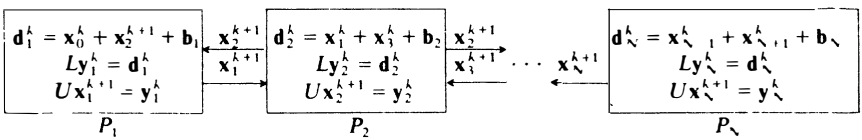


Рис. 3.1.12. Параллельный полинейный метод Якоби

смотренное упорядочение представляется идеальным для линейно связанной сети процессоров.

Блочную матрицу более общего вида (3.1.19) можно обрабатывать сходным образом. Так как предполагается, что все матрицы A_{ii} невырожденны, то для каждой из них существует

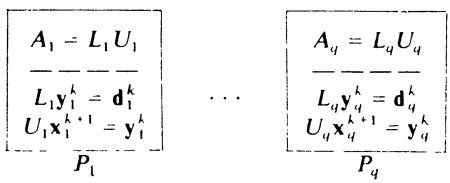


Рис. 3.1.13. Параллельный блочный метод Якоби

разложение L_iU_i, где (если в процессе разложения потребуются перестановки) матрица L_i уже не обязана быть нижнетреугольной. Соответствующие вычисления можно выполнить параллельно в самом начале, как показано на рис. 3.1.13, где пунктирная линия отделяет эти разложения, выполняемые один раз, от последующего итерационного процесса, и где предполагается, что число процессоров равно q. Как следует из (3.1.20), вектор d_i^k на рис. 3.1.13 равен

$$\mathbf{d}_i^k = -\sum_{j \neq i} A_{ij}\mathbf{x}_j^k + \mathbf{b}_i. \tag{3.1.22}$$

Таким образом, в общем случае каждый вектор $\mathbf{x}_i^{k+1}$ должен быть передан всем остальным процессорам перед началом

следующей итерации. Однако итерационный метод вряд ли будет использоваться для матрицы, имеющей заполненную блочную структуру. Скорее матрица A возникнет в результате расщепления области, на которой определено некоторое дифференциальное уравнение. Два таких расщепления показаны на рис. 3.1.14.

Предполагается, что подобласти содержат часть точек сетки, образующих внутренние границы между подобластями, причем

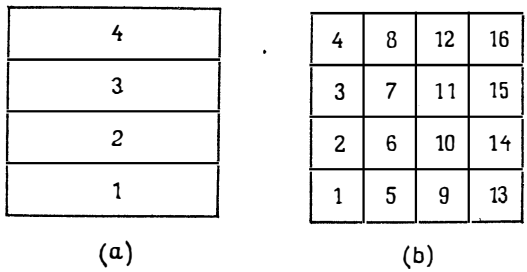


Рис. 3.1.14. Декомпозиция области: (a) на полосы; (b) на квадратные подобласти

каждая внутренняя точка сетки связана с одной и только одной подобластью. Например, мы можем предположить, что на рис. 3.1.14a каждая подобласть содержит точки своей верхней внутренней границы. Предельным случаем, отвечающим рис. 3.1.14a, является ситуация, когда каждая полоса содержит только одну линию точек сетки; при этом мы опять получаем полинейный метод Якоби. В общем случае, если предположить, что любое неизвестное, расположенное на i-й полосе, связано только с неизвестными на полосах с номерами $i-1$ и $i+1$, то A будет иметь блочно-трехдиагональную форму:

$$A = \begin{bmatrix} A_{11} & A_{12} & & & \\ & \ddots & \ddots & & \\ A_{21} & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & A_{q-1,q} \\ & & \ddots & \ddots & \\ & & & A_{q,q-1} & A_{qq} \end{bmatrix}, \tag{3.1.23}$$

В этом случае значение x_i^{k+1} нужно передавать только процессорам P_{i-1} и P_{i+1}, как показано на рис. 3.1.12. Конечно, мы предполагаем, что вычислительная загрузка процессоров максимально сбалансирована. Это так, если все матрицы A_{ii} имеют

одинаковый размер и структуру (и все внедиагональные блоки таковы, что формирование векторов d_i требует примерно одного и того же времени). В противном случае нужно рассматривать какое-то другое распределение данных между процессорами. И, естественно, если у нас менее q процессоров, каждому процессору нужно будет поручить обработку более чем одной подсистемы. На рис. 3.1.14b неизвестные, расположенные во внутренних подобластях, могут быть связаны с неизвестными в любой из окружающих подобластей. В этом случае блочная форма матрицы уже не будет трехдиагональной, но и не будет заполненной, как в (3.1.19). Блочные методы, так же как и рассматривавшиеся ранее поточечные методы, могут выполняться асинхронно, но теперь уже возможна ситуация, когда целые блоки неизвестных не успевают обновиться ко времени выполнения следующей итерации; это зависит от того, как осуществляется передача данных.

Векторизация блочных методов

Теперь мы рассмотрим реализацию блочного метода Якоби (3.1.20) на векторных компьютерах и начнем со специального случая (3.1.21). Предположим, что матрица T факторизована в виде произведения LU, где L — нижняя двухдиагональная матрица с единичной диагональю. Тогда прямую и обратную подстановки (см. рис. 3.1.11) можно выполнить при помощи *сквозной векторизации* набора систем, как показано на рис. 3.1.15. Псевдокод этой операции представлен на рис. 3.1.16.

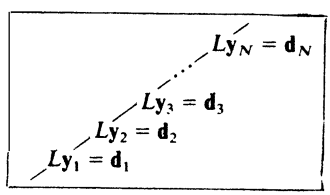


Рис. 3.1.15. Сквозная векторизация множества систем

Предполагается, что L1 — одномерный массив, содержащий элементы матрицы L, лежащие на поддиагонали, а y — двумерный массив, такой, что $y(\ , J)$ представляет собой вектор длины N, образованный J-ми компонентами всех векторов решений. Аналогично, $d(\ , J)$ является вектором длины N, образованным J-ми компонентами всех правых частей. Таким образом, длина вектора в алгоритме на рис. 3.1.16 всюду равна N. Обратную подстановку $U\mathbf{x} = \mathbf{y}$ можно выполнить аналогичным образом (см. упражнение 3.1.15).

Кроме того, на каждой итерации необходимо будет определить новые правые части в соответствии с (3.1.21). Их также можно вычислить с использованием сквозной векторизации по

векторам $\mathbf{x}_i$; действительно, требуемая схема хранения $\mathbf{x}_i$ как раз получится при выполнении обратной подстановки, так как код, построенный по аналогии с рис. 3.1.15, позволит получить векторы $x(\ ,J)$, образованные J-ми компонентами. Таким образом, очередной набор правых частей можно вычислить следующим образом:

Для $J = 1, \ldots, N$

$$d(\ ,J) = x(\ ,J)_{-1} + x(\ ,J)_{+1} + b(\ ,J).$$

Здесь $x(\ ,J)_{\pm 1}$ обозначает вектор $x(\ ,J)$ сдвинутый вверх или вниз на одну позицию.

```
y( ,1) = d( ,1)
Для J от 2 до N
    y( ,J) = — L1(J) * y( ,J — 1) + d( ,J)
```

Рис. 3.1.16. Прямая векторная подстановка для множества систем

Рассмотрим теперь матрицу более общего вида (3.1.23) (или (3.1.19)). Если все диагональные блоки имеют одну и ту же размерность и совпадающую структуру (например, если все они имеют одинаковую ширину ленты), то можно провести сквозную векторизацию LU-разложений $A_{ii} = L_i U_i$ по всем системам. Для каждого фиксированного значения j и k предположим, что (j, k)-е компоненты всех матриц A_{ii} хранятся в виде вектора. Указанные факторизации могут быть вычислены раз и навсегда перед началом выполнения итераций Якоби с сохранением полученных множителей. Заметим, что в случае, когда все матрицы A_{ii} идентичны (как в (3.1.21), где $A_{ii} = T$, $i = 1, \ldots, N$), подобным способом векторизации LU-разложений ничего достичь не удается; здесь лучше вычислить разложение с использованием техники, обсуждавшейся в § 2.3. Эти вычисления можно провести в скалярной арифметике, так как одно рассматриваемое разложение обычно составляет лишь малую часть общего объема вычислений, требующихся для выполнения итераций Якоби.

Располагая вычисленными множителями L_i и U_i, можно выполнить прямые и обратные подстановки в итерациях Якоби способом, аналогичным показанному на рис. 3.1.16, после того как вычислены новые правые части. Эти правые части задаются формулами (3.1.22), и для их вычисления можно применить сквозную векторизацию по матрицам A_{ij} при условии, что (если рассматривается случай (3.1.23)) все матрицы $A_{i,\,i+1}$, $i = 1, \ldots$

..., $q-1$, имеют одинаковый размер и совпадающую структуру и аналогичное условие справедливо для $A_{i-1,\,i}$, $i=2,\ldots,q$. В таком случае длина векторов на всех этапах вычисления (разложения, формирования правых частей, прямой и обратной подстановок) будет равна q, и от конкретного значения q будет зависеть, подходит ли описанный подход для того или иного векторного компьютера.

Методы переменных направлений

Закончим этот параграф кратким обсуждением неявных методов переменных направлений ADI[1]). Возможно, центральным из этих методов является метод Писмена — Рэкфорда, который мы опишем сначала для случая дискретного уравнения Пуассона (3.1.5) на квадрате. Итерация метода Писмена — Рэкфорда с номером $k+1$ состоит из двух шагов.

Шаг 1. Для некоторого параметра α_k решаем N линейных систем для $j=1,\ldots,N$:

$$(2+\alpha_k)\,u_{i,\,j}^{k+1/2}-u_{i-1,\,j}^{k+1/2}-u_{i+1,\,j}^{k+1/2}=$$
$$=(-2+\alpha_k)\,u_{i,\,j}^{k}+u_{i,\,j+1}^{k}+u_{i,\,j-1}^{k}+h^2f_{i,\,j},\quad i=1,\ldots,N. \tag{3.1.24a}$$

Шаг 2. Решаем N линейных систем для $i=1,\ldots,N$:

$$(2+\alpha_k)\,u_{i,\,j}^{k+1}-u_{i,\,j-1}^{k+1}-u_{i,\,j+1}^{k+1}=$$
$$=(-2+\alpha_k)\,u_{i,\,j}^{k+1/2}+u_{i+1,\,j}^{k+1/2}+u_{i-1,\,j}^{k+1/2}+h^2f_{i,\,j},\quad j=1,\ldots,N. \tag{3.1.24b}$$

На первом шаге (3.1.24a) получаются промежуточные значения $u^{k+1/2}$, которые затем используются на втором шаге (3.1.24b) для вычисления окончательных значений итерационных приближений, соответствующих $(k+1)$-й итерации.

Системы в (3.1.24a) имеют размер N и являются трехдиагональными. Они оказываются такими же, как в полинейном методе Якоби. Отличие сводится к параметру α_k и расщеплению u_{ij} на два слагаемых, одно из которых берется с предыдущей итерации и переносится в правую часть. Во всяком случае, как и для полинейного метода Якоби, (3.1.24a) сводится к отысканию новых значений u на каждой линии, в то время как значения u на смежных линиях сохраняются прежними. В (3.1.24b) направления линий изменяются на вертикальные, т. е. осуществляется вычисление новых значений u вдоль каждой

[1]) В оригинале alternating direction implicit method. — *Прим. перев.*

вертикальной линии точек сетки с использованием значений на смежных вертикальных линиях, только что полученных при выполнении шага 1; отсюда и происходит название «метод переменных направлений». Используемые направления показаны на рис. 3.1.17.

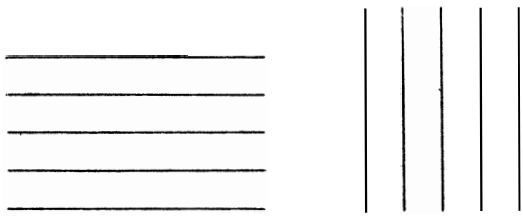


Рис. 3.1.17. Переменные направления: (a) горизонтальные направления; (b) вертикальные направления

Переформулируем теперь (3.1.24) в матричных обозначениях. При $n = N^2$ пусть H и V представляют собой матрицы $n \times n$

$$H = \begin{bmatrix} C & & & \\ & C & & \\ & & \ddots & \\ & & & C \end{bmatrix}, \quad V = \begin{bmatrix} 2I & -I & & \\ -I & \ddots & \ddots & \\ & \ddots & \ddots & I \\ & & -I & 2I \end{bmatrix}, \tag{3.1.25}$$

где все единичные матрицы имеют размеры $N \times N$, а C является матрицей $N \times N$ вида

$$C = \begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix}. \tag{3.1.26}$$

Тогда

$$A = H + V, \tag{3.1.27}$$

где A — матрица (3.1.7) коэффициентов системы, отвечающей дискретному уравнению Пуассона. Нетрудно убедиться (см. упражнение 3.1.22), что формулы (3.1.24) эквивалентны соотношениям

$$(\alpha_k I + H)\,\mathbf{x}^{k+\frac{1}{2}} = (\alpha_k I - V)\,\mathbf{x}^k + \mathbf{b}, \tag{3.1.28а}$$

$$(\alpha_k I + V)\,\mathbf{x}^{k+1} = (\alpha_k I - H)\,\mathbf{x}^{k+\frac{1}{2}} + \mathbf{b}, \tag{3.1.28б}$$

где $\mathbf{x}$ — вектор неизвестных на внутренних точках сетки, определенный в (3.1.6), а вектор $\mathbf{b}$ содержит граничные значения и значения f.

Если H и V заданы формулами (3.1.25), то из (3.1.28а) ясно, что вектор $x^{k+\frac{1}{2}}$ получается при помощи решения N трехдиагональных систем с матрицами коэффициентов вида $\alpha_k I + C$. Системы в (3.1.28b) уже не являются трехдиагональными, однако в этом случае существует такая матрица перестановки P, что $PVP^{\mathrm{T}} = H$ (см. упражнение 3.1.23). Таким образом, системы (3.1.28b) перестановочно подобны трехдиагональным системам и могут решаться аналогично.

Поскольку матрица C в (3.1.26) положительно определена (см. упражнение 3.1.19), тем же свойством обладает и матрица H. Таким образом, матрица V также положительно определена, поскольку является симметричной перестановкой матрицы H. Следовательно, для $\alpha_k \geqslant 0$ матрицы коэффициенты систем в (3.1.28) положительно определены и поэтому невырожденны. Указанное свойство положительной определенности является ключевым для сходимости итераций. Сформулируем следующую теорему, доказательство которой дано в приложении 2.

3.1.3. Теорема о сходимости ADI. *Пусть A, H и V — симметричные положительно определенные матрицы $n \times n$, удовлетворяющие соотношению* (3.1.27), *и пусть $\alpha_k = \alpha > 0$, $k = 0, 1, \ldots$. Тогда итерации* (3.1.28) *корректно определены и сходятся к единственному решению системы $A\mathbf{x} = \mathbf{b}$ при любом начальном приближении $\mathbf{x}^0$.*

Теорема 3.1.3 справедлива для любой положительно определенной матрицы A, однако матрицы H и V должны быть подобраны так, чтобы системы в (3.1.28) «легко» решались. Условие $\alpha > 0$ является существенным: сходимость итераций отсутствует, если все α_k в (3.1.28) равны нулю (см. упражнение 3.1.24). Теорему 3.1.3 можно распространить на случай, когда α_k образуют последовательность различных положительных чисел. Роль такого допущения заключается в том, что при

специальном выборе значений α_k можно добиться очень быстрой сходимости итераций. Однако в этом случае нужно наложить довольно сильное ограничение $HV = VH$. Это условие коммутативности выполняется (см. упражнение 3.1.25) для матриц H и V в (3.1.25), однако в общем случае оно будет справедливо только для некоторых эллиптических краевых задач при подходящих способах дискретизации.

Параллельную и векторную реализации (3.1.24) можно осуществить следующим образом. Поскольку N трехдиагональных систем в (3.1.24a) независимы, их можно решать параллельно или при помощи сквозной векторизации по аналогии с полинейным методом Якоби. То же самое в принципе можно было бы сделать и с системами (3.1.24b). Однако если способ хранения итерационного приближения был корректным при решении систем (3.1.24a), то он уже не является таковым для решения систем (3.1.24b); по существу, теперь требуется предварительное переупорядочение данных типа транспонирования матрицы. Однако принятый способ хранения данных остается корректным для решения трехдиагональных систем (3.1.24b) методом циклической редукции, описанным в § 2.3.

Упражнения к параграфу 3.1

1. Проверить, что (3.1.2) можно переписать в матричной форме (3.1.3).

2. Пусть

$$A = \begin{bmatrix} 1 & \alpha & \alpha \\ \alpha & 1 & \alpha \\ \alpha & \alpha & 1 \end{bmatrix}.$$

Показать, что матрица A положительно определена при $-1 < 2\alpha < 2$, однако условия теоремы (3.1.2) о сходимости итераций Якоби выполняются только при $-1 < 2\alpha < 1$.

3. Выписать полностью систему (3.1.7) для $N = 2$, 4 и 6.

4. Предположим, что есть параллельная система с локальной памятью, содержащая p процессоров, для которой выполнение каждой арифметической операции требует α единиц времени, а время передачи одного слова данных между любыми двумя процессорами составляет β единиц времени. Пусть $N^2 = mp$, так что каждый процессор обрабатывает m неизвестных. Определить время выполнения одной итерации Якоби для дискретного уравнения Пуассона.

5. Полностью описать вычисления, соответствующие рис. 3.1.4, 3.1.5 и 3.1.6 для $N = 4$

6. Полностью описать вычисления, соответствующие рис. 3.1.8 для $N = 4$.

7. Предположим, что векторному компьютеру требуется $(1000 + 10n)$ нс для выполнения арифметической операции с векторами длины n и 200 нс

для выполнения скалярной операции. Оценить время выполнения одной итерации Якоби при использовании (a) алгоритма, показанного на рис 3.1.4, и (b) алгоритма на рис. 3.1.8. Сравнить полученные оценки для $N = 50$, 100, 200.

8 Для векторного компьютера, описанного в упражнении 7, оценить долю времени, расходуемого на избыточные вычисления в алгоритме на рис. 3.1.8 для $N = 50$, 100, 200

9. Для параллельной системы, описанной в упражнении 4, при $p = N$ оценить время выполнения одной итерации полинейного метода Якоби, определяемого формулой (3.1.21) и рис. 3.1.11, 3.1.12.

10. Для векторного компьютера из упражнения 7 оценить время выполнения одной итерации полинейного метода Якоби (3 1.21) при использовании векторизации, показанной на рис. 3.1.14.

11. Показать, что код типа показанного на рис 3 1.8 не может дать правильного результата для уравнения Пуассона (и даже для уравнения Лапласа), если массив U содержит только внутренние точки

12. Если уравнение Пуассона (3.1.4) дискретизовано при помощи конечных разностей с использованием шага сетки Δx по направлению x и шага Δy по направлению y, то уравнения (3.1 5) заменяются на

$$(\Delta x)^{-2}(u_{i+1,\,j} - 2u_{ij} + u_{i-1,\,j}) + (\Delta y)^{-2}(u_{i,\,j+1} - 2u_{ij} + u_{i,\,j-1}) = f_{ij}$$

или

$$u_{i+1,\,j} + u_{i-1,\,j} + c\,(u_{i,\,j+1} + u_{i,\,j-1}) - 2\,(1 + c)\,u_{ij} = (\Delta x)^2 f_{ij},$$

где $c = (\Delta x)^2/(\Delta y)^2$. Итерации Якоби принимают тогда вид

$$u_{ij}^{k+1} = (2 + 2c)^{-1}\left(u_{i+1,\,j}^{k} + u_{i-1,\,j}^{k} + c\left(u_{i,\,j+1}^{k} + u_{i,\,j-1}^{k}\right) - (\Delta x)^2 f_{ij}\right).$$

Показать, что код, приведенный на рис. 3.1 8, нельзя модифицировать таким образом, чтобы с его помощью можно было выполнять итерации Якоби для $\Delta x \neq \Delta y$.

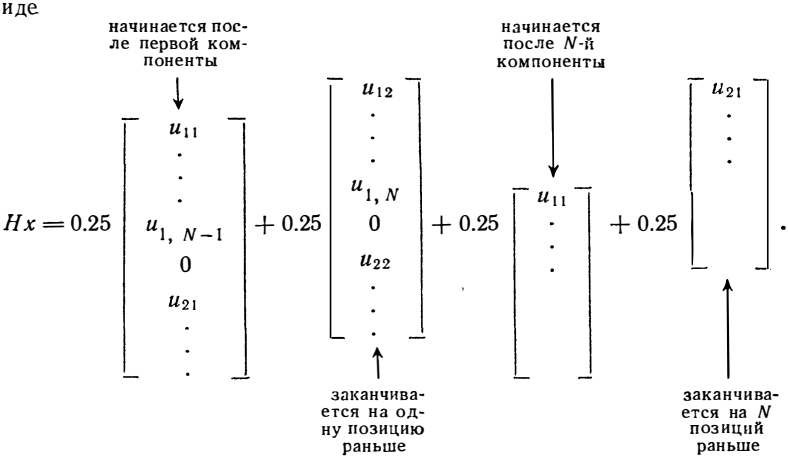
13. Проверить, что разностные уравнения (3.1.15) можно записать в виде $A\mathbf{x} = \mathbf{b}$, где матрица A определена в (3.1.17).

14. Показать, что для дискретного уравнения Пуассона на квадрате (3.1.5) матрица перехода метода Якоби принимает вид

$$H = \frac{1}{4}\begin{bmatrix} T_0 & I & & & \\ I & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & I \\ & & & I & T_0 \end{bmatrix}, \qquad T_0 = \begin{bmatrix} 0 & 1 & & & \\ 1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 0 \end{bmatrix},$$

где все подматрицы имеют размеры $N \times N$. Показать, что матрично-векторное умножение $H\mathbf{x}$, где вектор $\mathbf{x}$ определен в (3.1.6), можно представить

в виде



Рассмотреть реализацию этого вычисления с использованием триад и подавления записи в память. Предлагая, что затраты времени при обработке векторов длины m составляют $(1000 + 10m)$ нс для сложения и умножения в $(1700 + 10m)$ нс кля триады, показать, что время выполнения одной итерации Якоби составит примерно $(50N^2 - 40N + 7000)$ нс Показать, что время выполнения одной итерации Якоби, соответствующей рис. 3 1.8, составит около $(40N^2 \times 100N + 4000)$ нс, а для уравнения Пуассона при использовании команды усреднения (для компьютера CYBER 205) указанное время можно уменьшить до $(20N^2 + 50N + 2000)$ нс.

15. Написать псевдокод, аналогичный представленному на рис. 3.1.16, для выполнения обратной подстановки $U\mathbf{x} = \mathbf{y}$

16. Рассмотрим уравнение Пуассона $u_{xx} + u_{yy} = 4$ на единичном квадрате с условием $u(x, y) = x^2 + y^2$ на границе квадрата. Показать, что его точным решением является функция $u(x, y) = x^2 + y_2$. Показать, что точным решением дискретных уравнений (3.1.5) является $u_{ij} = x_i^2 + y_j^2$ и, таким образом, погрешность дискретизации здесь отсутствует.

17. Для дискретной задачи, описанной в упражнении 16, реализовать итерации Якоби в соответствии с рис. 3.1.6, ориентируясь на обработку столбцов, как показано на рис. 3.1.5; реализовать итерации Якоби с использованием векторов полной длины, см. рис. 3.1.8. Сравнить время выполнения одной итерации и затраты времени для этих методов.

18. Для задачи из упражнения 17 сравнить три способа проверки сходимости итераций: $\|\mathbf{x}^{k+1} - \mathbf{x}^k\|_2 \leqslant \varepsilon$, $\|\mathbf{x}^{k+1} - \mathbf{x}^k\|_\infty \leqslant \varepsilon$, $\|\mathbf{b} - A\mathbf{x}^{k+1}\|_2 \leqslant \varepsilon$. Какой из этих способов требует наибольших затрат времени при реализации на вашей машине?

19. Показать, что собственные значения матрицы $N \times N$

$$S = \begin{bmatrix} 2 & -1 & & & \\ -1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}$$

равны $2 - 2\cos\dfrac{k\pi}{N+1}$, $k = 1, \ldots, N$, а отвечающие им собственные векторы равны

$$\left(\sin\frac{k\pi}{N+1}, \ldots, \sin\frac{Nk\pi}{N+1}\right)^{\mathrm{T}}.$$

Указание: использовать тригонометрическое тождество $\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$ для проверки того, что $Sx = \lambda x$.

20. *Кронекеровским произведением* двух матриц B и C размерности $n \times n$ называют матрицу $n^2 \times n^2$ вида

$$B \otimes C = \begin{bmatrix} b_{11}C & \ldots & b_{1n}C \\ \vdots & & \vdots \\ b_{n1}C & \ldots & b_{nn}C \end{bmatrix}.$$

Кронекеровской суммой матриц B и C называют матрицу $A = (I \otimes B) + (C \otimes I)$, где I — единичная матрица $n \times n$. Известно (см., например, [Ortega, 1987a, теорема 6.3.2]), что собственные значения кронекеровской суммы A равны $\mu_i + \nu_j$, $i, j = 1, \ldots, n$, где $\mu_1, \ldots, \mu_n$ и $\nu_1, \ldots, \nu_n$ — собственные значения матриц B и C соответственно. Показать, что матрица A в (3.1.7) равна кронекеровской сумме $(I \otimes S) + (S \otimes I)$, где S — матрица, определенная в упражнении 19. Затем с помощью упражнения 19 показать, что собственные значения матрицы A равны

$$4 - 2\left(\cos\frac{k\pi}{N+1} + \cos\frac{j\pi}{N+1}\right), \qquad j, k = 1, \ldots, N.$$

21. Используя результаты и/или методы упражнения 20, показать, что собственные значения матрицы перехода метода Якоби из упражнения 14 равны

$$\frac{1}{2}\left(\cos\frac{k\pi}{N+1} + \cos\frac{j\pi}{N+1}\right), \qquad j, k = 1, \ldots, N.$$

22. Показать, что формулы (3.1.24) и (3.1.28) эквивалентны, если матрицы H и V имеют вид (3.1.25).

23. Найти такую матрицу перестановки P, что $PVP^{\mathrm{T}} = H$, где H и V имеют вид (3.1.25). *Указание*: рассмотреть упорядочение точек сетки снизу вверх и слева направо.

24. Пусть $A = H + V$, где A, H и V симметричны и положительно определены. Показать, что итерации (3.1.28) при $\alpha_k = 0$, $k = 0, 1, \ldots$, не сходятся.

25. Пусть H и V имеют вид (3.1.25). Показать, что $HV = VH$.

Литература и дополнения к параграфу 3.1

1. Итерации Якоби описаны во многих монографиях. Исчерпывающее изложение основных свойств метода можно найти, например, в книгах [Varga, 1962] и [Young, 1971]. Благодаря почти идеальному параллелизму итерации Якоби были реализованы многими авторами на параллельных и векторных машинах; см., например, [Lambiotte, 1975], где дается обзор ранних работ, а также развивается подход, основанный на использовании векторов полной

длины (см. рис. 3.1.8). Этот подход с использованием матричного умножения был применен Вашпрессом.

2 В основном тексте мы рассмотрели лишь простейший вид распределения точек сетки по процессорам в параллельной системе. В более сложных задачах встречаются нерегулярные области, неравномерные сетки и динамически изменяющиеся сетки. Возникающие при этом вопросы освещаются в работах [Bokhari, 1985], [Berger, Bokhari, 1985], [Gropp, 1986], [McBryan, van de Velde, 1985, 1986a], [Morison, Otto, 1987]. В последней работе рассматриваются отображения нерегулярных областей на гиперкубы или другие сети процессоров посредством «рассеянного» расщепления.

3. Идея асинхронных итераций восходит по крайней мере к «хаотической релаксации», рассмотренной в [Chazan, Miranker, 1969]. Детальное изучение асинхронных итераций, включающее численные эксперименты, можно найти в [Baudet, 1978]; см. также [Barlow, Evans, 1982], [Deminet, 1982], [Dubois, Briggs, 1982]

4. Блочные методы Якоби являются классическими Полинейные методы и их скорости сходимости подробно рассматриваются в монографии [Varga, 1962] Среди более современных работ по блочным методам (и не только по методам Якоби) можно назвать [Parter, Steuerwalt, 1980, 1982], [Fergusson, 1986].

5. Методы ADI восходят к 50-м годам. Хорошее обсуждение этих методов и их свойств сходимости можно найти в монографии [Varga, 1962]. Способы реализации методов ADI на векторных компьютерах рассматриваются в работах [Lambiotte, 1975], [Oppe, Kincaid, 1987], [Kincaid, Oppe, Young, 1986a, b]. В работе [Chan, 1987] описывается реализация метода ADI на гиперкубе с использованием стратегии, исключающей необходимость транспонирования массива данных (см. также работу [Saied et al., 1987]).

6. Хотя дискретное уравнение Пуассона (3 1 5) давно является стандартной модельной задачей для итерационных методов, его можно очень эффективно решать прямыми методами, известными как «быстрые методы решения уравнения Пуассона». Краткий обзор таких методов для векторных и параллельных компьютеров, а также более подробную библиографию можно найти в работе [Ortega, Voigt, 1985]. Среди последних работ по быстрым методам для уравнения Пуассона, использующим расщепление на подобласти, можно назвать статьи [Chan, Resasco, 1987a, b].

3.2. Методы Гаусса — Зейделя и SOR

Рассмотрим теперь другие итерационные методы решения системы $A\mathbf{x} = \mathbf{b}$; здесь мы опять будем предполагать, что диагональные элементы матрицы A отличны от нуля. После вычисления нового итерационного приближения по методу Якоби для x_1

$$x_1^{k+1} = \frac{1}{a_{11}}\left(-\sum_{j=2}^{n} a_{1j}x_j^k + b_1\right)$$

представляется разумным использовать полученное обновленное значение вместо старого x_1^k при вычислении последующих

значений x_i^{k+1}. *Принцип Гаусса — Зейделя* заключается в том, что полученная информация должна использоваться сразу же, как только она становится доступна. Итерации *метода Гаусса — Зейделя* имеют вид

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(-\sum_{j<i} a_{ij}x_j^{k+1} - \sum_{j>i} a_{ij}x_j^k + b_i\right), \qquad (3.2.1)$$

$$i = 1, \ldots, n, \quad k = 0, 1, \ldots.$$

Пусть

$$A = D - L - U, \qquad (3.2.2)$$

где D — диагональная часть матрицы A, а $-L$ и $-U$ — соответственно строго нижняя и строго верхняя треугольные части матрицы A. Тогда нетрудно убедиться (см. упражнение 3.2.1), что соотношения (3.2.1) можно представить в виде

$$D\mathbf{x}^{k+1} = L\mathbf{x}^{k+1} + U\mathbf{x}^k + \mathbf{b},$$

или

$$\mathbf{x}^{k+1} = H\mathbf{x}^k + \mathbf{d}, \quad k = 0, 1, \ldots,$$

$$H = (D - L)^{-1}U, \quad \mathbf{d} = (D - L)^{-1}\mathbf{b}. \qquad (3.2.3)$$

Так как матрица $D - L$ — это не что иное, как нижняя треугольная часть матрицы A, то обратная к ней матрица существует в силу предположения о том, что диагональные элементы матрицы A отличны от нуля.

Опишем важную модификацию метода Гаусса — Зейделя. Обозначим через $\hat{x}_i^{k+1}$ правую часть (3.2.1), задающую итерационное приближение метода Гаусса — Зейделя, и определим

$$x_i^{k+1} = x_i^k + \omega\left(\hat{x}_i^{k+1} - x_i^k\right). \qquad (3.2.4)$$

Подставляя выражение для $\hat{x}_i^{k+1}$ в (3.2.4) и переупорядочивая слагаемые, можно показать, что величины x_i^{k+1} удовлетворяют уравнениям

$$a_{ii}x_i^{k+1} + \omega\sum_{j<i} a_{ij}x_j^{k+1} = (1 - \omega)a_{ii}x_i^k - \omega\sum_{j>i} a_{ij}x_j^k + \omega b_i, \qquad (3.2.5)$$

или, с использованием матриц, фигурирующих в (3.2.2),

$$\mathbf{x}^{k+1} = (D - \omega L)^{-1}\left((1 - \omega)D + \omega U\right)\mathbf{x}^k + \omega(D - \omega L)^{-1}\mathbf{b}, \qquad (3.2.6)$$

$$k = 0, 1, \ldots.$$

Полученная формула определяет итерационный *метод последовательной верхней релаксации* SOR (successive overrelaxation). Заметим, что при $\omega = 1$ (3.2.6) сводится к итерациям Гаусса —

Зейделя. Заметим также, что матрица $D - \omega L$ невырожденна в силу принятого предположения о диагональных элементах матрицы A.

В то время как (3.2.6) и (3.2.3) дают полезные матричные представления итераций SOR и Гаусса — Зейделя, практические вычисления должны выполняться по формулам (3.2.1) и (3.2.4). В любом случае следующее итерационное приближение получается в результате решения системы уравнений с нижнетреугольной матрицей $D - \omega L$. Уравнения (3.2.1) и (3.2.5) явно описывают процесс решения этой системы.

Теоремы сходимости

Приведем теперь несколько основных теорем, устанавливающих сходимость итераций Гаусса — Зейделя и SOR. Доказательства этих теорем, а также некоторые вспомогательные результаты и необходимые определения даются в приложении 2. Всюду далее предполагается, что матрица A невырожденна.

3.2.1. Теорема. *Если матрица A либо имеет строгое диагональное преобладание, либо является неприводимой с диагональным преобладанием, либо является М-матрицей, то итерации метода Гаусса — Зейделя сходятся при любом начальном приближении* $\mathbf{x}^0$.

Следующий результат показывает, что, независимо от свойств матрицы A, вещественный параметр ω метода SOR должен принадлежать интервалу (0, 2), чтобы метод мог сходиться.

3.2.2. Лемма Кахана. *Для того чтобы итерации метода SOR сходились при любом начальном приближении* $\mathbf{x}^0$, *необходимо, чтобы параметр* ω *принадлежал интервалу* (0, 2).

Из этого утверждения, естественно, не вытекает сходимость метода при $\omega \in (0, 2)$. Однако для важного класса матриц указанный интервал действительно описывает истинную область сходимости.

3.2.3. Теорема Островского — Райха. *Если A — симметричная положительно определенная матрица и* $\omega \in (0, 2)$, *то итерации метода SOR сходятся при любом начальном приближении* $\mathbf{x}^0$.

Заметим, что параметр ω был введен для того, чтобы увеличить скорость сходимости итераций Гаусса — Зейделя. Этот вопрос обсуждается в приложении 2.

Блочный метод SOR

Рассмотрим теперь варианты метода SOR, играющие важную роль в дальнейшем обсуждении. Как и метод Якоби, метод Гаусса — Зейделя можно применить к матрице с блочным разбиением

$$A=\begin{bmatrix} A_{11} & \dots & A_{1q} \\ \vdots & & \vdots \\ A_{q1} & \dots & A_{qq} \end{bmatrix}; \tag{3.2.7}$$

в результате получается *блочный метод Гаусса — Зейделя*

$$A_{ii}\mathbf{x}_i^{k+1}=-\sum_{j<i}A_{ij}\mathbf{x}_j^{k+1}-\sum_{j>i}A_{ij}\mathbf{x}_j^{k}+\mathbf{b}_i, \tag{3.2.8}$$

$$i=1,\ \dots,\ q,\quad k=0,\ 1,\ \dots,$$

где разбиение векторов $\mathbf{x}^k$ и $\mathbf{b}$ согласовано с разбиением матрицы A. Здесь предполагается, что все матрицы A_{ii} невырождены. Параметр ω вводится по аналогии с (3.2.4):

$$\mathbf{x}_i^{k+1}=\mathbf{x}_i^{k}+\omega\left(\widehat{\mathbf{x}}_i^{k+1}-\mathbf{x}_i^{k}\right), \tag{3.2.9}$$

где $\widehat{\mathbf{x}}_i^{k+1}$ теперь обозначает итерационное приближение Гаусса — Зейделя, вычисленное по формуле (3.2.8); тем самым определяются итерации *блочного метода SOR*. В качестве примера применения блочных методов рассмотрим опять матрицу (3.1.7), отвечающую дискретному уравнению Пуассона на единичном квадрате. В этом случае (3.2.8) сводится к

$$T\mathbf{x}_i^{k+1}=\mathbf{x}_{i-1}^{k+1}+\mathbf{x}_{i+1}^{k}+\mathbf{b}_i,\quad i=1,\ \dots,\ N. \tag{3.2.10}$$

Как и для соответствующего полинейного метода Якоби, новые итерационные приближения на линии узлов сетки обновляются одновременно, отличие же состоит в том, что полученные итерационные приближения используются сразу же, как только мы переходим к обработке следующей линии узлов. Поэтому (3.2.10) называют *полинейным методом Гаусса — Зейделя*, а соответствующие итерации метода SOR обычно называют *методом последовательной полинейной верхней релаксации SLOR* или *LSOR* (successive line overrelaxation).

Метод SSOR

Другая важная модификация итераций SOR — это *метод симметричной последовательной верхней релаксации SSOR* (symmetric successive overrelaxation). Каждый шаг метода

SSOR осуществляется в два этапа: после итерации метода SOR выполняется итерация SOR с обратным упорядочением. Рассмотрим такую схему сначала для метода Гаусса — Зейделя (при этом получается *симметричный метод Гаусса — Зейделя SGS* (symmetric Gauss — Seidel)). На k-й итерации приближение Гаусса — Зейделя, которое мы будем помечать полуцелым индексом $k+\frac{1}{2}$, получается по формуле

$$\mathbf{x}^{k+\frac{1}{2}} = (D-L)^{-1}U\mathbf{x}^k + (D-L)^{-1}\mathbf{b}. \qquad (3.2.11$$

Далее мы проходим по всем уравнениям в обратном порядке, используя только что вычисленные значения $\mathbf{x}^{k+\frac{1}{2}}$:

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(-\sum_{j>i} a_{ij}x_j^{k+1} - \sum_{j<i} a_{ij}x_j^{k+\frac{1}{2}} + b_i\right),$$
$$i = n,\ n-1,\ \ldots,\ 1.$$

С помощью матриц D, L и U эти соотношения можно представить в виде

$$D\mathbf{x}^{k+1} = U\mathbf{x}^{k+1} + L\mathbf{x}^{k+\frac{1}{2}} + \mathbf{b}$$

или

$$\mathbf{x}^{k+1} = (D-U)^{-1}L\mathbf{x}^{k+\frac{1}{2}} + (D-U)^{-1}\mathbf{b}. \qquad (3.2.12)$$

Видно, что на этом этапе матрицы L и U меняются ролями. Одна итерация метода SGS получается, таким образом, в результате комбинирования (3.2.11) и (3.2.12):

$$\mathbf{x}^{k+1} = (D-U)^{-1}L(D-L)^{-1}U\mathbf{x}^k + \hat{\mathbf{d}}, \qquad (3.2.13)$$

где

$$\hat{\mathbf{d}} = (D-U)^{-1}L(D-L)^{-1}\mathbf{b} + (D-U)^{-1}\mathbf{b}.$$

Для того чтобы получить метод SSOR, достаточно просто вставить параметр ω в формулы, описывающие оба полушага, подобно тому, как это сделано в (3.2.4) — (3.2.6). Матричная запись итерации метода SSOR принимает тогда следующий вид:

$$\mathbf{x}^{k+1} = (D-\omega U)^{-1}((1-\omega)D + \omega L)(D-\omega L)^{-1}\times$$
$$\times((1-\omega)D + \omega U)\mathbf{x}^k + \hat{\mathbf{d}}, \qquad (3.2.14)$$

где теперь

$$\hat{\mathbf{d}} = \omega (D - \omega U)^{-1} (((1 - \omega) D + \omega L)(D - \omega L)^{-1} + I)\, \mathbf{b}.$$

Конечно, при выполнении вычислений метод SSOR реализуется по простым формулам типа (3.2.1) и (3.2.4), а довольно громоздкое выражение (3.2.14) является лишь матричным представлением рассматриваемого итерационного процесса. С позиций дискретизации уравнений в частных производных каждая итерация симметричного метода Гаусса — Зейделя или метода SSOR сводится к проходу по точкам сетки на первом полушаге с последующим проходом в обратном порядке на втором полушаге.

Основная теорема о сходимости метода SSOR соответствует теореме 3.2.3 Островского — Райха; доказательство приведено в приложении 2.

3.2.4. Теорема о сходимости метода SSOR. *Если A — симметричная положительно определенная матрица и $\omega \in (0, 2)$, то итерации метода SSOR сходятся при любом начальном приближении $\mathbf{x}^0$.*

Красно-черное упорядочение

Обратимся теперь к параллельной и векторной реализации методов, изучающихся в данном параграфе. Начнем с метода Гаусса — Зейделя.

В силу (3.2.3) итерация метода Гаусса — Зейделя может рассматриваться как решение системы уравнений с нижнетреугольной матрицей. Эта задача довольно подробно анализировалась в предыдущей главе, где было отмечено, что параллельные методы оказываются для нее лишь умеренно эффективными. Однако метод Гаусса — Зейделя (и SOR) применяется на практике исключительно для разреженных систем уравнений, и в этом случае методы решения треугольных систем, описанные в предыдущей главе, могут оказаться совершенно неудовлетворительными.

Одним из перспективных подходов к параллельной реализации метода SOR является такое переупорядочение уравнений линейной системы, чтобы решение соответствующей нижнетреугольной системы можно было эффективно распараллелить. Заметим, что, в отличие от итераций Якоби, итерации Гаусса — Зейделя существенно зависят от порядка уравнений. Таким образом, для заданных уравнений

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, \ldots, n,$$

мы можем выбрать какой-либо иной порядок применения принципа Гаусса — Зейделя, отличный от заданного упорядочения $i = 1, \ldots, n$. Другими словами, мы можем применить обычный метод Гаусса — Зейделя к системе уравнений $PA\mathbf{x} = P\mathbf{b}$, где P — матрица перестановки, задающая желаемый порядок обработки уравнений. Однако, поскольку основным предположением, гарантирующим применимость метода Гаусса — Зейделя, было отличие от нуля диагональных элементов a_{ii}, хотелось бы сохранить указанное свойство при переупорядочении. Поэтому наряду с перестановкой уравнений мы будем производить соответствующую перенумерацию неизвестных. Это эквивалентно

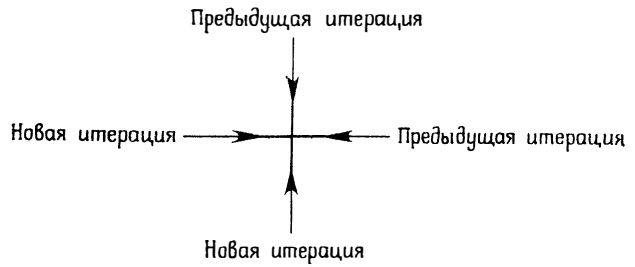


Рис. 3.2.1. Пересчет в одной точке по методу Гаусса — Зейделя

рассмотрению новых матриц коэффициентов вида PAP^{T}, где P — матрица перестановки. Для заданной системы из n уравнений существует $n!$ возможных переупорядочений такого типа и, следовательно, $n!$ соответствующих методов Гаусса — Зейделя (и методов SOR). Наша задача — установить, какая из этих $n!$ возможных перестановок (если она вообще существует) приводит матрицу к виду PAP^{T}, подходящему для параллельных и векторных вычислений.

Чтобы подойти к этой задаче, сначала сосредоточим внимание на дискретном уравнении Пуассона (3.1.5) на квадратной сетке. Обновление значения итерационного приближения в точке сетки по методу Гаусса — Зейделя (3.2.1) схематически показано на рис. 3.2.1. В предположении, что обход точек сетки осуществляется слева направо и снизу вверх, рис. 3.2.1 описывает ситуацию в (i, j)-й точке сетки: новые итерационные приближения, уже вычисленные в процессе обхода точек сетки, комбинируются с приближениями, вычисленными на предыдущей итерации, для получения нового итерационного приближения к решению в (i, j)-й точке сетки. Непосредственная векторизация указанных вычислений (как для метода Якоби) в данном случае не подходит, хотя позднее мы обсудим некоторый ее вариант.

Руководствуясь предыдущим обсуждением переупорядочения уравнений, перенумеруем теперь точки сетки и соответствующие неизвестные специальным образом. Важным типом переупорядочения, ставшим классическим, является *красно-черное* (или *шашечное*) переупорядочение, показанное на рис. 3.2.2. Точки сетки сначала разделяются на два подмножества, «красное» и «черное», а затем упорядочиваются внутри каждого из этих

$\dot{B}6$	$\dot{R}6$	$\dot{B}7$	$\dot{R}7$	$\dot{B}8$
$\dot{R}3$	$\dot{B}4$	$\dot{R}4$	$\dot{B}5$	$\dot{R}5$
$\dot{B}1$	$\dot{R}1$	$\dot{B}2$	$\dot{R}2$	$\dot{B}3$

Рис. 3.2.2. Красно-черное упорядочение точек сетки

подмножеств. На рис. 3.2.2 показано упорядочение слева направо и снизу вверх для каждого из двух подмножеств. Неизвестные в точках сетки упорядочиваются аналогично: например, u в точке $R1$ будет первым неизвестным, u в точке $R2$ — вторым и так далее, пока не будут исчерпаны все красные точ-

$$\left[\begin{array}{cccc} 4 & 0 & -1 & -1 \\ 0 & 4 & -1 & -1 \\ -1 & -1 & 4 & 0 \\ -1 & -1 & 0 & 4 \end{array}\right]\left[\begin{array}{c} u_{R1} \\ u_{R2} \\ u_{B1} \\ u_{B2} \end{array}\right]=\left[\begin{array}{c} \\ \\ \\ \end{array}\right]\quad\left[\begin{array}{cc} \dot{B}2 & \dot{R}2 \\ \dot{R}1 & \dot{B}1 \end{array}\right]$$

Рис. 3.2.3. Красно-черное упорядочение уравнений для четырех неизвестных

ки; затем этот процесс продолжается для черных точек. Получающееся упорядочение иллюстрирует рис. 3.2.3, где явно выписаны уравнения (без правых частей) для дискретного уравнения Пуассона в случае $N = 2$.

Для любого значения N в любой внутренней точке сетки красные точки окажутся связанными только с черными, и наоборот (рис. 3.2.4). Таким образом, при условии, что все красные точки упорядочены первыми, в матричной форме система будет иметь вид

$$\left[\begin{array}{cc} D_R & C \\ C^{\mathrm{T}} & D_B \end{array}\right]\left[\begin{array}{c} \mathbf{u}_R \\ \mathbf{u}_B \end{array}\right]=\left[\begin{array}{c} \mathbf{b}_1 \\ \mathbf{b}_2 \end{array}\right]. \qquad (3.2.15)$$

Здесь $D_R = 4I_R$ и $D_B = 4I_B$, где I_R — единичная матрица порядка, равного числу внутренних красных точек, и аналогичным

образом определяется I_B. Матрица C описывает связи между красными и черными неизвестными. Частный случай (3.2.15) для $N = 2$ показан на рис. 3.2.3. Заметим, что матрица коэффициентов в (3.2.15) должна быть симметричной, поскольку она является матрицей вида PAP^{T}, где A — симметричная матрица.

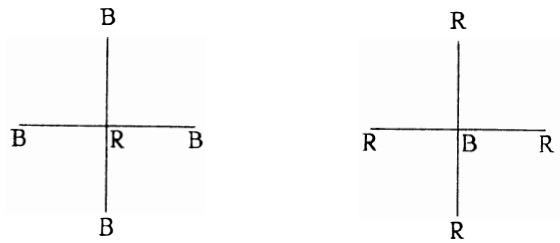


Рис. 3.2.4. Красно-черные шаблоны

Рассмотрим теперь итерации Гаусса — Зейделя для системы (3.2.15):

$$\begin{bmatrix} D_R & 0 \\ C^{\mathrm{T}} & D_B \end{bmatrix} \begin{bmatrix} \mathbf{u}_R^{k+1} \\ \mathbf{u}_B^{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} - \begin{bmatrix} 0 & C \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_R^{k} \\ \mathbf{u}_B^{k} \end{bmatrix}. \quad (3.2.16)$$

Эти соотношения делятся на две части:

$$\mathbf{u}_R^{k+1} = D_R^{-1}(\mathbf{b}_1 - C\mathbf{u}_B^k), \quad \mathbf{u}_B^{k+1} = D_B^{-1}(\mathbf{b}_2 - C^{\mathrm{T}}\mathbf{u}_R^{k+1}). \quad (3.2.17)$$

Таким образом, хотя (3.2.16) формально требует решения системы с нижнетреугольной матрицей для того, чтобы продвинуться еще на один итерационный шаг, то обстоятельство, что матрицы D_R и D_B являются диагональными, позволяет свести решение этой системы к матрично-векторным умножениям (3.2.17).

Для итераций метода SOR параметр ω вводится обычным образом. Применяя (3.2.6) к (3.2.16), получаем

$$\begin{bmatrix} D_R & 0 \\ \omega C^{\mathrm{T}} & D_B \end{bmatrix} \begin{bmatrix} \mathbf{u}_R^{k+1} \\ \mathbf{u}_B^{k+1} \end{bmatrix} = \begin{bmatrix} \omega\mathbf{b}_1 \\ \omega\mathbf{b}_2 \end{bmatrix} + \begin{bmatrix} (1-\omega)D_R & -\omega C \\ 0 & (1-\omega)D_B \end{bmatrix} \begin{bmatrix} \mathbf{u}_R^{k} \\ \mathbf{u}_B^{k} \end{bmatrix}$$

или

$$\begin{aligned} \mathbf{u}_R^{k+1} &= D_R^{-1}(\omega\mathbf{b}_1 + (1-\omega)D_R\mathbf{u}_R^k - \omega C\mathbf{u}_B^k), \\ \mathbf{u}_B^{k+1} &= D_B^{-1}(\omega\mathbf{b}_2 + (1-\omega)D_B\mathbf{u}_B^k - \omega C^{\mathrm{T}}\mathbf{u}_R^{k+1}). \end{aligned} \quad (3.2.18)$$

Полученные соотношения представляют собой матричную запись итерационного шага метода SOR, однако при практиче-

ском выполнении вычислений нет необходимости в явном использовании этих матричных формул. Вместо этого сначала вычисляются $\hat{\mathbf{u}}_R^{k+1}$, то есть приближения Гаусса — Зейделя для красных точек, а затем приближения метода SOR получаются по формуле

$$\mathbf{u}_R^{k+1} = \mathbf{u}_R^k + \omega\left(\hat{\mathbf{u}}_R^{k+1} - \mathbf{u}_R^k\right). \tag{3.2.19}$$

С использованием полученных обновленных значений в красных точках аналогичным образом можно вычислить новые приближения метода SOR в черных точках. Новые итерационные приближения метода Гаусса — Зейделя для красных точек совпадают с приближениями метода Якоби для этих же точек; аналогичное утверждение справедливо для черных точек. Таким образом, итерация метода SOR реализуется при помощи двух итераций метода Якоби, каждая из которых затрагивает свою часть (примерно половину) узлов сетки.

Параллельная реализация

Предположим сначала, что в нашем распоряжении есть $p = N^2/2$ процессоров, соединенных в виде решетки, как показано на рис. 3 1.2. Припишем каждому из этих процессоров одну красную и одну черную точку, как показано на рис. 3.2.5. Предполагается, что процессоры, смежные с границей области, содержат соответствующие граничные значения. Чтобы выполнить одну итерацию Гаусса — Зейделя, все процессоры сначала параллельно пересчитывают свои красные значения, а затем свои черные значения. Как уже отмечалось, производятся, в сущности, те же вычисления, что и в итерациях Якоби. После каждого полушага можно вставить пересчет с использованием параметра ω для того, чтобы получить приближения метода SOR. Для красных точек это будут вычисления по формулам (3.2.19), и аналогично для черных точек.

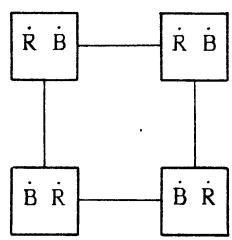


Рис. 3.2.5. Распределение красных и черных неизвестных по процессорам

При более реалистическом предположении $p \ll N^2$ неизвестные нужно подходящим образом разместить в имеющихся процессорах. Например, если $N^2 = 2mp$, то каждому процессору приписывается m красных и m черных точек. Если $2p$ не является делителем N^2, то мы припишем каждому процессору настолько близкие количества красных и черных точек,

насколько это возможно, однако полностью избежать несбалансированности распределения не удастся. Заметим, что крайний случай $p = N^2$ (на каждое неизвестное приходится по одному процессору), который обсуждался при анализе итераций Якоби, здесь окажется невыгодным, так как в каждый момент половина процессоров будет простаивать, пока будут выполняться вычисления с неизвестными другого цвета.

Для описанных выше итераций SOR справедливы те же рассуждения, касающиеся организации операций обмена, проверки сходимости итераций и синхронизации, что и для метода Якоби. В нашем случае синхронизация должна выполняться после каждого шага Якоби; таким образом, перед тем, как начать вычисления с красными точками, необходимо вычислить все черные значения и передать их тем процессорам, которым они потребуются, и наоборот. Альтернативой является, как и в случае итераций Якоби, асинхронное выполнение итераций; получающийся метод мы будем называть *асинхронным красно-черным* методом SOR.

Векторизация

Векторизация красно-черных итераций SOR выполняется, в сущности, так же, как и для метода Якоби. Рассмотрим сначала векторный компьютер, который использует векторные регистры, причем значение $N/2$ является подходящей длиной вектора. Предположим, что красные неизвестные на каждой строке точек сетки хранятся как векторы и аналогичное соглашение принято для черных неизвестных. Тогда для дискретного уравнения Пуассона пересчет красных неизвестных по методу Гаусса — Зейделя на i-й линии точек сетки выполняется следующим образом:

$$UR(I,) = 0.25\,(UB(I,)_{-1} + UB(I,)_{+1} + UB(I-1,) + UB(I+1,)), \tag{3.2.20}$$

где через $UB(I,)_{-1}$ обозначен вектор черных неизвестных на i-й линии, сдвинутый влево на одну позицию, и, аналогично, $UB(I,)_{+1}$ обозначает тот же вектор, но сдвинутый на одну позицию вправо. Предполагается, что все векторы UB и UR содержат также граничные значения в соответствующих строках и что векторы с индексами 0 или $N+1$ содержат граничные значения на нижней и верхней линиях сетки. Если функция f, фигурирующая в правой части уравнения Пуассона, отлична от нуля, то ее следует добавить к выражению (3.2.20); очевидным образом можно также ввести релаксационный параметр ω, чтобы получить итерационную схему SOR. Построение

схемы распределения векторных регистров, соответствующей рис. 3.1.5, мы оставляем в качестве упражнения 3.2.5.

Если для векторного компьютера желательно использование как можно более длинных векторов, то можно применить процедуры, аналогичные построенным ранее для метода Якоби. В соответствии с подходом, показанным на рис. 3.1.8, теперь мы имеем дело с двумя векторами, каждый из которых имеет длину около $N^2/2$, причем один содержит все значения неизвестных в красных точках, а другой — все значения в черных точках. Каждый из этих векторов содержит также и граничные значения соответствующего цвета. Иллюстрацией служат формулы (3.2.21), соответствующие рис. 3.2.2, где предполагается, что крайние точки снизу и с двух боковых сторон являются граничными точками. Тогда уравнения (3.2.21) дают представление о вычислениях, позволяющих получить новые приближения Гаусса — Зейделя в красных точках, исходя из значений в черных точках:

$$T=\begin{bmatrix} B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}+\begin{bmatrix} B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix},\quad R=\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}+\begin{bmatrix} T_4 \\ T_5 \\ T_6 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}=\begin{bmatrix} B_2+B_4+B_5+B_7 \\ B_3+B_5+B_6+B_8 \\ B_4+B_6+B_7+B_9 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}*\cdot \tag{3.2.21}$$

После выполнения (3.2.21) для получения новых значений в красных точках вектор R нужно умножить на 0.25. Аналогичные вычисления позволяют получить новые красные значения, исходя из черных. В указанную вычислительную схему легко ввести релаксационный параметр метода SOR и правую часть f. Как и в случае метода Якоби, использование формул (3.2.21) приводит к «порче» граничных значений, если не обеспечивается подавление операции записи в память. Одно из таких «ложных» значений отмечено в (3.2.21) звездочкой.

Формула (3.2.21) иллюстрирует вычисления, выполняемые для сетки, показанной на рис. 3.2.2, где $N = 3$. Она допускает очевидное обобщение на случай нечетного N, однако для сетки с четным N такое обобщение не проходит. Это иллюстрирует рис. 3.2.6, где показана сетка с четырьмя внутренними точками $B3$, $R4$, $R5$ и $B6$. Вычисления, аналогичные (3.2.21), выглядят

следующим образом:

$$T = \begin{bmatrix} B_2 \\ B_3 \\ B_4 \\ B_5 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} + \begin{bmatrix} B_3 \\ B_4 \\ B_5 \\ B_6 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}, \quad R = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} + \begin{bmatrix} T_3 \\ T_4 \\ T_5 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} B_2 + B_3 + B_4 + B_5 \\ B_3 + B_4 + B_5 + B_6 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix},$$

что приводит к неверному результату. Во многих случаях значение N определяется методом решения дифференциальной задачи и может быть выбрано нечетным. Если N должно быть

$R7$	$B7$	$R8$	$B8$
$B5$	$R5$	$B6$	$R6$
$R3$	$B3$	$R4$	$B4$
$B1$	$R1$	$B2$	$R2$

Рис. 3.2.6. Случай четырех внутренних точек сетки

четным, то можно применить простой искусственный прием, заключающийся в расширении сетки путем добавления дополнительного столбца узлов с граничными значениями. При этом сетка, показанная на рис. 3.2.6, заменится на сетку, отвечающую рис. 3.2.2, причем внутренними будут точки $B4$, $R4$, $R6$ и $B7$. Крайний правый столбец может содержать произвольные значения; они будут участвовать в вычислениях, не влияя существенным образом на результаты. Теперь вычисления, выполняемые в соответствии с (3.2.21), будут давать верные значения во внутренних точках сетки. Такое добавление фиктивных граничных точек повлечет за собой некоторую потерю эффективности как в отношении времени вычислений, так и в смысле объема памяти (см. упражнение 3.2.7), и поэтому, если есть такая возможность, желательно выбирать значение N нечетным.

Переход к вычислениям с длинными векторами, основанный на использовании матричного умножения, который обсуждался в случае применения метода Якоби к уравнению (3.1.14), также можно распространить на красно-черный метод SOR. Это осуществляется на основе формул (3.2.17) или (3.2.18). Однако теперь средняя длина векторов будет более чем в два раза меньше той, которая была достигнута для метода Якоби, так

как красно-черное упорядочение приводит к распределению элементов длинных диагоналей матрицы перехода метода Якоби между несколькими более короткими диагоналями матрицы C.

Многоцветные упорядочения

Выполнение итераций метода SOR с красно-черным упорядочением точек сетки ограничено преимущественно простыми уравнениями в частных производных, такими, как уравнение Пуассона, дискретизованными простейшим образом.

Рассмотрим, например, на единичном квадрате уравнение

$$u_{xx} + u_{yy} + au_{xy} = 0 \quad (3.2.22)$$

(где a — некоторая постоянная). Оно представляет собой уравнение Лапласа с добавочным членом, содержащим смешанную производную. Стандартная конечно-разностная аппроксимация выражения u_{xy} имеет вид

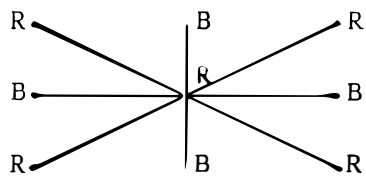


Рис. 3.2.7. Красно-черное упорядочение девятиточечного шаблона

$$u_{xy} = \frac{1}{4h^2}(u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}), \quad (3.2.23)$$

что дает, в комбинации с приведенной выше аппроксимацией (3.1.5) для уравнения Лапласа, систему разностных уравнений

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij} + \\ + \frac{a}{4}(u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}) = 0, \quad (3.2.24)$$

$$i,\ j = 1,\ \ldots,\ N.$$

Для красно-черного упорядочения точек сетки, аналогичного показанному на рис. 3.2.2, нетрудно убедиться (см. упражнение 3.2.8), что систему (3.2.24) можно записать в форме

$$\begin{bmatrix} D_R & C \\ C^T & D_B \end{bmatrix} \begin{bmatrix} \mathbf{u}_R \\ \mathbf{u}_B \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}, \quad (3.2.25)$$

как это было ранее сделано для уравнения Пуассона. Однако матрицы D_R и D_B в (3.2.25) уже не будут диагональными. Проблема заключается в том, что неизвестное в (i, j)-й точке сетки связано теперь с неизвестными в восьми ближайших соседних точках, часть из которых имеет тот же цвет, что и центральная точка. Иллюстрацией служит рис. 3.2.7.

Решение этой проблемы достигается за счет введения большего числа цветов, как показано на рис. 3.2.8. Если мы сначала запишем все уравнения для красных точек, затем для черных и так далее, то рассматриваемая система примет вид

$$\begin{bmatrix} D_1 & B_{12} & B_{13} & B_{14} \\ B_{21} & D_2 & B_{23} & B_{24} \\ B_{31} & B_{32} & D_3 & B_{34} \\ B_{41} & B_{42} & B_{43} & D_4 \end{bmatrix} \begin{bmatrix} \mathbf{u}_R \\ \mathbf{u}_B \\ \mathbf{u}_G \\ \mathbf{u}_W \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \mathbf{b}_4 \end{bmatrix}, \tag{3.2.26}$$

где диагональные блоки D_i являются диагональными матрицами. Итерация Гаусса — Зейделя в этом случае выглядит так:

$$\begin{aligned} D_1\mathbf{u}_R^{k+1} &= -B_{12}\mathbf{u}_B^k - B_{13}\mathbf{u}_G^k - B_{14}\mathbf{u}_W^k + \mathbf{b}_1, \\ D_2\mathbf{u}_B^{k+1} &= -B_{21}\mathbf{u}_R^{k+1} - B_{23}\mathbf{u}_G^k - B_{24}\mathbf{u}_W^k + \mathbf{b}_2, \\ D_3\mathbf{u}_G^{k+1} &= -B_{31}\mathbf{u}_R^{k+1} - B_{32}\mathbf{u}_B^{k+1} - B_{34}\mathbf{u}_W^k + \mathbf{b}_3, \\ D_4\mathbf{u}_W^{k+1} &= -B_{41}\mathbf{u}_R^{k+1} - B_{42}\mathbf{u}_B^{k+1} - B_{43}\mathbf{u}_G^{k+1} + \mathbf{b}_4. \end{aligned} \tag{3.2.27}$$

Поскольку D_i — диагональные матрицы, решение треугольной системы, необходимое для выполнения итерации Гаусса — Зейделя, опять сводится к матрично-векторным умножениям.

R	*B*	*G*	*W*	*R*	*B*	*G*	*W*
G	*W*	*R*	*B*	*G*	*W*	*R*	*B*
R	*B*	*G*	*W*	*R*	*B*	*G*	*W*

Рис. 3.2.8. Четырехцветное упорядочение

Четырехцветное упорядочение, показанное на рис. 3.2.8, основано на той связи точек сетки, которую иллюстрирует рис. 3.2.7; мы будем называть подобные образцы связи между точками сетки *шаблонами*. Шаблон показывает, каким образом связаны точки сетки со своими соседями; он зависит как от дифференциального уравнения, так и от способа его дискретизации. Некоторые другие достаточно распространенные шаблоны приводятся на рис. 3.2.9.

Количество цветов, указанное в пояснениях к рис. 3.2.9, определено в предположении, что в каждой точке сетки принят один и тот же шаблон. В таком случае критерием удачной раскраски служит требование, чтобы любая точка сетки, взятая в качестве центральной точки шаблона, имела цвет, отличный

от цвета всех остальных точек шаблона. Это и есть то «локальное разделение» неизвестных, которое позволяет получить матричную запись задачи, имеющую в общем случае вид

$$\begin{bmatrix} D_1 & B_{12} & \dots & & B_{1,c} \\ B_{21} & D_2 & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & B_{c-1,c} \\ B_{c,1} & & \dots & B_{c,c-1} & D_c \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{u}_c \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{b}_c \end{bmatrix}, \quad (3.2.28)$$

где диагональные блоки D_i представляют собой диагональные матрицы. Будем называть матрицы вида (3.2.28) *c-цветными* матрицами. Заметим, что в (3.2.26) $c = 4$, а для красно-чер-

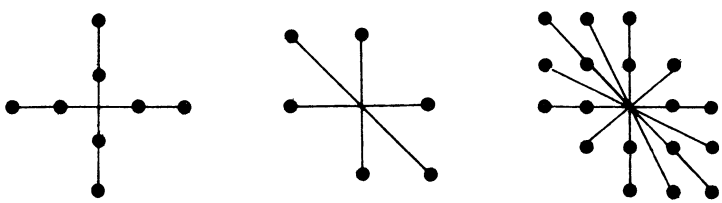


Рис. 3.2.9. Некоторые распространенные шаблоны: (a) девятиточечный, раскрашивается в 3 цвета, (b) соответствующий применению линейных конечных элементов, раскрашивается в 3 цвета; (c) соответствующий применению квадратичных конечных элементов, раскрашивается в 6 цветов

ного упорядочения $c = 2$. Для системы, представленной в виде (3.2.28), итерацию метода Гаусса — Зейделя можно выполнить (по аналогии с (3.2.27)) по формулам

$$\mathbf{u}_i^{k+1} = D_i^{-1}\left(\mathbf{b}_i - \sum_{j<i} B_{ij}\mathbf{u}_j^{k+1} - \sum_{j>i} B_{ij}\mathbf{u}_j^{k}\right), \quad i = 1, \dots, c, \quad (3.2.29)$$

и опять решение треугольной системы сводится к выполнению операций, аналогичных использовавшимся для реализации метода Якоби.

Вообще говоря, желательно использовать минимальное число цветов, позволяющее добиться приведения матрицы к виду (3.2.28). Для произвольных шаблонов, могущих изменяться от одной точки сетки к другой, и произвольных сеток это является трудной задачей. Однако если для каждой точки сетки шаблон остается одним и тем же, способ минимизации числа цветов обычно оказывается очевидным. Но такой способ раскраски не обязательно будет единственным, даже если используется минимальное число цветов. Рис. 3.2.10 дает четырехцветное

упорядочение для шаблона на рис. 3.2.7, отличное от показанного на рис. 3.2.8; на нем также дано трехцветное упорядочение для шаблонов (a) и (b) на рис. 3.2.9. Отыскание шестицветного упорядочения для шаблона (c) на рис. 3.2.9 мы оставляем в качестве упражнения 3.2.10.

Заметим, что для систем уравнений в частных производных число цветов обычно увеличивается во столько раз, сколько

G	*W*	*G*	*W*	*G*	*W*		*R*	*B*	*G*	*R*	*B*	*G*
R	*B*	*R*	*B*	*R*	*B*		*B*	*G*	*R*	*B*	*G*	*R*
G	*W*	*G*	*W*	*G*	*W*		*G*	*R*	*B*	*G*	*R*	*B*
R	*B*	*R*	*B*	*R*	*B*		*R*	*B*	*G*	*R*	*B*	*G*

Рис. 3.2.10. Раскрашивания для рисунков 3.2.7 и 3.2.9: (a) четырехцветное раскрашивание; (b) трехцветное раскрашивание

уравнений в системе. Таким образом, если, например, сеточный шаблон требует трех цветов и в системе связано два дифференциальных уравнения, то для того, чтобы получить c-цветную матрицу, потребуется шесть цветов. Заметим также, что скорость сходимости методов SOR и Гаусса — Зейделя зависит от упорядочения уравнений и влияние многоцветных упорядочений на скорость сходимости изучено еще не полностью.

Параллельная и векторная реализации

Обратимся теперь к вопросам реализации итераций SOR и Гаусса — Зейделя с использованием многоцветных упорядочений. Для простоты рассмотрим опять квадратную сетку с N^2 внутренними точками. Начнем с анализа параллельной системы с локальной памятью, образованной p процессорами, соединенными в виде двумерной решетки, и предположим, что $N^2 = cmp$, где c — количество цветов. Таким образом, каждый процессор будет содержать mc неизвестных. Это иллюстрирует рис. 3.2.11, где $m = 1$ и $c = 4$, причем используется раскрашивание, показанное на рис. 3.2.8. Сначала все процессоры пересчитывают свои значения в красных точках и передают вычисленные значения тем процессорам, которым они потребуются. На рис. 3.2.11 мы использовали шаблон, показанный на рис. 3.2.7, чтобы показать, как происходит обмен данными. Обновленное значение в красной точке, вычисленное процессором P_{12}, должно

быть передано процессорам P_{11}, P_{21} и P_{22}, но не процессорам P_{13} и P_{23}. Затем пересчитываются все значения в черных точках, за ними — в зеленых и, наконец, в белых. В промежутках между этими стадиями осуществляется передача данных и синхронизация; альтернативой может служить асинхронное выполнение вычислений на итерациях. Для каждого множества значений, соответствующих определенному цвету, мы выполняем операции типа тех, которые отвечают методу Якоби, а затем коррекцию с использованием параметра ω, предусматриваемую методом SOR. Таким образом, каждая итерация реализуется с помощью четырех проходов метода Якоби, каждый из которых затрагивает примерно $N^2/4$ точек сетки, или, в общем случае, при помощи c проходов метода Якоби, если используется c цветов. Принятое нами условие $N^2 = cmp$ связано с неявным предположением, что пересчет в каждой точке сетки требует одного и того же объема вычислений. Поэтому, приписывая каждому процессору mc неизвестных, мы достигаем сбалансированности вычислительной загрузки.

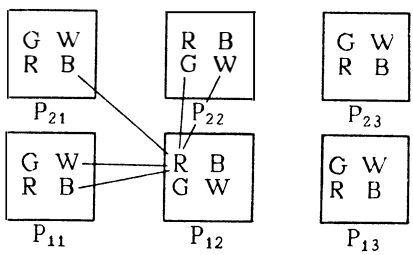


Рис. 3.2.11. Распределение неизвестных по процессорам

Рассмотрим теперь реализацию метода на векторных компьютерах, использующих векторные регистры. Предположим для простоты, что область является квадратом или прямоугольником и что неизвестные хранятся в c двумерных массивах, по одному на каждый цвет, с упорядочением точек сетки слева направо и снизу вверх. При этом граничные значения размещаются в соответствующих позициях этих массивов, а значения правой части f уравнения на сетке — в отдельном массиве. Тогда сначала пересчитываются неизвестные первого цвета, затем второго цвета, и т. д. Векторные операции в этом случае имеют длину примерно N/c.

В качестве примера рассмотрим разностные уравнения (3.2.24) на квадратной сетке с четырехцветным раскрашиванием, показанным на рис. 3.2.8. Неизвестные будут размещаться в четырех массивах: UR, UB, UG и UW. Верхний и нижний ряды точек сетки, а также первая и последняя позиции в каждом ряду точек будут содержать граничные значения. Тогда пересчет значений в красных точках по методу Гаусса — Зейделя в i-м ряду точек будет выполняться посредством векторных

операций

$$UR(I,) = 0.25(UB(I,)_{+1} + UW(I,)_{-1} + UG(I+1,) + UG(I-1,)) + \\ + 0.0625a(UB(I+1,)_{-1} - UB(I-1,)_{-1} - \\ - UW(I+1,)_{+1} + UW(I-1,)_{+1}), \qquad (3.2.30)$$

где использованы те же обозначения, что и в (3.2.20). Аналогичные операции позволяют осуществить пересчет в черных, зеленых и белых точках.

На векторных компьютерах, требующих для повышения эффективности перехода к вычислениям с длинными векторами, удобно работать с одномерными массивами длины $O(N^2/c)$, по одному для каждого цвета. Как раньше для уравнения Пуассона, указанные векторы будут также содержать граничные значения, и мы будем предполагать, что запись в граничные позиции подавляется. Используем опять в качестве примера уравнения (3.2.24) и раскрашивание, показанное на рис. 3.2.10.

*G*8	*W*8	*R*9	*B*9	*G*9	*W*9	*R*10	*B*10	*G*10	*W*10
*R*6	*B*6	*G*6	*W*6	*R*7	*B*7	*G*7	*W*7	*R*8	*B*8
*G*3	*W*3	*R*4	*B*4	*G*4	*W*4	*R*5	*B*5	*G*5	*W*5
*R*1	*B*1	*G*1	*W*1	*R*2	*B*2	*G*2	*W*2	*R*3	*B*3

Рис. 3.2.12. Многоцветное упорядочение с нумерацией

Это раскрашивание воспроизведено на рис. 3.2.12, где указано также упорядочение точек сетки. Для упрощения записи в приведенной ниже схеме пересчета значений в красных точках по методу Гаусса — Зейделя мы не используем специальный прием, показанный на рис. 3.1.8, как это делалось для уравнения Пуассона; соответствующие рассмотрения оставлены в качестве упражнения 3.2.15. Мы опускаем также умножения на константы 0.25 и $0.0625a$:

$$\begin{array}{cccccccccc} & R_4 & G_1 & G_6 & W_3 & B_4 & W_6 & W_1 & B_6 & B_1 & \\ & R_5 & G_2 & G_7 & W_4 & B_5 & W_7 & W_2 & B_7 & B_2 & \\ * & R_6 & G_3 & G_8 & W_5 & B_6 & W_8 & W_3 & B_8 & B_3 & \\ & R_7 = G_4 & + G_9 & + W_6 & + B_7 & + W_9 & + W_4 & + B_9 & + B_4 & . \\ & R_8 & G_5 & G_{10} & W_7 & B_8 & W_{10} & W_5 & B_{10} & B_5 & \\ & R_9 & G_6 & G_{11} & W_8 & B_9 & W_{11} & W_6 & B_{11} & B_6 & (3.2.31) \\ & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \end{array}$$

Аналогичные вычисления позволяют получить новые значения неизвестных в черных, зеленых и белых точках. Звездочкой в (3.2.31) помечено вычисление, дающее неверный результат, запись которого в соответствующую граничную позицию должна быть подавлена. Уравнения (3.2.31) служат иллюстрацией вычислений, выполняемых при $N = 8$. В общем случае для этой задачи требуется, чтобы N являлось целым числом вида $N = 4M$, где M также целое.

Полинейный метод SOR

Теперь кратко обсудим параллельную и векторную реализации полинейного метода SOR. Рассмотрим сначала дискретное уравнение Пуассона и полинейный метод Гаусса — Зейделя (3.2.10) для разностных уравнений, записанных в «естественном» порядке. Эффективное распараллеливание (3.2.10) встречает те же трудности, что и для поточечного метода Гаусса —

B	B	B	...	B
R	R	R	...	R
B	B	B	...	B
R	R	R	...	R

Рис. 3.2.13. Полинейное красно-черное упорядочение

G	G	G	...	G
B	B	B	...	B
R	R	R	...	R
G	G	G	...	G
B	B	B	...	B
R	R	R	...	R

Рис. 3.2.14. Полинейное трехцветное упорядочение

Зейделя, и опять одним из путей их преодоления является переупорядочение уравнений. На этот раз, однако, мы будем раскрашивать точки сетки по линиям, как показано на рис. 3.2.13. Иногда это упорядочение называют «*зеброй*» (заменяя красный цвет белым). Если принято упорядочение, показанное на рис. 3.2.13, то уравнения для неизвестных на i-й линии связывают красные точки только с черными точками, расположенными на смежных линиях. Если мы обозначим через $u_{R,i}$ вектор, образованный неизвестными на i-й красной линии, и введем аналогичное обозначение для неизвестных на черных линиях, то шаг полинейного метода Гаусса — Зейделя, соответствующий пересчету значений в красных точках, имеет вид

$$T\mathbf{u}_{R,i}^{k+1} = \mathbf{u}_{B,i-1}^{k} + \mathbf{u}_{B,i}^{k} + \mathbf{b}_i, \quad i = 1, \ldots, \frac{N}{2}, \qquad (3.2.32)$$

и аналогично для пересчета в черных точках. Трехдиагональные системы (3.2.32) можно теперь решать параллельно или

при помощи сквозной векторизации, как это делалось для полинейного метода Якоби. Единственное различие заключается в том, что теперь длина векторов составляет $O(N/2)$.

Для более общих уравнений и/или дискретизаций применяется тот же принцип, хотя могут потребоваться дополнительные цвета. Возвращаясь к рис. 3.2.9, можно заметить, что красно-черное раскрашивание по линиям еще пригодно для шаблона (b), однако для случаев (a) и (c) требуются уже три цвета; соответствующее раскрашивание показано на рис. 3.2.14. Системы уравнений, для которых используется раскрашивание по линиям, по-прежнему допускают представление в виде (3.2.28), однако теперь диагональные блоки D_i будут иметь форму

$$D_i = \begin{bmatrix} D_{i,1} & & & & \\ & D_{i,2} & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & D_{i,c_i} \end{bmatrix}, \qquad (3.2.33)$$

где c_i — количество линий i-го цвета и каждая матрица $D_{i,j}$ имеет ленточную структуру. Мы будем называть матрицу вида (3.2.28), (3.2.33) *блочной c-цветной матрицей*.

Метод SSOR

Метод SSOR можно реализовать точно так же, как метод SOR, с использованием красно-черного или многоцветного упорядочений. Однако рассматриваемый случай имеет одну любопытную особенность, позволяющую сократить вычислительные затраты. Предположим сначала, что красно-черное упорядочение используется для уравнения Пуассона или для любой другой задачи, матрица которой принимает двуцветный вид при красно-черном упорядочении. Допустим, что порядок проходов по сетке задается последовательностью

красный, черный, черный (обратный), красный (обратный), красный, ...,

которая соответствует проходу по всем точкам сетки с последующим проходом в обратном направлении и так далее. Однако в этом случае пересчет для черных точек в обратном порядке использует только значения в красных точках, которые не подвергались изменению в процессе предыдущего прохода по черным точкам. Таким образом, в случае $\omega = 1$ обратный проход по черным точкам приводит в точности к тем же значениям,

которые были вычислены на предыдущем проходе, и первая итерация симметричного метода Гаусса — Зейделя фактически сводится лишь к трем проходам:

красный, черный, красный (обратный).

Аналогичная ситуация имеет место при использовании c цветов $C_1, \ldots, C_c$. Здесь последовательность проходов имеет вид

$$C_1, \ldots, C_{c-1}, C_c, C_c, C_{c-1}, \ldots, C_1, C_1, \ldots, \qquad (3.2.34)$$

и если $\omega = 1$, то два последовательных прохода для C_c идентичны; то же самое справедливо для двух последовательных проходов, относящихся к C_1. Таким образом, общая схема вычислений, реализующих симметричный метод Гаусса — Зейделя, принимает вид

$$(C_1, \ldots, C_{c-1}, C_c, C_{c-1}, \ldots, C_1)(C_2, \ldots, C_c, \ldots, C_1), \ldots, \qquad (3.2.35)$$

и естественно будет называть отрезки последовательности проходов, выделенные скобками, симметричными итерациями Гаусса — Зейделя. По этому же образцу переписываются и все последующие итерации. Для двух цветов, то есть в случае красно-черного упорядочения, последовательность (3.2.35) принимает вид

$$(R, B, R), (B, R), (B, R), \ldots, \qquad (3.2.36)$$

где каждая пара скобок определяет вычисления, соответствующие полной итерации симметричного метода Гаусса — Зейделя. Таким образом, после того как выполнена первая итерация, симметричный метод Гаусса — Зейделя сводится в точности к методу Гаусса — Зейделя, но с порядком обработки цветов, обратным по отношению к исходному.

Если $\omega \neq 1$, то шаги C_c, C_c в (3.2.34) уже не будут идентичными из-за влияния параметра ω. Выясним, в чем заключается отличие случая $\omega \neq 1$. Пусть $\mathbf{u}_c^1$ — результат вычислений по методу SOR на первом проходе для цвета C_c, $\mathbf{u}_c^2$ — результат, полученный после второго прохода, а $\widehat{\mathbf{u}}_c^i$ — соответствующие результаты вычислений по методу Гаусса — Зейделя. Тогда

$$\mathbf{u}_c^2 = (1 - \omega)\,\mathbf{u}_c^1 + \omega \widehat{\mathbf{u}}_c^2. \qquad (3.2.37)$$

Но результат вычислений по методу Гаусса — Зейделя зависит только от значений, отвечающих остальным цветам $1, \ldots, c-1$, которые не изменялись после предыдущей итерации Гаусса — Зейделя. Поэтому $\widehat{\mathbf{u}}_c^2 = \widehat{\mathbf{u}}_c^1$, и, подставляя в (3.2.37) выражение

для $\mathbf{u}_c^1$ через $\hat{\mathbf{u}}_c^1$ и исходное значение $\mathbf{u}_c^0$, получаем

$$\mathbf{u}_c^2 = (1-\omega)\left((1-\omega)\,\mathbf{u}_c^0 + \omega\hat{\mathbf{u}}_c^1\right) + \omega\hat{\mathbf{u}}_c^1 =$$
$$= (1-\omega)^2 u_c^0 + ((1-\omega)\,\omega + \omega)\,\hat{u}_c^1 = (1-\hat{\omega})\,\mathbf{u}_c^0 + \hat{\omega}\mathbf{u}_c^1,$$

где $\hat{\omega} = \omega(2-\omega)$, так как $(1-\omega)^2 = 1-\omega(2-\omega)$. Таким образом, результат выполнения двух последовательных шагов C_c, C_c в (3.2.34) с параметром ω будет тем же самым, что и при выполнении одного прохода метода SOR с параметром $\hat{\omega}$. Отсюда следует, что можно выполнить два шага C_c, C_c при помощи одного прохода метода Гаусса — Зейделя или, что то же самое, прохода метода Якоби с последующим использованием скорректированного значения $\hat{\omega}$ параметра релаксации. То же самое будет справедливо и для последовательных шагов C_1, C_1. Это позволяет достичь значительного сокращения вычислительных затрат на одну итерацию метода SSOR.

Следует отметить, что в случае, когда исходная система уравнений может быть записана в красно-черной форме, известно, что оптимальным значением параметра ω с точки зрения скорости сходимости метода SSOR является $\omega = 1$. Однако для c-цветных матриц при $c > 2$ таких данных нет.

Прием Конрада — Валяха

Опишем теперь другой способ сокращения вычислительных затрат метода SSOR, который назовем *приемом Конрада — Валяха*. Итерацию SSOR (3.2.14) можно представить в виде

$$(D-\omega L)\,\mathbf{x}^{k+\frac{1}{2}} = (1-\omega)\,D\mathbf{x}^k + \omega\mathbf{y}^k + \omega\mathbf{b}, \quad \mathbf{y}^{k+\frac{1}{2}} = L\mathbf{x}^{k+\frac{1}{2}} + \mathbf{b}, \tag{3.2.38a}$$

$$(D-\omega U)\,\mathbf{x}^{k+1} = (1-\omega)\,D\mathbf{x}^{k+\frac{1}{2}} + \omega\mathbf{y}^{k+\frac{1}{2}}, \quad \mathbf{y}^{k+1} = U\mathbf{x}^{k+1}, \tag{3.2.38b}$$

где $\mathbf{y}^0 = U\mathbf{x}^0$. Смысл формул (3.2.38) заключается в таком выполнении итерации метода SSOR, чтобы величины $\mathbf{y}^{k+\frac{1}{2}}$ и $\mathbf{y}^{k+1}$ получались без каких-либо дополнительных затрат. Для того чтобы увидеть, как это сделать, рассмотрим покомпонентную форму записи итерации.

Пусть к началу $(k+1)$-й итерации $\mathbf{y}^k = U\mathbf{x}^k$. Тогда вычисление величин $\mathbf{x}_i^{k+\frac{1}{2}}$ задается формулами

$$\hat{x}_i^{k+\frac{1}{2}} = \frac{1}{a_{ii}}\left(\sum_{j<i} l_{ij}x_j^{k+\frac{1}{2}} + y_i^k + b_i\right),$$
$$x_i^{k+\frac{1}{2}} = x_i^k + \omega\left(\hat{x}_i^{k+\frac{1}{2}} - x_i^k\right). \tag{3.2.39}$$

В процессе этих вычислений мы заменяем значения y_i^k на

$$y_i^{k+\frac{1}{2}} = \sum_{j<i} l_{ij} x_j^{k+\frac{1}{2}} + b_i.$$

Таким образом, закончив вычисление $\mathbf{x}^{k+\frac{1}{2}}$, мы получаем также $\mathbf{y}^{k+\frac{1}{2}} = L\mathbf{x}^{k+\frac{1}{2}} + \mathbf{b}$. На следующем полушаге мы вычисляем

$$\hat{x}^{k+1} = \frac{1}{a_{ii}}\left(\sum_{j>i} u_{ij} x_j^{k+1} + y_i^{k+\frac{1}{2}}\right),$$

$$x_i^{k+1} = x_i^{k+\frac{1}{2}} + \omega\left(\hat{x}_i^{k+1} - x_i^{k+\frac{1}{2}}\right). \tag{3.2.40}$$

В процессе выполнения этого полушага мы заменяем $y_i^{k+\frac{1}{2}}$ на значения

$$y_i^{k+1} = \sum_{j>i} u_{ij} x_j^{k+1}, \tag{3.2.41}$$

так что по завершении вычисления $\mathbf{x}^{k+1}$ мы получаем также вектор $\mathbf{y}^{k+1} = U\mathbf{x}^{k+1}$, необходимый для того, чтобы начать выполнение следующей итерации. Таким образом, мы устраняем явное вычисление $L\mathbf{x}^{k+\frac{1}{2}}$ и $U\mathbf{x}^{k+1}$, за исключением самого начала итераций, когда нужно вычислить $\mathbf{y}^0 = U\mathbf{x}^0$. Применение описанного подхода к дискретному уравнению Пуассона (3.1.5) с использованием естественного и красно-черного упорядочений мы оставляем в качестве упражнения 3.2.16.

Подход к методу SOR, основанный на анализе потоков данных

Рассмотрим теперь, казалось бы, совершенно иной подход к параллельной реализации алгоритма SOR, который на самом деле тесно связан с многоцветными упорядочениями. Мы проиллюстрируем этот подход, рассмотрев сначала систему уравнений, возникающую при дискретизации какого-либо уравнения в частных производных с использованием девятиточечного сеточного шаблона, показанного на рис. 3.2.7. Будем предполагать, что точки сетки упорядочены естественным образом: слева направо и снизу вверх.

Для типичной точки сетки соответствующее итерационное приближение можно пересчитать методом SOR только тогда, когда уже получены новые значения в соседних южной, западной, юго-восточной и юго-западной точках, как показано на

рис. 3.2.15. Идея алгоритма, основанного на анализе потоков данных, заключается в пересчете значения в каждой точке сетки, как только это оказывается возможным. Иллюстрацией служит рис. 3.2.16, где номерами в точках сетки отмечены дискретные моменты времени, в которые осуществляется пересчет соответствующего значения.

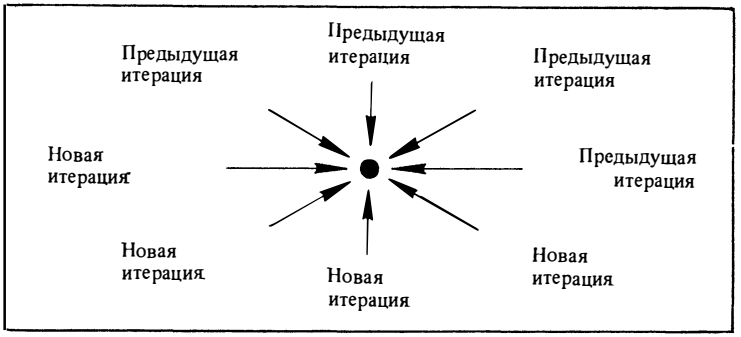


Рис. 3.2.15. Схема пересчета неизвестного значения в точке сетки

Как показано на рис. 3.2.16, на первых двух временны́х шагах пересчитываются неизвестные в точках сетки (1, 1) и (1, 2). После этого становится доступной информация, необходимая для пересчета неизвестных в точках (1, 3) и (2, 1), который

11, 15, 19	12, 16, 20	13, 17, 21	14, 18, 22	15, 19, 23
9, 13, 17	10, 14, 18	11, 15, 19	12, 16, 20	13, 17, 21
7, 11, 15	8, 12, 16	9, 13, 17	10, 14, 18	11, 15, 19
5, 9, 13	6, 10, 14	7, 11, 15	8, 12, 16	9, 13, 17
3, 7, 11	4, 8, 12	5, 9, 13	6, 10, 14	7, 11, 15
1, 5, 9	2, 6, 10	3, 7, 11	4, 8, 12	5, 9, 13

Рис. 3.2.16. Времена пересчета в точках сетки

осуществляется параллельно на третьем шаге. Аналогично, на четвертом временно́м шаге могут быть пересчитаны неизвестные в точках (1, 4) и (2, 2). Теперь для точки (1, 1) оказывается, что все предшествующие ей соседние точки уже подверглись пересчету, и поэтому неизвестное в точке (1, 1) можно пересчитать второй раз параллельно с выполнением первого пересчета других неизвестных на пятом временно́м шаге. Те-

перь становится ясным, каким образом продолжаются вычисления.

Вычисления продвигаются по сетке подобно волновым фронтам, по аналогии с потоковой реализацией LU-разложения в § 2.2. Рассмотрим одиннадцатый временно́й шаг. Неизвестные в точках сетки (1, 3) и (2, 1) пересчитываются в третий раз, неизвестные в точках сетки (4, 1), (3, 3) и (5, 2) пересчитываются во второй раз, а в точках (6, 1), (5, 3) и (4, 5) — в первый раз. Это показывает, что в любой момент времени после начала процесса некоторое число неизвестных может пересчитываться параллельно, причем вычисляемые значения будут соответствовать различным номерам итераций.

Полуитерационные методы

Введение параметра ω, позволившее получить метод SOR, было способом ускорения исходного метода Гаусса — Зейделя. Другая процедура ускорения заключается в следующем. Исходя из некоторого базисного итерационного метода (Якоби, SOR и т. п.) запишем его в виде

$$\mathbf{x}^{k+1} = H\mathbf{x}^k + \mathbf{d}, \quad k = 0, 1, \ldots. \tag{3.2.42}$$

Взяв итерационные приближения $\mathbf{x}^k$ и $\mathbf{x}^{k+1}$, попытаемся получить улучшенное приближение к искомому решению $\hat{\mathbf{x}}$ в виде линейной комбинации этих двух приближений. Определим

$$\mathbf{y}^{k+1} = \alpha_1\mathbf{x}^{k+1} + \alpha_2\mathbf{x}^k = \alpha_1(H\mathbf{x}^k + \mathbf{d}) + \alpha_2\mathbf{x}^k. \tag{3.2.43}$$

Предположим, что итерационный метод (3.2.42) является согласованным, иными словами, решение системы удовлетворяет соотношению $\hat{\mathbf{x}} = H\hat{\mathbf{x}} + \mathbf{d}$ (см. приложение 2). Желательно, чтобы метод, определенный соотношением (3.2.43), также был согласованным. Таким образом, если заменить $\mathbf{x}^k$ и $\mathbf{y}^{k+1}$ в (3.2.43) на $\hat{\mathbf{x}}$, мы приходим к условию

$$\hat{\mathbf{x}} = \alpha_1(H\hat{\mathbf{x}} + \mathbf{d}) + \alpha_2\hat{\mathbf{x}} = \alpha_1\hat{\mathbf{x}} + \alpha_2\hat{\mathbf{x}},$$

т. е. $\alpha_1 + \alpha_2 = 1$. Учитывая это ограничение на α_1 и α_2, можно переписать (3.2.43) с использованием одного параметра γ. Далее, поскольку на самом деле вычисляться будут векторы $\mathbf{y}^k$, запишем (3.2.43) в виде

$$\mathbf{y}^{k+1} = \gamma(H\mathbf{y}^k + \mathbf{d}) + (1 - \gamma)\mathbf{y}^k. \tag{3.2.44}$$

Такого рода соотношение называют экстраполяцией исходного метода (3.2.42) или *экстраполяционным методом*. На k-й итерации, исходя из приближения $\mathbf{y}^k$, мы выполняем один шаг

базисного итерационного метода, чтобы получить $H\mathbf{y}^k + \mathbf{d}$, а затем вычисляем $\mathbf{y}^{k+1}$ согласно (3.2.44).

Указанный подход, основанный на образовании линейных комбинаций итерационных приближений базисного итерационного метода, можно обобщить на любое число приближений, включаемых в линейную комбинацию. Предположим, что $\mathbf{x}^1, \ldots, \mathbf{x}^m$ вычислены в результате применения базисного итерационного метода (3.2.42), и определим

$$\mathbf{y}^{m+1} = \sum_{i=1}^{m} \alpha_{m,i}\mathbf{x}^i. \tag{3.2.45}$$

Полученный метод известен под названием *полуитерационного метода* (semi-iterative method). Если в качестве базисного итерационного метода (3.2.42) выбран метод Якоби, то соответствующий полуитерационный метод называется *полуитерационным методом Якоби*, или *методом Якоби-SI*. Аналогично можно определить методы SOR-SI и SSOR-SI.

Из согласованности метода (3.2.42) следует (см. приложение 2), что векторы погрешности $\mathbf{x}^k - \hat{\mathbf{x}}$ удовлетворяют соотношению

$$\mathbf{x}^k - \hat{\mathbf{x}} = H^k(\mathbf{x}^0 - \hat{\mathbf{x}}), \tag{3.2.46}$$

и если потребовать, чтобы метод (3.2.45) был согласованным, то должно выполняться условие

$$\hat{\mathbf{x}} = \sum_{i=1}^{m} \alpha_{m,i}\hat{\mathbf{x}},$$

или

$$\sum_{i=1}^{m} \alpha_{m,i} = 1. \tag{3.2.47}$$

Таким образом,

$$\mathbf{y}^{m+1} - \hat{\mathbf{x}} = \sum_{i=1}^{m} \alpha_{m,i}(\mathbf{x}^i - \hat{\mathbf{x}}) = \sum_{i=1}^{m} \alpha_{m,i}H^i(\mathbf{x}^0 - \hat{\mathbf{x}}) = p(H)(\mathbf{x}^0 - \hat{\mathbf{x}}), \tag{3.2.48}$$

где $p(H)$ — матричный полином вида

$$p(H) = \sum_{i=1}^{m} \alpha_{m,i}H^i. \tag{3.2.49}$$

Из-за той роли, какую играет этот полином в поведении погрешности, полуитерационные методы рассматриваемого типа известны также под названием *методов полиномиального ускорения*.

Если мы хотим выбрать скаляры $\alpha_{m,i}$ так, чтобы максимизировать скорость сходимости, нужно минимизировать спектральный радиус полинома $p(H)$ (см. приложение 2). Условие (3.2.47) эквивалентно

$$p(1) = 1. \tag{3.2.50}$$

Определим $\mathscr{P}_m$ как множество полиномов p степени m, для которых выполняется (3.2.50). Поскольку собственные значения матрицы $p(H)$ равны $p(\lambda_i)$, $i = 1, \ldots, n$, где $\lambda_1, \ldots, \lambda_n$ — собственные значения матрицы H, возникает задача минимизации

$$\min_{p \in \mathscr{P}_m} \max_{1 \leqslant i \leqslant n} |p(\lambda_i)|. \tag{3.2.51}$$

Решение задачи (3.2.51) в такой постановке представляется невозможным, так как собственные значения матрицы H известны лишь для некоторых специальных случаев; обычно мы располагаем (в лучшем случае) лишь некоторыми оценками собственных значений. В качестве примера предположим, что все собственные значения матрицы H вещественны. Этот случай имеет место для итерационных методов Якоби и SSOR, если матрица A симметрична и положительно определена (см. приложение 2), однако, вообще говоря, это не относится к методу SOR. Пусть λ_1 и λ_n — соответственно минимальное и максимальное собственные значения матрицы H, и пусть

$$-1 < a \leqslant \lambda_1 < \lambda_n \leqslant b < 1. \tag{3.2.52}$$

Тогда величина

$$\min_{p \in \mathscr{P}_m} \max_{a \leqslant \lambda \leqslant b} |p(\lambda)| \tag{3.2.53}$$

является верхней оценкой величины (3.2.51), поскольку максимум по всем собственным значениям заменен на максимум по интервалу, который содержит все эти собственные значения. Минимаксная задача (3.2.53) является классической и может быть решена с использованием полиномов Чебышёва первого рода. Эти полиномы можно определить при помощи рекуррентных соотношений

$$T_0(\lambda) = 1, \quad T_1(\lambda) = \lambda, \quad T_{i+1}(\lambda) = 2\lambda T_i(\lambda) - T_{i-1}(\lambda), \quad i = 1, \ldots, m-1, \tag{3.2.54}$$

или явного выражения

$$T_m(\lambda) = \cos(m \arccos \lambda), \quad |\lambda| \leqslant 1. \tag{3.2.55}$$

Многочлены, определенные формулами (3.2.54), дают решение задачи (3.2.53) в случае, когда $a = -1$ и $b = 1$. Для других интервалов достаточно сделать линейную замену переменной,

отображающую отрезок $[a, b]$ в отрезок $[-1, 1]$; тогда решение задачи (3.2.53) будет выражаться формулой

$$p_m(\lambda) = T_m\left(\frac{2\lambda - b - a}{b - a}\right) \Big/ T_m\left(\frac{2 - b - a}{b - a}\right). \tag{3.2.56}$$

Из (3.2.54) следует (см. упражнение 3.2.19), что полиномы p_m, определенные соотношением (3.2.56), удовлетворяют рекуррентным соотношениям

$$p_0(\lambda) = 1, \quad p_1(\lambda) = \gamma\lambda - \gamma + 1, \tag{3.2.57}$$
$$p_{i+1}(\lambda) = \rho_{i+1}(\gamma\lambda - \gamma + 1) p_i(\lambda) + (1 - \rho_{i+1}) p_{i-1}(\lambda),$$

где при $\beta = (2 - b - a)/(b - a)$

$$\gamma = \frac{2}{2 - b - a}, \quad \rho_{i+1} = \frac{2\gamma\beta p_i(\beta)}{p_{i-1}(\beta)}. \tag{3.2.58}$$

Используя (3.2.54), можно показать, что величины ρ_i также удовлетворяют рекуррентным соотношениям, имеющим следующий вид:

$$\rho_1 = 1, \quad \rho_2 = 2\beta^2/(2\beta^2 - 1), \quad \rho_{i+1} = 4\beta^2/(4\beta^2 - \rho_i),$$
$$i = 2, 3, \ldots . \tag{3.2.59}$$

В принципе можно получить коэффициенты $\alpha_{m,i}$ полинома p_m и затем использовать (3.2.45)[1]). Однако значительно более эффективным оказывается использование рекуррентных соотношений для p_m с целью получения аналогичных рекуррентных соотношений для «ускоренных» итерационных приближений $\mathbf{y}^m$, определяемых формулой (3.2.45). Из (3.2.48) и (3.2.57) при $\mathbf{e}^0 = \mathbf{x}^0 - \hat{\mathbf{x}}$ получим следующие формулы:

$$\begin{aligned}
\mathbf{y}^{m+1} - \hat{\mathbf{x}} &= p_m(H)\,\mathbf{e}^0 = \rho_m(\gamma H + (1 - \gamma) I) p_{m-1}(H)\,\mathbf{e}^0 + \\
&\quad + (1 - \rho_m) p_{m-2}(H)\,\mathbf{e}^0 = \\
&= \rho_m(\gamma H + (1 - \gamma) I)(y^m - \hat{\mathbf{x}}) + (1 - \rho_m)(\mathbf{y}^{m-1} - \hat{\mathbf{x}}) = \\
&= \rho_m(\gamma H + (1 - \gamma) I)\,\mathbf{y}^m + (1 - \rho_m)\,\mathbf{y}^{m-1} - \\
&\quad - \rho_m(\gamma H + (1 - \gamma) I)\,\hat{\mathbf{x}} - (1 - \rho_m)\,\hat{\mathbf{x}}.
\end{aligned} \tag{3.2.60}$$

Так как $H\hat{\mathbf{x}} = \hat{\mathbf{x}} - \mathbf{d}$, получаем

$$\rho_m(\gamma H + (1 - \gamma) I)\,\hat{\mathbf{x}} + (1 - \rho_m)\,\hat{\mathbf{x}} = -\rho_m\gamma\mathbf{d} + \hat{\mathbf{x}},$$

и (3.2.60) принимает вид

$$\mathbf{y}^{m+1} = \rho_m(\gamma(H\mathbf{y}^m + \mathbf{d}) + (1 - \gamma)\,\mathbf{y}^m) + (1 - \rho_m)\,\mathbf{y}^{m-1}. \tag{3.2.61}$$

[1]) Такая схема, как правило, оказывается совершенно неприемлемой для практических вычислений, так как она даже при умеренных значениях m приводит к резкому возрастанию погрешностей округления. — *Прим. перев.*

Это и есть основное рекуррентное соотношение, связывающее «ускоренные» приближения $\mathbf{y}^i$, которое носит название *чебышёвского полуитерационного метода* или *метода чебышёвского ускорения*.

Теперь вычислительную процедуру можно построить следующим образом. В начале $(m+1)$-го шага выполняется одна итерация базисного метода, исходя из приближения $\mathbf{y}^m$, после чего формируется вектор $H\mathbf{y}^m + \mathbf{d}$; затем вычисленный вектор включается в линейную комбинацию с $\mathbf{y}^m$ и $\mathbf{y}^{m-1}$, чтобы в соответствии с (3.2.61) получить $\mathbf{y}^{m+1}$. Поэтому, если вектор $H\mathbf{y}^m + \mathbf{d}$ вычислен, то для получения $\mathbf{y}^{m+1}$ достаточно умножений скаляра на вектор, а также вычисления триад. Таким образом, этот метод очень хорош для векторных и параллельных вычислений, при условии, что и сама базисная итерация также хорошо векторизуется и распараллеливается. Главной проблемой остается получение хороших оценок a и b наименьшего и наибольшего собственных значений матрицы H.

Упражнения к параграфу 3.2

1. Проверить, что (3.2.3) эквивалентно (3.2.1).

2. Убедиться, что (3.2.6) действительно является матричным представлением итерации метода SOR (3.2.4).

3. Для системы из трех уравнений с тремя неизвестными выписать все шесть возможных переупорядочений этой системы, получаемых при помощи перестановок уравнений и соответствующих неизвестных.

4. Выписать в явном виде уравнения (3.2.15) при $N = 3$ и $N = 4$.

5. Выписать схему распределения векторных регистров, соответствующую рис. 3.1.6, для красно-черного метода Гаусса — Зейделя.

6. Выполнить вычисления, соответствующие (3.2.21), чтобы получить новые значения в черных точках, исходя из значений в красных точках.

7. Предположим, что значение N четно и к сетке добавлен столбец фиктивных граничных точек, обеспечивающий корректное выполнение красно-черной итерационной процедуры. Для векторного компьютера, затрачивающего на обработку вектора длины m время $1000 + 10m$, оценить потерю эффективности, связанную с указанными дополнительными точками сетки, как функцию от N.

8. Записать уравнения (3.2.24) в матричной форме (3.2.25) с использованием красно-черного упорядочения и показать, какую структуру будут иметь матрицы D_R и D_B. Выписать уравнения в явном виде при $N = 4$.

9. Выписать в явном виде систему (3.2.26) для уравнений (3.2.24), используя четырехцветное упорядочение, показанное на рис. 3.2.8, при $N = 4$.

10. Найти раскрашивания, пригодные для шаблонов (a) и (c), показанных на рис. 3.2.9, и убедиться, что минимальное количество цветов, обеспечивающее локальное разделение неизвестных, действительно будет таким, как указано в пояснении к этому рисунку.

11. Проверить, что раскрашивания, приведенные на рис. 3 2 10, действительно подходят для шаблонов, показанных на рис. 3 2.7 и 3 2.9b соответственно

12. Найти схемы вычислений, аналогичные 3 2 31, для пересчета значений в черных, зеленых и белых точках Рассмотреть также случай, когда $N \neq 4M$, и исследовать возникающие при этом проблемы

13. Проверить, что раскрашивания, приведенные на рис. 3.2.14, действительно подходят для шаблонов (a) и (c) на рис. 3.2 9, если используется полинейный метод SOR

14. Показать, что для девятиточечного шаблона, приведенного на рис. 3 2.7, три четырехцветных упорядочения

$$\begin{matrix} G & 0 & R & B \\ R & B & G & 0 \\ G & 0 & R & B \\ R & B & G & 0 \end{matrix} \qquad \begin{matrix} 0 & B & 0 & B \\ G & R & G & R \\ B & 0 & B & 0 \\ R & G & R & G \end{matrix} \qquad \begin{matrix} G & 0 & G & 0 \\ R & B & R & B \\ G & 0 & G & 0 \\ R & B & R & B \end{matrix}$$

являются единственными (с точностью до перестановок цветов) упорядочениями, позволяющими добиться локального разделения точек шаблона.

15. Написать псевдокод итераций метода SOR с четырехцветным упорядочением, используя специальный прием, продемонстрированный на рис. 3 1.8.

16. Применить прием Конрада — Валяха при $\omega = 1$ для дискретного уравнения Пуассона (3.1 5). Сделать то же самое для уравнений с красно-черным переупорядочением

17. Рассмотреть дифференциальное уравнение $u_{xx} + u_{yy} + u_{xy} = 4$ на единичном квадрате с условием $u(x, y) = x^2 + y^2$ на границе квадрата. Показать, что точным решением этой задачи является функция $u(x, y) = x^2 + y^2$. Для дискретных уравнений (3 2.24) с $a = 1$ и с правой частью, равной $4h^2$ вместо нуля, показать, что их точным решением является функция $u_{ij} = x_i^2 + y_j^2$. (Заметим, что указанные решения в точности те же, что и для уравнения $u_{xx} + u_{yy} = 4$, см. упражнение (3 1.16).)

18 Для дискретной задачи из упражнения 17 написать псевдокод, реализующий многоцветный метод SOR Оценить время выполнения одной итерации как функцию от N.

19. Используя (3.2.54), проверить соотношения (3 2 57).

Литература и дополнения к параграфу 3.2

1. Методы Гаусса — Зейделя, SOR и SSOR, а также их блочные и полуитерационные версии являются классическими; подробное описание их математических свойств можно найти в монографиях [Varga, 1962] и [Young, 1971].

2. Релаксационный множитель можно ввести также и в схему метода Якоби, но обычно это не делается. Однако в работе [Schonauer, 1983] сообщается об обнадеживающих результатах, полученных для метода Якоби с переменной верхней релаксацией, где релаксационный параметр находится в довольно сложной зависимости от номера итерации.

3. Уже на ранних этапах исследований по параллельным вычислениям был осознан тот факт, что красно-черное упорядочение позволяет достичь

высокой степени параллелизма при решении дискретного уравнения Пуассона методом SOR (см., например, [Ericksen, 1972] и [Lambiotte, 1975]. Через несколько лет ряд исследователей независимо выдвинули идею использования более чем двух цветов для более сложных ситуаций; трактовка многоцветных упорядочений, принятая в тексте, следует работе [Adams, Ortega, 1982]. Влияние многоцветных переупорядочений на скорость сходимости остается еще не вполне ясным, хотя в работе [Adams, Jordan, 1985] показано, что в определенных ситуациях асимптотическая скорость сходимости при использовании некоторых многоцветных упорядочений совпадает с асимптотической скоростью сходимости при естественном упорядочении См. также работу [Adams et al., 1987], где проводится анализ скорости сходимости для четырехцветных упорядочений девятиточечной дискретизации уравнения Пуассона, и [Adams, 1986], где приводится дальнейшее обсуждение многоцветных упорядочений В работе [Kuo et al., 1987] анализируется метод SOR с красно-черным упорядочением для мультипроцессора с решетчатой структурой межпроцессорных связей, причем рассматривается использование релаксационных параметров ω_{ij}, зависящих от точки сетки Затраты времени на межпроцессорные обмены данными при реализации красно-черного метода SOR на вычислительной системе с локальной памятью изучаются в [Saltz et al., 1987]. Кроме того, там же показано, что проверка сходимости итераций может потребовать чрезмерных затрат времени на вычислительных системах такого типа, и предложены различные вероятностные способы проверки. Идея многоцветного раскрашивания была также использована в работе [Berger et al, 1982] для построения методов сборки конечно-элементных уравнений.

4. В работе [O'Leary, 1984] были предложены другие типы многоцветных упорядочений, в частности упорядочения вида

3 3 1 1 3 3 1 2 3 3
3 3 2 2 3 3 2 2 3 1
3 1 2 2 1 1 2 2 1 1
1 1 2 3 1 1 3 3 1 1
1 1 3 3 1 2 3 3 2 2

Здесь точки сетки группируются в блоки по пять точек в каждом (за исключением приграничных точек). Сначала упорядочиваются все точки, помеченные единицей, затем — все точки, помеченные двойкой, и, наконец, все точки, помеченные тройкой Возникающая система уравнений имеет вид (3 2.28) при $c = 3$, но теперь D_i представляют собой блочно-диагональные матрицы с блоками размера не более 5×5. Теперь можно применять итерации блочного метода SOR с проходами типа блочного метода Якоби, включающими решение систем уравнений пятого порядка (или меньше).

5. Прием Конрада — Валяха описан в работе [Conrad, Wallach, 1977] Впоследствии эти же авторы (см. [Conrad, Wallach, 1979]) распространили предложенный подход на случай «попеременных итераций» более общего вида

$$B_1\mathbf{x}^{k+\frac{1}{2}} = C_1\mathbf{x}^{k} + \mathbf{b}, \qquad B_2\mathbf{x}^{k+1} = C_2\mathbf{x}^{k+\frac{1}{2}} + \mathbf{b},$$

где $A = B_1 - C_1 = B_2 - C_2$ представляют собой такие расщепления матрицы A, что B_1 и B_2 являются треугольными матрицами.

6. Подход к распараллеливанию метода SOR, основанный на анализе потоков данных, впервые был реализован в работе [Patel, Jordan, 1984]. Впоследствии он был использован в [Adams, Jordan, 1985] в качестве инструмента анализа скорости сходимости многоцветного метода SOR.

7. При использовании метода SOR, а также других итерационных методов иногда предлагается масштабировать матрицу A перед началом итераций таким образом, чтобы диагональные элементы стали равны единице. Так, если D — диагональная часть матрицы A, то диагональные элементы матрицы $D^{-1}A$ будут равны единице. Однако такое преобразование, вообще говоря, нарушит симметрию матрицы A, и поэтому обычно для симметричных матриц A с положительными диагональными элементами применяется масштабирование вида $D^{1/2}AD^{1/2}$ Если проведено такое масштабирование, то во время выполнения итераций SOR не потребуется деления (или умножения на D^{-1}). Требуется все же некоторый анализ, позволяющий оценить, будет ли указанная экономия перевешивать затраты на выполнение масштабирования перед началом итераций. Это зависит, конечно, от требуемого числа итераций; однако для многих задач масштабирование оказывается полезным.

8. Результаты вычислений на компьютерах CDC CYBER 205 и CRAY-1 с использованием красно-черного метода SOR приводятся в работах [Young et al., 1985] и [Kincaid et al., 1986a, b]. В работе [Houstis et al., 1987] для решения системы уравнений, возникающей при дискретизации уравнения Пуассона кубическими сплайнами по методу коллокации, использовался блочный метод SOR. Рассматривались как синхронная, так и асинхронная версии; для них приводится время вычислений на векторном компьютере CYBER 205, на системе Alliant FX/8 с восемью процессорами, на системе FLEX/32 с семью процессорами, а также на Sequent Balance 21000 с 24 процессорами См также работы [Plemmons, 1986], где рассматривается параллельный блочный метод SOR в приложении к задачам анализа конструкций, и [Saad, Sameh, Saylor, 1985], где рассматриваются чебышёвские полуитерационные методы.

9. В работе [O'Leary, White, 1985] (см. также [Neumann, Plemmons, 1987] и [White, 1987]) рассматриваются мультирасщепления матрицы A вида

$$A = B_i - C_i, \quad i = 1, \ldots, k, \quad \sum_{i=1}^{k} D_i = I \quad (D_i \geqslant 0),$$

$$H = \sum_{i=1}^{k} D_i B_i^{-1} C_i, \quad \mathbf{d} = \left(\sum_{i=1}^{k} D_i B_i^{-1} \right) \mathbf{b},$$

и соответствующие итерации вида $\mathbf{x}^{k+1} = H\mathbf{x}^k + \mathbf{d}$. Многие стандартные методы, включая метод SOR, можно использовать для построения таких итераций. Это указывает еще один подход к распараллеливанию вычислений.

10. В работе [Reed, Adams, Patrick, 1987] изучается время, затрачиваемое на передачу данных, как функция от вида межпроцессорных связей, способа распределения вычислений по процессорам и вида шаблона, заданного на разностной сетке. В частности, изложение сосредоточено на треугольных, прямоугольных и шестиугольных решетках процессоров и пяти- и девятиточечных сеточных шаблонах. Утверждается, что все эти три фактора должны рассматриваться согласованно и что рассмотрение только одного или двух из них, вероятно, приведет к неоптимальным результатам Среди теоретических результатов получен следующий: для пятиточечных шаблонов наилучшими являются шестиугольные соединения процессоров, при условии, что возможна передача малых пакетов данных. Если же допустима передача только больших пакетов данных, то справедливо обратное заключение См. также работу [Patrick et al., 1987].

11. Весьма изощренное использование итераций Гаусса — Зейделя можно встретить в многосеточных методах. Предположим, что уравнение в частных производных дискретизовано на сетке, в точках которой ищутся требуемые приближенные значения решения. Назовем эту сетку *мелкой*, а подмноже-

ства ее точек будем называть *более грубыми* сетками. В многосеточном методе выполняется несколько итераций Гаусса — Зейделя на мелкой сетке, затем информация с узлов мелкой сетки «собирается» в узлы более грубой сетки, и выполняется несколько итераций Гаусса — Зейделя на этой грубой сетке. Этот процесс повторяется с использованием все более грубых сеток. Информация, полученная на этих грубых сетках, затем возвращается обратно на более мелкие сетки при помощи интерполяции. Описанный процесс включает в себя много важных деталей, имеется также много вариантов основной идеи многосеточного метода. Реализованные надлежащим образом, многосеточные методы становятся все более и более привлекательными для многих задач. Одной из ранних фундаментальных работ на эту тему является [Brandt, 1977][1]). Среди более современных работ, посвященных параллельным и векторным реализациям, можно назвать [Barkai, Brandt, 1983], [Chan, Saad, Schultz, 1987], [Gannon, van Rosendale, 1986], [Chan, Tuminaro, 1987], [Kamowitz, 1987], [Naik, Ta'asan, 1987]. В работе [McBryan, 1987] рассматривается реализация ряда методов (SOR, сопряженных градиентов, многосеточных) на компьютере Connection Machine с 32000 процессоров.

3.3. Методы минимизации

Если вещественная матрица $n \times n$ является симметричной и положительно определенной, то (см. упражнение 3.3.1) решение

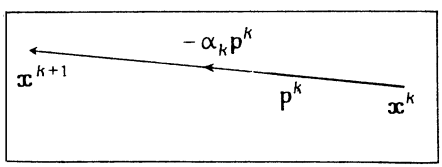


Рис. 3.3.1. Передвижение вдоль вектора направления

линейной системы $A\mathbf{x} = \mathbf{b}$ эквивалентно минимизации квадратичной функции

$$Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}. \tag{3.3.1}$$

Известно большое количество итерационных методов минимизации функций Q типа (3.3.1). Большинство таких методов имеют общий вид

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \mathbf{p}^k, \quad k = 0, 1, \ldots, \tag{3.3.2}$$

где $\mathbf{p}^k$ — *векторы направлений*, а скаляры α_k определяют расстояние, на которое осуществляется продвижение по направлению $\mathbf{p}^k$, как показано на рис. 3.3.1. В зависимости от выбора α_k и $\mathbf{p}^k$ можно получить множество различных методов.

[1]) Хороший обзор отечественных работ по этой тематике, в частности работ Н. С. Бахвалова и Р. П. Федоренко, можно найти в работе [Федоренко Р. П. Итерационные методы решения разностных эллиптических уравнений. — УМН, 1973, 28, вып. 2]. — *Прим. перев.*

Возможно, наиболее естественным способом выбора α_k представляется тот, при котором достигается минимум Q в направлении $\mathbf{p}^k$, т. е.

$$Q(\mathbf{x}^k - \alpha_k \mathbf{p}^k) = \min_{\alpha} Q(\mathbf{x}^k - \alpha \mathbf{p}^k). \tag{3.3.3}$$

Для заданных векторов $\mathbf{x}^k$ и $\mathbf{p}^k$ (3.3.3) представляет собой одномерную задачу минимизации по переменной α, которую

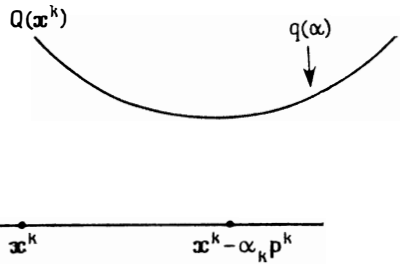


Рис. 3.3.2. Одномерная минимизация

можно решить в явном виде. Для простоты обозначений опустим индексы для $\mathbf{x}^k$ и $\mathbf{p}^k$. Тогда

$$\begin{aligned} q(\alpha) \equiv Q(\mathbf{x} - \alpha\mathbf{p}) &= \frac{1}{2}(\mathbf{x} - \alpha\mathbf{p})^{\mathrm{T}} A(\mathbf{x} - \alpha\mathbf{p}) - \mathbf{b}^{\mathrm{T}}(\mathbf{x} - \alpha\mathbf{p}) = \\ &= \frac{1}{2}\mathbf{x}^{\mathrm{T}}A\mathbf{x} - \alpha\mathbf{p}^{\mathrm{T}}A\mathbf{x} + \frac{1}{2}\alpha^2\mathbf{p}^{\mathrm{T}}A\mathbf{p} + \alpha\mathbf{p}^{\mathrm{T}}\mathbf{b} - \mathbf{b}\,\mathbf{x} = \\ &= \frac{1}{2}\mathbf{p}^{\mathrm{T}}A\mathbf{p}\alpha^2 - \mathbf{p}^{\mathrm{T}}(A\mathbf{x} - \mathbf{b})\alpha + \frac{1}{2}\mathbf{x}^{\mathrm{T}}(A\mathbf{x} - 2\mathbf{b}). \end{aligned} \tag{3.3.4}$$

Поскольку предполагается, что матрица A положительно определена, то $\mathbf{p}^{\mathrm{T}}A\mathbf{p} > 0$ и квадратный трехчлен q относительно α достигает минимума при $q'(\alpha) = 0$, т. е. при

$$\alpha_k = (\mathbf{p}^k)^{\mathrm{T}}(A\mathbf{x}^k - \mathbf{b})/(\mathbf{p}^k)^{\mathrm{T}}A\mathbf{p}^k. \tag{3.3.5}$$

Эту ситуацию иллюстрирует рис. 3.3.2.

Покоординатная релаксация [1])

Пусть $\mathbf{e}_i$ — вектор, i-я компонента которого равна единице, а все остальные — нулю. Один из простейших способов выбора векторов направлений — это циклический перебор векторов

[1]) В оригинале univariate relaxation, т. е. релаксация по одной переменной — *Прим. перев.*

$\mathbf{e}_1, \ldots, \mathbf{e}_n$:

$$\mathbf{p}^0 = \mathbf{e}_1, \quad \mathbf{p}^1 = \mathbf{e}_2, \ \ldots, \ \mathbf{p}^{n-1} = \mathbf{e}_n, \quad \mathbf{p}^n = \mathbf{e}_1, \ \ldots . \tag{3.3.6}$$

Заметим, что $\mathbf{e}_i^T A \mathbf{e}_i = a_{ii}$ и

$$\mathbf{e}_i^T (A\mathbf{x} - \mathbf{b}) = \sum_{j=1}^{n} a_{ij}\mathbf{x}_j - \mathbf{b}_i.$$

Таким образом, если $\mathbf{p}^k = \mathbf{e}_i$ и α_k выбирается по принципу минимизации (3.3.5), то следующее итерационное приближение задается формулой

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \mathbf{e}_i = \mathbf{x}^k - \frac{1}{a_{ii}}\left(\sum_{j=1}^{n} a_{ij}x_j^k - \mathbf{b}_i\right)\mathbf{e}_i. \tag{3.3.7}$$

В соотношении (3.3.7) векторы $\mathbf{x}^{k+1}$ и $\mathbf{x}^k$ различаются только значениями i-й компоненты. Действительно, (3.3.7) эквивалентно минимизации Q по i-й переменной $\mathbf{x}_i^k$ при сохранении постоянных значений всех остальных компонент $\mathbf{x}^k$.

Рассмотрим теперь первые n шагов (3.3.7) для тех компонент, значения которых изменяются, с учетом того, что $x_j^k = x_j^0$, пока не изменится j-я компонента:

$$x_i^i = x_i^0 - \alpha_i = x_i^0 - \frac{1}{a_{ii}}\left(\sum_{j=1}^{i-1} a_{ij}x_j^i + \sum_{j=i}^{n} a_{ij}x_j^0 - b_i\right) =$$

$$= \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^i - \sum_{j \geqslant i} a_{ij}x_j^0\right), \quad i = 1, \ldots, n. \tag{3.3.8}$$

Получается, что первые n шагов покоординатной релаксации образуют одну итерацию метода Гаусса — Зейделя, т. е. метод Гаусса — Зейделя оказывается эквивалентным выполнению n последовательных шагов покоординатной релаксации и объявлению полученного результата очередным итерационным приближением.

Верхняя релаксация

Для любого метода вида (3.3.2) можно в дополнение к принципу минимизации ввести релаксационный параметр ω. При этом α_k определяется как

$$\alpha_k = \omega\hat{\alpha}_k, \tag{3.3.9}$$

где $\hat{\alpha}_k$ — то значение, при котором Q минимизируется в направлении $\mathbf{p}^k$. Иллюстрацией для $\omega > 1$ служит рис. 3.3.3. Таким

образом, при $\omega \neq 1$ новое приближение $\mathbf{x}^{k+1}$ уже не минимизирует Q в направлении $\mathbf{p}^k$.

Рис. 3.3.3 показывает, что $Q(\mathbf{x}^k - \omega\hat{\alpha}_k\mathbf{p}^k) < Q(\mathbf{x}^k)$ при $\omega > 0$ до тех пор, пока не будет достигнуто значение ω, при котором $Q(\mathbf{x}^k - \omega\hat{\alpha}_k\mathbf{p}^k) = Q(\mathbf{x}^k)$. В силу симметрии квадратичной функции одной переменной относительно ее точки минимума указанное значение ω равно 2, как показано на рисунке. Таким

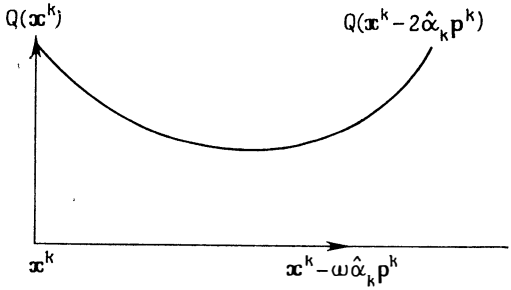


Рис 3.3.3. Верхняя релаксация

образом, $Q(\mathbf{x}^{k+1}) < Q(\mathbf{x}^k)$ при $0 < \omega < 2$, и $Q(\mathbf{x}^{k+1}) \geqslant Q(\mathbf{x}^k)$ в противном случае (см. упражнение 3.3.3).

Убывание Q при $0 < \omega < 2$ лежит в основе теоремы Островского — Райха 3.2.3 о сходимости итераций метода SOR.

Скорейший спуск

Для функции g от n переменных вектор градиента ∇g в точке $\mathbf{x}$, взятый с отрицательным знаком, определяет направление наиболее сильного локального убывания функции g в точке $\mathbf{x}$. Поэтому естественным выбором вектора направления $\mathbf{p}^k$ в алгоритме минимизации является

$$\mathbf{p}^k = \nabla Q(\mathbf{x}^k) = A\mathbf{x}^k - \mathbf{b}, \tag{3.3.10}$$

что и определяет *метод скорейшего спуска*

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k(A\mathbf{x}^k - \mathbf{b}), \tag{3.3.11}$$

известный также как *метод Ричардсона*. Заметим, что если матрица A масштабирована так, что ее диагональные элементы равны единице и $\alpha_k = 1$, то (3.3.11) сводится к итерационному методу Якоби (см. упражнение 3.3.4).

Несмотря на то что перемещение в направлении градиента, взятого с противоположным знаком, приводит к наибольшему локальному убыванию значения функции Q, метод скорейшего спуска обычно сходится очень медленно.

Методы сопряженных направлений

Весьма интересный и важный класс методов, получивших название *методов сопряженных направлений*, возникает, когда в нашем распоряжении имеются n векторов направлений $\mathbf{p}^0, \ldots, \mathbf{p}^{n-1}$, удовлетворяющих условию

$$(\mathbf{p}^i)^T A\mathbf{p}^j = 0, \quad i \neq j. \tag{3.3.12}$$

Таким образом, эти векторы ортогональны относительно скалярного произведения $(\mathbf{x}, \mathbf{y}) \equiv \mathbf{x}^T A\mathbf{y}$, определенного при помощи матрицы A; их называют также *сопряженными* относительно A.

Одно из основных свойств методов сопряженных направлений формулируется в виде следующего результата.

3.3.1. Теорема о сопряженных направлениях. *Если A — вещественная симметричная положительно определенная матрица $n \times n$, а $\mathbf{p}^0, \ldots, \mathbf{p}^{n-1}$ — ненулевые векторы, удовлетворяющие условию* (3.3.12), *то при любом начальном приближении $\mathbf{x}^0$ итерационные приближения $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \mathbf{p}^k$, где α_k выбраны в соответствии с принципом минимизации* (3.3.5), *сходятся к точному решению системы $A\mathbf{x} = \mathbf{b}$ не более чем за n шагов.*

Заметим, что теорема 3.3.1 гарантирует не только сходимость итераций, но и то, что в отсутствие ошибок округления для сходимости требуется конечное число итераций, не превосходящее n. Таким образом, методы сопряженных направлений в сущности являются прямыми методами, однако наиболее полезной оказывается их трактовка как итерационных методов, на чем мы в дальнейшем вкратце остановимся.

Чтобы убедиться в справедливости теоремы 3.3.1, рассмотрим сначала квадратичную функцию

$$Q(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^{n} a_{ii} x_i^2 - \mathbf{b}^T \mathbf{x}, \tag{3.3.13}$$

которая отвечает диагональной матрице A. Нетрудно видеть (см. упражнение 3.3.5), что методом покоординатной релаксации минимум функции (3.3.13) достигается за n шагов. Однако $\mathbf{e}_1, \ldots, \mathbf{e}_n$ образуют набор сопряженных направлений относительно диагональной матрицы, и, таким образом, в этом случае покоординатная релаксация эквивалентна методу сопряженных направлений, в котором $\mathbf{p}^{i-1} = \mathbf{e}_i$, $i = 1, \ldots, n$.

В общем случае, если P — матрица со столбцами $\mathbf{p}^0, \ldots, \mathbf{p}^{n-1}$, то соотношения (3.3.12) эквивалентны матричному тождеству $P^T AP = D$, где D — диагональная матрица. Таким

образом, выполнив замену переменных $x = P_{11}$, квадратичную форму (3.3.1) можно представить в виде

$$\frac{1}{2}(P\mathbf{y})^{\mathrm{T}} AP\mathbf{y} - \mathbf{b}^{\mathrm{T}}P\mathbf{y} = \frac{1}{2}\mathbf{y}^{\mathrm{T}}D\mathbf{y} - (P^{\mathrm{T}}\mathbf{b})^{\mathrm{T}}\mathbf{y}, \qquad (3.3.14)$$

т. е. она принимает тот же вид, что и (3.3.13), но в переменных $\mathbf{y}$. Легко убедиться в том, что метод сопряженных направлений в исходных переменных $\mathbf{x}$ эквивалентен покоординатной релаксации в новых переменных $\mathbf{y}$ (см. упражнение 3.3.6), и из приведенных выше рассуждений следует, что минимум рассматриваемой квадратичной формы будет найден не более чем за n шагов.

С другой стороны, можно дать непосредственное доказательство теоремы 3.3.1; это делается следующим образом. Заметим сначала, что

$$(A\mathbf{x}^{k+1} - \mathbf{b})^{\mathrm{T}}\mathbf{p}^j = (A\mathbf{x}^k - \alpha_k A\mathbf{p}^k - \mathbf{b})^{\mathrm{T}}\mathbf{p}^j =$$
$$= (A\mathbf{x}^k - \mathbf{b})^{\mathrm{T}}\mathbf{p}^j - \alpha_k (A\mathbf{p}^k)^{\mathrm{T}}\mathbf{p}^j.$$

Используя сопряженность направлений $\mathbf{p}^j$ и $\mathbf{p}^k$ при $j \neq k$ и определение (3.3.5) величин α_k при $j = k$, получаем

$$(A\mathbf{x}^{k+1} - \mathbf{b})^{\mathrm{T}}\mathbf{p}^j = (A\mathbf{x}^k - \mathbf{b})^{\mathrm{T}}\mathbf{p}^j, \quad j < k,$$
$$(A\mathbf{x}^{k+1} - \mathbf{b})^{\mathrm{T}}\mathbf{p}^j = 0, \quad j = k.$$

Отсюда

$$(A\mathbf{x}^n - \mathbf{b})^{\mathrm{T}}\mathbf{p}^j = (A\mathbf{x}^{n-1} - \mathbf{b})^{\mathrm{T}}\mathbf{p}^j = \ldots = (A\mathbf{x}^{j+1} - \mathbf{b})^{\mathrm{T}}\mathbf{p}^j = 0$$

для $j = 0, \ldots, n-1$ и, поскольку $\mathbf{p}^0, \ldots, \mathbf{p}^{n-1}$ линейно независимы, $A\mathbf{x}^n - \mathbf{b} = 0$. Может случиться так, что $A\mathbf{x}^m = \mathbf{b}$ для некоторого номера $m < n$; это означает, что решение уже получено на m-м шаге.

Чтобы применить метод сопряженных направлений, мы, естественно, должны располагать векторами $\mathbf{p}^j$, которые удовлетворяют условиям (3.3.12). Одним из классических множеств сопряженных векторов является набор собственных векторов матрицы A. Пусть $\mathbf{x}_1, \ldots, \mathbf{x}_n$ — ортогональные собственные векторы, отвечающие собственным значениям $\lambda_1, \ldots, \lambda_n$. Тогда

$$\mathbf{x}_i^{\mathrm{T}} A\mathbf{x}_j = \lambda_j \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j = 0, \quad i \neq j,$$

т. е. $\mathbf{x}_i$ сопряжены относительно A. Однако это не приблизит нас к практической реализации методов сопряженных направлений, поскольку отыскание всех собственных векторов матрицы A представляет собой гораздо более сложную задачу, чем решение линейной системы.

Другая возможность заключается в ортогонализации множества линейно независимых векторов $y_1, \ldots, y_n$ по отношению к скалярному произведению $(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T A\mathbf{y}$. Однако и этот подход требует чрезмерных затрат.

Метод сопряженных градиентов

Наиболее эффективный способ построения последовательности векторов сопряженных направлений для системы $A\mathbf{x} = \mathbf{b}$ — это *метод сопряженных градиентов*, в котором вектора направлений генерируются в процессе выполнения самого метода. Ниже приводится основной алгоритм, в котором используется скалярное произведение $(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$, а через $\mathbf{r}^k \equiv \mathbf{b} - A\mathbf{x}^k$ обозначается невязка на k-м шаге. Здесь формула (3.3.5) вычисления α_k заменена на другую формулу, эквивалентную (3.3.5) в контексте метода сопряженных градиентов. Правомерность такой замены обосновывается в приложении 3.

Выбрать $\mathbf{x}^0$, положить $\mathbf{p}^0 = \mathbf{r}^0$, вычислить $(\mathbf{r}^0, \mathbf{r}^0)$.

Для $k = 0, 1, \ldots$

$$\alpha_k = -(\mathbf{r}^k, \mathbf{r}^k)/(\mathbf{p}^k, A\mathbf{p}^k), \tag{3.3.15a}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k\mathbf{p}^k, \tag{3.3.15b}$$

$$\mathbf{r}^{k+1} = \mathbf{r}^k + \alpha_k A\mathbf{p}^k. \tag{3.3.15c}$$

$$\text{Если } \|\mathbf{r}^{k+1}\|_2^2 \geqslant \varepsilon, \text{ продолжать.} \tag{3.3.15d}$$

$$\beta_k = (\mathbf{r}^{k+1}, \mathbf{r}^{k+1})/(\mathbf{r}^k, \mathbf{r}^k). \tag{3.3.15e}$$

$$\mathbf{p}^{k+1} = \mathbf{r}^{k+1} + \beta_k\mathbf{p}^k. \tag{3.3.15f}$$

Формула (3.3.15c) описывает следующую невязку, так как

$$\mathbf{r}^{k+1} = \mathbf{b} - A\mathbf{x}^{k+1} = \mathbf{b} - A(\mathbf{x}^k - \alpha_k\mathbf{p}^k).$$

Поскольку вектор $A\mathbf{p}^k$ уже известен, это позволяет сэкономить вычисление $A\mathbf{x}^k$. На двух заключительных шагах происходит вычисление следующего сопряженного направления; в приложении 3 показано, что получающиеся таким образом векторы $\mathbf{p}^k$ действительно являются сопряженными. Таким образом, на основании теоремы 3.3.1 можно заключить, что в точной арифметике метод сопряженных градиентов сходится к решению системы $A\mathbf{x} = \mathbf{b}$ не более чем за n операций. Заметим, что существуют и другие эквивалентные формулировки алгоритма сопряженных градиентов; некоторые из них приведены в приложении 3.

Метод сопряженных градиентов всегда используется в сочетании с той или иной формой предобусловливания. Это будет темой следующего параграфа.

Параллельная и векторная реализация

Методы, обсуждаемые в настоящем параграфе, базируются в основном на операциях вычисления скалярного произведения и матрично-векторных умножениях вида $A\mathbf{p}$. В частности, полные вычислительные потребности одной итерации алгоритма метода сопряженных градиентов можно обрисовать следующим образом:

(3.3.15a): $A\mathbf{p}^k$, одно скалярное произведение $((\mathbf{r}^k, \mathbf{r}^k)$ уже известно), одно деление;

(3.3.15b): одна триада;

(3.3.15c): одна триада ($A\mathbf{p}^k$ уже известно);

(3.3.15d): одно скалярное произведение, одно сравнение;

(3.3.15e): одно деление $((\mathbf{r}^{k+1}, \mathbf{r}^{k+1})$ и $(\mathbf{r}^k, \mathbf{r}^k)$ уже известны);

(3.3.15f): одна триада.

Таким образом, общая потребность в вычислениях на одну итерацию составляет:

вычисление $A\mathbf{p}^k$,

два скалярных произведения,

три триады,

два скалярных деления и одно скалярное сравнение.

Условие (3.3.15d) отвечает использованию лишь одного из многих возможных способов проверки сходимости. Преимущество избранного способа (3.3.15d) заключается в том, что он почти не требует дополнительных вычислений, так как величина $(\mathbf{r}^{k+1}, \mathbf{r}^{k+1})$ все равно потребуется на следующем шаге, если сходимость еще не достигнута. Напротив, проверка, использующая величину $\|\mathbf{x}^{k+1} - \mathbf{x}^k\| = \|\alpha_k \mathbf{p}^k\|$, требует расщепления триады (3.3.15b), а также отдельного вычисления нормы.

Для любой серьезной задачи в вычислительных затратах метода сопряженных градиентов будет доминировать вычисление произведений $A\mathbf{p}^k$, и поэтому эффективная реализация этой операции является ключевым моментом. Например, если A — диагонально разреженная матрица, то для векторных машин предпочтительно хранение матрицы A по диагоналям и использование техники матрично-векторного умножения по диагоналям, описанной в § 1.3.

В тех случаях, когда матрица A имеет хотя бы несколько «постоянных» диагоналей, может быть достигнуто дополнительное сокращение затрат. В качестве крайнего случая рассмотрим матрицу (3.1.7), отвечающую дискретному уравнению Пуассона на квадрате. Если разбиение вектора $\mathbf{p}$ согласовано с разбиением матрицы A, получаем

$$A\mathbf{p} = \begin{bmatrix} T & -I & & & \\ -I & \ddots & \ddots & & \\ & \ddots & \ddots & -I & \\ & & -I & T & \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_{N-1} \\ \mathbf{p}_N \end{bmatrix} =$$

$$= \begin{bmatrix} T\mathbf{p}_1 - \mathbf{p}_2 \\ T\mathbf{p}_2 - \mathbf{p}_1 - \mathbf{p}_3 \\ \cdots\cdots\cdots \\ T\mathbf{p}_{N-1} - \mathbf{p}_{N-2} - \mathbf{p}_N \\ T\mathbf{p}_N - \mathbf{p}_{N-1} \end{bmatrix}. \qquad (3.3.16)$$

Произведения $T\mathbf{p}_i$, в свою очередь, могут быть вычислены по формуле

$$T\mathbf{p}_i = 4\mathbf{p}_i - \begin{bmatrix} 0 \\ p_{i,1} \\ \vdots \\ p_{i,N-1} \end{bmatrix} - \begin{bmatrix} p_{i,2} \\ \vdots \\ p_{i,N} \\ 0 \end{bmatrix}. \qquad (3.3.17)$$

Таким образом, вычисление рассматриваемого произведения аналогично выполнению итерации Якоби на сетке с использованием векторов длины N. Действительно, так как $A = D(I - H)$, где D — диагональная часть матрицы A, а H — матрица перехода для итераций Якоби, то A и H имеют одинаковую структуру, за исключением главной диагонали, равной нулю у матрицы H. Таким образом, умножение матриц A или H на вектор требует одного и того же подхода. В частности, выполнение операции (3.3.16) с использованием векторов длины $O(N^2)$ потребует применения той же техники, что и для метода

Якоби, а именно, перехода к обработке векторов длины $(N+2)^2$, включающих граничные позиции, или же вычисления $A\mathbf{p}$ посредством матрично-векторного умножения по диагоналям. Детали мы оставляем в качестве упражнения 3.3.7.

Приведенное выше обсуждение относится главным образом к реализации метода сопряженных градиентов на векторных компьютерах. На параллельных вычислительных системах распределение компонент вектора $\mathbf{x}^k$ по процессорам определяет также и распределение компонент векторов $\mathbf{p}^k$ и $\mathbf{r}^k$. Например, для дискретного уравнения Пуассона или более общего уравнения приписывание некоторой точки сетки какому-либо процессору приведет к тому, что компоненты векторов $\mathbf{x}^k$, $\mathbf{p}^k$ и $\mathbf{r}^k$, отвечающие этой точке сетки, будут приписаны этому же процессору. Тогда умножение $A\mathbf{p}^k$ происходит в точном соответствии с алгоритмом метода Якоби, обсуждавшимся в § 3.1, с тем отличием, что теперь роль вектора неизвестных играет вектор $\mathbf{p}^k$. Скалярные произведения вычисляются при помощи суммирования сдваиванием, и затем значение α_k рассылается всем процессорам. Операции триад для $\mathbf{x}^{k+1}$, $\mathbf{p}^{k+1}$ и $\mathbf{r}^{k+1}$ выполняются особенно эффективно, так как в каждом процессоре происходит обновление только тех компонент, которые в нем уже содержатся. Таким образом, ключевым моментом реализации метода сопряженных градиентов на параллельных машинах является процедура матрично-векторного умножения $A\mathbf{p}^k$, которая в основном зависит от структуры матрицы A. Однако даже если эту процедуру удается сделать эффективной, необходимость вычисления скалярных произведений приводит к некоторому понижению общей эффективности [1]).

Декомпозиция области

Одно из преимуществ метода сопряженных градиентов заключается в том, что он не требует явного задания матрицы A: достаточно лишь обеспечить возможность вычисления произведения $A\mathbf{p}$. Рассмотрим в качестве иллюстрации декомпозицию области, которая обсуждалась в § 2.3.

Рассмотрим систему уравнений (2.3.22) и перепишем ее, в предположении симметрии матрицы коэффициентов, с учетом

---

[1]) Потери эффективности при параллельной реализации метода сопряженных градиентов можно сократить, если использовать видоизмененные алгоритмы метода сопряженных градиентов, где, в отличие от (3.3.15), два или три скалярных произведения могут вычисляться одновременно (см., например, [Meurant, 1984]). — *Прим. перев.*

соотношения $C_i = B_i^{\mathrm{T}}$:

$$\begin{bmatrix} A_1 & & & B_1 \\ & \ddots & & \vdots \\ & & A_p & B_p \\ B_1^{\mathrm{T}} & \cdots & B_p^{\mathrm{T}} & A_S \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_p \\ \mathbf{x}_S \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_p \\ \mathbf{b}_S \end{bmatrix}. \qquad (3.3.18)$$

Чтобы выполнить шаг 2 алгоритма декомпозиции на подобласти, показанного на рис. 2.3.16, нужно решить систему $\hat{A}x_S = \hat{b}$, где

$$\hat{A} = A_S - \sum_{i=1}^{p} B_i^{\mathrm{T}} A_i^{-1} B_i, \quad \hat{b} = b_S - \sum_{i=1}^{p} B_i^{\mathrm{T}} A_i^{-1} b_i. \qquad (3.3.19)$$

Для многих задач матрицы A_i и B_i оказываются разреженными, однако это не всегда так для матриц $B_i^{\mathrm{T}} A_i^{-1} B_i$, а значит, и

Шаг 1. Формируем $\mathbf{y}_i = B_i \mathbf{p}$, $i = 1, \ldots, p$.
Шаг 2. Решаем системы $A_i \mathbf{z}_i = \mathbf{y}_i$, $i = 1, \ldots, p$.
Шаг 3. Формируем $\mathbf{w}_i = B_i^{\mathrm{T}} \mathbf{z}_i$, $i = 1, \ldots, p$.
Шаг 4. Формируем $\hat{A}\mathbf{p} = A_S \mathbf{p} - \sum_{i=1}^{p} \mathbf{w}_i$.

Рис. 3.3.4. Формирование вектора $\hat{A}\mathbf{p}$

для $\hat{A}$. Чтобы использовать разреженность исходной матрицы A, применим метод сопряженных градиентов к решению системы $\hat{A}\mathbf{x}_S = \hat{\mathbf{b}}$ без явного формирования матрицы $\hat{A}$. Напомним, что в § 2.3 было установлено, что если матрица системы (3.3.18) является симметричной и положительно определенной, то этим же свойством обладает и матрица $\hat{A}$.

Пусть $\mathbf{p}$ — какой-либо вектор направления. Тогда можно вычислить произведение $\hat{A}\mathbf{p}$, не прибегая к формированию матрицы $\hat{A}$, способом, показанным на рис. 3.3.4. Вектор $\hat{\mathbf{b}}$ строится аналогично. Решение систем на шаге 2 осуществляется с использованием разложений Холесского $A_i = L_i L_i^{\mathrm{T}}$, необходимых для выполнения остальных шагов алгоритма, показанного на рис. 2.3.16.

Напомним, что в алгоритме на рис. 2.3.16 формирование матрицы $\hat{A}$ и вектора $\hat{\mathbf{b}}$, а также решение системы $\hat{A}\mathbf{x}_S = \hat{\mathbf{b}}$ составляли как раз те фрагменты, которые не обязательно должны были хорошо распараллеливаться. Если же используется метод сопряженных градиентов для системы $\hat{A}\mathbf{x}_S = \hat{\mathbf{b}}$ с вычислением $\hat{A}\mathbf{p}$ описанным выше способом, то создается возможность глубокого распараллеливания всего алгоритма в целом. В предположении, что используются $\mathbf{p}$ процессоров, вычисления, отвечающие каждому из шагов 1—3 на рис. 3.3.4, могут быть распределены по процессорам и выполняться параллельно без обмена информацией. Только на шаге 4, так же, как и при вычислении скалярных произведений в методе сопряженных градиентов, потребуются сбор информации путем сдваивания и межпроцессорные обмены.

Упражнения к параграфу 3.3

1 Проверить, что для симметричной положительно определенной матрицы A минимум квадратичной функции (3 3 1) достигается на решении системы $A\mathbf{x} = \mathbf{b}$

2 Показать, что выражение (3 3 5) задает значение, при котором достигается минимум функции (3 3 3)

3. Проверить, что если $\hat{\alpha}_k$ — значение, на котором достигается минимум $Q\left(\mathbf{x}^k - \alpha_k \mathbf{p}^k\right)$, то $Q\left(\mathbf{x}^k - \omega\hat{\alpha}_k\mathbf{p}^k\right) < Q\left(\mathbf{x}^k\right)$ тогда и только тогда, когда $0 < \omega < 2$.

4. Показать, что если диагональные элементы матрицы A равны единице и $\alpha_k = 1$, то (3 3 11) совпадает с методом Якоби.

5. Проверить, что покоординатная релаксация минимизирует квадратичную функцию (3 3 13) за n шагов.

6 Пусть $P = [\mathbf{p}^0 \ldots \mathbf{p}^{n-1}]$, где $\mathbf{p}^i$ сопряжены относительно A. Заметим, что $P^{-1}\mathbf{p}^{i-1} = \mathbf{e}_i$, и поэтому шаг метода сопряженных направлений $\mathbf{x}^i = \mathbf{x}^{i-1} - \alpha_{i-1}\mathbf{p}^{i-1}$ в переменных $\mathbf{y} = P^{-1}\mathbf{x}$ имеет вид

$$\mathbf{y}^i = \mathbf{y}^{i-1} - \alpha_{i-1}\mathbf{e}_i.$$

Показать, что он совпадает с i-м шагом метода покоординатной релаксации (3.3.14).

7. Рассмотреть реализацию вычислений (3.3.16) с использованием векторов длины $O(N+2)^2$ путем включения граничных значений, а также реализацию с использованием векторов длины $O(N^2)$ при помощи матрично-векторного умножения по диагоналям.

Литература и дополнения к параграфу 3.3

1 Методы минимизации (для компьютеров с последовательной обработкой) обсуждаются во многих книгах. См, например, монографию [Dennis, Schnabel, 1983], где изучаются также неквадратичные задачи. Более старая книга [Forsythe, Wasow, 1960] содержит больше информации о классических

методах для квадратичных задач, таких, как покоординатная релаксация или скорейший спуск.

2. Метод сопряженных градиентов был разработан Хестенсом и Штифелем [Hestenes, Stiefel, 1952], которые изучили его основные свойства. Так как в точной арифметике метод сопряженных градиентов сходится к точному решению за n шагов, то он может рассматриваться как прямой метод, представляющий собой альтернативу, например, факторизации Холесского. Впрочем, ошибки округления нарушают указанное свойство конечной сходимости, и вскоре пришло осознание того факта, что этот метод не может конкурировать с методами факторизации для систем с плотными матрицами. Однако Рид [Reid, 1971], следуя более ранней работе [Engeli et al., 1959], показал, что для больших разреженных задач, возникающих, например, при дискретизации уравнений в частных производных, методы сопряженных градиентов имеют достаточно хорошую сходимость, причем число итераций оказывается гораздо меньше n. Это вызвало возрождение интереса к методу, особенно в случае использования предобусловливания (см. § 3.4). Аннотированная библиография статей, посвященных методу сопряженных градиентов, на период с 1948 по 1986 г. дается в [Golub, O'Leary, 1987].

3. В приложении 3 показано, что евклидовы нормы векторов погрешности $\hat{\mathbf{x}} - \mathbf{x}^k$ монотонно убывают с увеличением k. Однако указанное свойство не всегда справедливо для евклидовых норм невязок $\|\mathbf{r}^k\|_2$. В работе [Hestenes, Stiefel, 1952] приводится пример, в котором норма невязки возрастает на каждом шаге, кроме последнего! Хотя это крайний случай, вполне возможна ситуация, когда в процессе итераций наблюдаются довольно неопределенные флуктуации величины $\|\mathbf{r}^k\|_2$. Однако в норме $\|\mathbf{r}\|^2 = \mathbf{r}^T A^{-1}\mathbf{r}$ невязка убывает на каждой итерации. Это следует из соотношения

$$\mathbf{r}^T A^{-1}\mathbf{r} = (\mathbf{b} - A\mathbf{x})^T A^{-1}(\mathbf{b} - A\mathbf{x}) = \mathbf{b}^T A^{-1}\mathbf{b} - 2\mathbf{x}^T\mathbf{b} + \mathbf{x}^T A\mathbf{x},$$

показывающего, что $\mathbf{r}^T A^{-1}\mathbf{r}$ с точностью до постоянного слагаемого совпадает с той самой квадратичной формой, которая минимизируется методом сопряженных градиентов на каждой итерации.

4. Можно построить блочные алгоритмы сопряженных градиентов, которые позволяют сократить время, затрачиваемое на межпроцессорные обмены. См. дальнейшее обсуждение и описание параллельной реализации в [O'Leary, 1987].

5. Теоретические результаты, относящиеся к скорости сходимости метода сопряженных градиентов, можно найти в работах [van der Sluis, van der Vorst, 1986], [Axelsson, Lindskog, 1986].

3.4. Предобусловленный метод сопряженных градиентов

Результат, устанавливающий, что методы сопряженных направлений сходятся к точному решению не более чем за n итераций, сформулированный в предыдущем параграфе, представляет лишь теоретический интерес. Здесь мы рассмотрим метод сопряженных градиентов как итерационный метод. В этом случае оценить скорость сходимости позволяет неравенство

$$\|\mathbf{x}^k - \hat{\mathbf{x}}\|_2 \leqslant 2\sqrt{\varkappa}\,\alpha^k \|\mathbf{x}^0 - \hat{\mathbf{x}}\|_2, \qquad (3.4.1)$$

где $\hat{\mathbf{x}}$ — точное решение системы,

$$\alpha = (\sqrt{\varkappa} - 1)/(\sqrt{\varkappa} + 1) \tag{3.4.2}$$

и

$$\varkappa = \operatorname{cond}(A) = \| A \|_2 \| A^{-1} \|_2 = \lambda_n/\lambda_1. \tag{3.4.3}$$

В (3.4.3) величина $\varkappa$ есть не что иное, как число обусловленности матрицы A в норме l_2, причем предполагается, что A — симметричная положительно определенная матрица с собственными значениями $\lambda_n \geqslant \ldots \geqslant \lambda_1 > 0$. Доказательство оценки 3.4.1 можно найти в приложении 3.

Заметим, что $\alpha = 0$ при $\varkappa = 1$ [1]) и $\alpha \to 1$ при $\varkappa \to \infty$. Поэтому чем больше будет значение $\varkappa$, тем ближе будет величина α к единице, и тем сильнее замедлится скорость сходимости. Это наблюдение подводит нас к общему понятию *предобусловливания* матрицы A посредством преобразования конгруэнтности

$$\hat{A} = SAS^{\mathrm{T}}, \tag{3.4.4}$$

где S — невырожденная матрица, выбранная таким образом, что $\operatorname{cond}(\hat{A}) < \operatorname{cond}(A)$. Тогда система, которую нужно будет решать, запишется в виде

$$\hat{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}, \tag{3.4.5}$$

где $\hat{\mathbf{x}} = S^{-\mathrm{T}}\mathbf{x}$ и $\hat{\mathbf{b}} = S\mathbf{b}$.

В принципе мы можем получить идеально обусловленную матрицу, выбрав $S = A^{-1/2}$: при этом $\hat{A} = I$ и система (3.4.5) становится тривиальной; но, конечно же, такой подход лишен практической ценности [2]).

Простейшее предобусловливание задается выбором диагональной матрицы $S = D$; к сожалению, это обычно не дает удовлетворительного уменьшения числа обусловленности, и в дальнейшем мы изучим некоторые другие типы матриц-предобусловливателей. В отличие от случая, когда применяются диагональные матрицы, при использовании более сложных пред-

---

[1]) Если для симметричной положительно определенной матрицы выполняется $\operatorname{cond}(A) = 1$, то такая матрица является скалярной, т. е. матрицей вида γI, $\gamma > 0$. — *Прим. ред.*

[2]) Приведенный пример представляется несколько нарочитым, т. к. для того чтобы получить $\hat{A} = I$, достаточно взять $S = L^{-1}P$, где $LL^{\mathrm{T}} = PAP^{\mathrm{T}}$ — разреженное разложение Холесского переупорядоченной матрицы, что соответствует применению прямого метода. Использование же итерационных методов целесообразно в том случае, когда затраты на реализацию прямых методов оказываются чрезмерными. — *Прим. перев.*

обусловливателей наблюдается тенденция к нарушению структуры разреженности матрицы A. По этой причине, а также с целью устранения вычислений, требуемых для выполнения преобразования конгруэнтности (3.4.4) матрицы A, мы сформулируем метод сопряженных градиентов таким образом, чтобы работать с исходной матрицей A, хотя последовательность итерационных приближений будет при этом соответствовать применению метода сопряженных градиентов к системе (3.4.5). Чтобы увидеть, как это можно сделать, запишем сначала метод сопряженных градиентов (3.3.15) для предобусловленной системы (3.4.5):

Выбрать $\hat{\mathbf{x}}^0$; положить $\hat{\mathbf{p}}^0 = \hat{\mathbf{r}}^0$.

Для $k = 0, 1, \ldots$

$$\hat{\alpha}_k = -(\hat{\mathbf{r}}^k, \hat{\mathbf{r}}^k)/(\hat{\mathbf{p}}^k, A\hat{\mathbf{p}}^k), \tag{3.4.6a}$$

$$\hat{\mathbf{x}}^{k+1} = \hat{\mathbf{x}}^k - \hat{\alpha}_k \hat{\mathbf{p}}^k, \tag{3.4.6b}$$

$$\hat{\mathbf{r}}^{k+1} = \hat{\mathbf{r}}^k + \hat{\alpha}_k \hat{A}\hat{\mathbf{p}}^k, \tag{3.4.6c}$$

$$\hat{\beta}_k = (\hat{\mathbf{r}}^{k+1}, \hat{\mathbf{r}}^{k+1})/(\hat{\mathbf{r}}^k, \hat{\mathbf{r}}^k), \tag{3.4.6d}$$

$$\hat{\mathbf{p}}^{k+1} = \hat{\mathbf{r}}^{k+1} + \hat{\beta}_k \hat{\mathbf{p}}^k. \tag{3.4.6e}$$

Векторы $\mathbf{x}^k = S^T\hat{\mathbf{x}}^k$ — это итерационные приближения метода сопряженных градиентов, выраженные через исходные переменные. Мы хотим найти алгоритм получения $\mathbf{x}^k$, соответствующий 3.4.6. Для этого достаточно проанализировать первый шаг ($k = 0$) алгоритма (3.4.6). Имеем

$$\hat{\mathbf{r}}^0 = \hat{\mathbf{b}} - \hat{A}\hat{\mathbf{x}}^0 = S\mathbf{b} - SAS^TS^{-T}\mathbf{x}^0 = S\mathbf{r}^0.$$

Вектор $\hat{\mathbf{p}}^0$ удовлетворяет соотношениями $\hat{\mathbf{p}}^0 = S^{-T}\mathbf{p}^0$ и $\hat{\mathbf{p}}^0 = \hat{\mathbf{r}}^0$. Отсюда получаем

$$(\hat{\mathbf{p}}^0, \hat{A}\hat{\mathbf{p}}^0) = (S^{-T}\mathbf{p}^0)^T SAS^TS^{-T}\mathbf{p}^0 = (\mathbf{p}^0)^T A\mathbf{p}^0$$

и

$$(\hat{\mathbf{r}}^0, \hat{\mathbf{r}}^0) = (S\mathbf{r}^0)^T(S\mathbf{r}^0) = (S^TS\mathbf{r}^0)^T\mathbf{r}^0.$$

Последнее соотношение наводит на мысль о введении новой величины

$$\tilde{\mathbf{r}}^0 = S^TS\mathbf{r}^0, \tag{3.4.7}$$

и тогда

$$\hat{\alpha}_0 = -(\tilde{\mathbf{r}}^0, \mathbf{r}^0)/(\mathbf{p}^0, A\mathbf{p}^0).$$

Умножая (3.4.6b) на S^T и (3.4.6c) на S^{-1}, получаем

$$\mathbf{x}^1 = \mathbf{x}^0 - \hat{\alpha}_0 \mathbf{p}^0$$

и

$$\mathbf{r}^1 = \mathbf{r}^0 + \hat{\alpha}_0 S^{-1} S A S^T S^{-T} \mathbf{p}^0 = \mathbf{r}^0 + \hat{\alpha}_0 A \mathbf{p}^0.$$

Для (3.4.6d) замечаем, что

$$(\hat{\mathbf{r}}^i, \hat{\mathbf{r}}^i) = (S\mathbf{r}^i)^T S\mathbf{r}^i = (\tilde{\mathbf{r}}^i)^T \mathbf{r}^i, \quad i = 0, 1,$$

и, таким образом,

$$\hat{\beta}_0 = (\tilde{\mathbf{r}}^1, \mathbf{r}^1)/(\tilde{\mathbf{r}}^0, \mathbf{r}^0).$$

Наконец, умножая (3.4.6e) на S^T, получаем

$$\mathbf{p}^1 = S^T S \mathbf{r}^1 + \hat{\beta}_0 \mathbf{p}^0 = \tilde{\mathbf{r}}^1 + \hat{\beta}_0 \mathbf{p}^0.$$

На основе этих формул можно переписать алгоритм метода сопряженных градиентов с использованием матрицы A. Однако прежде чем явно сформулировать алгоритм, остановимся на вычислении вспомогательного вектора $\tilde{\mathbf{r}}$, определенного в (3.4.7). Если нам известна матрица-предобусловливатель S или матрица S^TS, то можно получить $\tilde{\mathbf{r}}$, выполняя указанное матрично-векторное умножение. Но наш подход к построению предобусловливания основан на использовании симметричной положительно определенной матрицы M, аппроксимирующей матрицу A, и матрица S фигурирует лишь неявным образом в соответствии с формулой

$$(S^T S)^{-1} = M. \tag{3.4.8}$$

Поскольку обычно нам известна матрица M, но не M^{-1}, вспомогательный вектор (3.4.7) находится путем решения линейной системы $M\tilde{\mathbf{r}} = \mathbf{r}$. Сформулируем теперь, в рамках указанных соглашений, *предобусловленный метод сопряженных градиентов* PCG (preconditioned conjugate gradient method):

Выбрать $\mathbf{x}^0$; положить $\mathbf{r}^0 = \mathbf{b} - A\mathbf{x}^0$.

Решить систему $M\tilde{\mathbf{r}}^0 = \mathbf{r}^0$; положить $\mathbf{p}^0 = \tilde{\mathbf{r}}^0$.

Для $k = 0, 1, \ldots$

$$\alpha_k = -(\tilde{\mathbf{r}}^k, \mathbf{r}^k)/(\mathbf{p}^k, A\mathbf{p}^k), \tag{3.4.9a}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \mathbf{p}^k, \tag{3.4.9b}$$

$$\mathbf{r}^{k+1} = \mathbf{r}^k + \alpha_k A\mathbf{p}^k. \tag{3.4.9c}$$

Проверить сходимость итераций. (3.4.9d)

Решить систему $M\tilde{\mathbf{r}}^{k+1} = \mathbf{r}^{k+1}$. (3.4.9e)

$\beta_k = (\tilde{\mathbf{r}}^{k+1}, \mathbf{r}^{k+1})/(\tilde{\mathbf{r}}^k, \mathbf{r}^k)$. (3.4.9f)

$\mathbf{p}^{k+1} = \tilde{\mathbf{r}}^{k+1} + \beta_k \mathbf{p}^k$. (3.4.9g)

В приведенном алгоритме мы опустили крышку, которой помечались ранее скаляры α_k и β_k. Заметим, что если $M = I$, то алгоритм предобусловленного метода сопряженных градиентов сводится к исходному алгоритму (3.3.15).

Поскольку вспомогательную систему $M\tilde{\mathbf{r}} = \mathbf{r}$ придется решать на каждой итерации метода, существенным является требование простоты решения этой системы. С другой стороны для того, чтобы предобусловливание было эффективным, желательно, чтобы матрица M хорошо аппроксимировала матрицу A. Ясно, что два указанных требования вступают в противоречие, поскольку чем ближе матрица M к матрице A, тем более вероятно, что решение системы $M\tilde{\mathbf{r}} = \mathbf{r}$ окажется почти таким же трудным, как решение системы $\Lambda\mathbf{x} = \mathbf{b}$.

Уточним теперь, как мы понимаем аппроксимацию матрицы A матрицей M. В силу (3.4.4) и (3.4.8) предобусловленная матрица $\hat{A}$ удовлетворяет соотношению

$$S^{\mathrm{T}} \hat{A} S^{\mathrm{T}} = S^{\mathrm{T}} S \Lambda = M^{-1} A. \tag{3.4.10}$$

Отсюда видно, что матрица $M^{-1}\Lambda$ подобна матрице $\hat{A}$, и поэтому число обусловленности предобусловленной матрицы равно отношению наибольшего собственного значения матрицы $M^{-1}A$ к ее наименьшему собственному значению:

$$\mathrm{cond}\,(\hat{A}) = \lambda_{\max}(M^{-1}\Lambda)/\lambda_{\min}(M^{-1}\Lambda). \tag{3.4.11}$$

Таким образом, поскольку нашей целью является уменьшение числа обусловленности матрицы $\hat{A}$ настолько, насколько это возможно, критерием близости матрицы M к A будет служить малость отношения наибольшего собственного значения матрицы $M^{-1}A$ к наименьшему. В принципе можно получить $\mathrm{cond}\,(\hat{A}) = 1$, выбирая $M = A$, но, конечно, такой выбор неприемлем с практической точки зрения. В дальнейшем мы рассмотрим ряд подходов к построению матрицы M, включая усеченные разложения в ряд и неполные факторизации, но сначала кратко обсудим вопросы, связанные с выбором способа проверки сходимости итераций.

Если использовать тот же самый способ проверки, что и для алгоритма метода сопряженных градиентов (3.3.15), основанный

на критерии $\|\mathbf{r}^{k+1}\|_2^2 < \varepsilon$, то придется специально вычислять скалярное произведение $(\mathbf{r}^{k+1}, \mathbf{r}^{k+1})$, так как теперь оно в алгоритме не фигурирует. Вместо него появляется величина $(\tilde{\mathbf{r}}^{k+1}, \mathbf{r}^{k+1}) = (M^{-1}\mathbf{r}^{k+1}, \mathbf{r}^{k+1})$, которая представляет собой скалярное произведение, определенное при помощи матрицы M^{-1} (см. приложение 4). В принципе возможно использование этой величины для проверки сходимости, однако при этом условие сходимости может оказаться выполненным, в то время как норма $\|\mathbf{r}^{k+1}\|_2$ еще не достигает желаемой малости. Поэтому разумной предосторожностью будет выполнение вспомогательной проверки значения $\|\mathbf{r}^{k+1}\|_2$ в том случае, если другая проверка дала положительный результат. Тогда (3.4.9d) принимает следующий вид:

> Если $(\tilde{\mathbf{r}}^{k+1}, \mathbf{r}^{k+1}) \geqslant \varepsilon$, то продолжать вычисления; в противном случае, если $(\mathbf{r}^{k+1}, \mathbf{r}^{k+1}) \geqslant \varepsilon$, то продолжать вычисления. (3.4.12)

Такая проверка сходимости должна осуществляться после выполнения этапа (3.4.9e), на котором вычисляется $\tilde{\mathbf{r}}^{k+1}$. Заметим, что использование величины $(\tilde{\mathbf{r}}^{k+1}, \mathbf{r}^{k+1})$ вместо $(\mathbf{r}^{k+1}, \mathbf{r}^{k+1})$ может привести к выполнению большего числа итераций, однако обычно этого не происходит.

Предобусловливание при помощи усеченных рядов

Рассмотрим теперь общий подход к предобусловливанию, основанный на представлении матрицы A^{-1} в виде ряда.

3.4.1. Теорема. *Пусть A — невырожденная матрица $n \times n$, а $A = P - Q$ — такое расщепление матрицы A, что матрица P невырожденна и наибольший из модулей собственных значений матрицы $P^{-1}Q$ не превосходит единицы. Тогда*

$$A^{-1} = \left(\sum_{k=0}^{\infty} H^k\right) P^{-1}, \tag{3.4.13}$$

где $H = P^{-1}Q$.

Доказательство. В силу разложения Неймана (см. приложение 4) ряд, включающий матрицу H, сходится и его сумма равна $(I - H)^{-1}$. Но $A = P - Q = P(I - H)$, откуда и следует требуемый результат. □

Основываясь на разложении (3.4.13), мы можем рассматривать матрицы

$$M = P(I + H + \ldots + H^{m-1})^{-1},$$
$$M^{-1} = (I + H + \ldots + H^{m-1})P^{-1} \tag{3.4.14}$$

как аппроксимации матриц A и A^{-1} соответственно. Таким образом, решение вспомогательной системы $M\tilde{\mathbf{r}} = \mathbf{r}$ можно найти с использованием соотношения

$$\tilde{\mathbf{r}} = M^{-1}\mathbf{r} = (I + H + \ldots + H^{m-1})P^{-1}\mathbf{r}. \tag{3.4.15}$$

Для этого не нужно явно формировать матрицы H и указанный усеченный ряд. Вместо этого заметим (см. упражнение 3.4.1), что вектор $\tilde{\mathbf{r}}$, заданный (3.4.15), является результатом m шагов итерационного метода

$$P\mathbf{r}^{i+1} = Q\mathbf{r}^i + \mathbf{r}, \quad i = 0, 1, \ldots, m-1, \quad \mathbf{r}^0 = 0, \tag{3.4.16}$$

и положим $\tilde{\mathbf{r}} = \mathbf{r}^m$. Мы будем называть (3.4.16) *вспомогательным итерационным методом.*

Приведем теперь некоторые конкретные предобусловливатели, полученные на основе такого подхода. Для поточечного метода Якоби матрица P равна диагональной части матрицы A и (3.4.16) сводится к выполнению m итераций метода Якоби. В результате получается *m-шаговый метод Якоби — PCG*, где этап (3.4.9e) формулируется следующим образом:

Выполнить m итераций метода Якоби для системы $A\mathbf{r} = \mathbf{r}^{k+1}$ и полученное итерационное приближение принять за $\tilde{\mathbf{r}}^{k+1}$. (3.4.17)

В качестве начального приближения в (3.4.17) берется нулевой вектор, как и указано в (3.4.16). Аналогично получается решение системы $M\tilde{\mathbf{r}}^0 = \mathbf{r}^0$

Формально мы могли бы использовать в качестве (3.4.16) методы Гаусса — Зейделя и SOR. Однако эти методы не позволяют получить симметричную положительно определенную матрицу M, как того требует теория. Действительно, для метода Гаусса — Зейделя при $m = 1$ получается, что $M = P = D - L$, где D — диагональная часть матрицы A, а $-L$ — ее строго нижняя треугольная часть. Поэтому матрица M несимметрична для $m = 1$, и это свойство сохраняется для всех $m > 1$, а также для метода SOR (см. упражнение 3.4.2). Указанную трудность можно преодолеть за счет использования метода SSOR; выполняя на каждой итерации m шагов метода SSOR, мы получаем

m-шаговый метод SSOR-PCG. В этом случае итерации Якоби в (3.4.17) заменяются на итерации SSOR.

Нужно проверить, что использование метода SSOR действительно дает симметричную положительно определенную матрицу M. Это вытекает из следующей теоремы, которая справедлива и для предобусловливателей, основанных на методе Якоби, и др. Ее утверждение формулируется с использованием матрицы

$$R = (I + H + \ldots + H^{m-1})P^{-1}, \tag{3.4.18}$$

которая корректно определена, если существует матрица P^{-1}. Заметим, что $M = R^{-1}$, если матрица R невырожденна.

3.4.2. Теорема. *Пусть матрица $A = P - Q$ симметрична и положительно определена, а матрица P симметрична и невырожденна. При $H = P^{-1}Q$ матрица R в* (3.4.18) *симметрична и для любого $m \geqslant 1$*

1) *если m нечетно, то матрица R положительно определена тогда и только тогда, когда положительно определена матрица P;*
2) *если m четно, то матрица R положительно определена тогда и только тогда, когда положительно определена матрица $P + Q$.*

Доказательство. Поскольку матрицы A и P симметричны, то Q также симметрична, и нетрудно убедиться (см. упражнение 3.4.3), что каждое слагаемое в сумме

$$R = P^{-1} + P^{-1}QP^{-1} + P^{-1}QP^{-1}Q^{-1}P^{-1} + \ldots + P^{-1}Q \ldots P^{-1}QP^{-1}$$

также представляет собой симметричную матрицу. Таким образом, матрица R симметрична.

Поскольку $H = P^{-1}(P - A) = I - P^{-1}A$ и матрица A положительно определена, результат, справедливый для собственных значений произведения симметричных матриц (см. теорему П.2.7 в приложении 2), гарантирует, что собственные значения матрицы $P^{-1}A$, а значит, и матрицы H, являются вещественными. Пусть

$$T = I + H + \ldots + H^{m-1},$$

и пусть λ — какое-либо собственное значение матрицы H. Тогда соответствующее собственное значение матрицы T равно

$$1 + \lambda + \ldots + \lambda^{m-1} = \begin{cases} (1 - \lambda^m)/(1 - \lambda), & \lambda \neq 1, \\ m, & \lambda = 1, \end{cases}$$

причем указанная дробь положительна при нечетном m [1]). Таким образом, при нечетном m все собственные значения матрицы T положительны. Предположим теперь, что матрица P положительно определена. Тогда в силу соотношения $T = RP$ из теоремы П.2.7, приведенной в приложении 2, следует, что матрица R положительно определена. Обратно, если R положительно определена, то в силу теоремы П.2.7 матрица P также положительно определена, что и завершает рассмотрение случая нечетного m.

Предположим теперь, что m четно. Так как $P + PH = P + P(P^{-1}Q) = P + Q$, получаем

$$R = P^{-1}(P + PH + PH^2 + PH^3 + \ldots + PH^{m-1})P^{-1} =$$
$$= P^{-1}((P + PH) + (P + PH)H^2 + \ldots + (P + PH)H^{m-2})P^{-1} =$$
$$= P^{-1}(P + Q)(I + H^2 + \ldots + H^{m-2})P^{-1}.$$

Определяя теперь $T = I + H^2 + \ldots + H^{m-2}$, мы видим, что собственные значения матрицы T представляют собой суммы только четных степеней собственных значений матрицы H и поэтому являются положительными. Положим $\hat{R} = PRP = (P + Q)T$. Если матрица $P + Q$ положительно определена, то $T = (P + Q)^{-1}\hat{R}$ и в силу теоремы П.2.7 матрица $\hat{R}$ также положительно определена. Обратно, если $\hat{R}$ положительно определена, то матрица $P + Q$ невырожденна и из теоремы П.2.7 вытекает положительная определенность матрицы $P + Q$. Но матрица $\hat{R}$ конгруэнтна матрице R и, таким образом, положительно определена тогда и только тогда, когда положительно определена матрица R. □

Обсудим теперь странное на первый взгляд условие теоремы 3.4.2, требующее, чтобы была положительно определена матрица $P + Q$. Для метода Якоби $P = D$, где D — диагональная часть матрицы A, и $Q = D - A$. Тогда требование положительной определенности матрицы $P + Q$ есть не что иное, как необходимое и достаточное условие сходимости итераций метода Якоби, сформулированное в теореме 3.1.2. Указанная теорема о сходимости метода Якоби является частным случаем следующей более общей теоремы о сходимости итерационного метода, определенного расщеплением $A = P - Q$. Напомним (см. приложение 2), что необходимым и достаточным условием сходимости соответствующих итераций $\mathbf{x}^{k+1} = P^{-1}Q\mathbf{x}^k + \mathbf{d}$ при любом начальном приближении $\mathbf{x}^0$ является ограничение на спектральный

[1]) При $\lambda \geqslant 0$ это очевидно, а при $\lambda < 0$ достаточно заметить, что рассматриваемая дробь равна $(1 + (-\lambda)^m)/(1 + (-\lambda))$. — *Прим. перев.*

радиус вида $\rho(P^{-1}Q) < 1$. Доказательство теоремы о сходимости можно найти в приложении 2.

3.4.3. Теорема. *Пусть $A = P - Q$, и пусть матрицы A и P симметричны и положительно определены. Тогда $\rho(P^{-1}Q) < 1$ в том и только в том случае, если матрица $P + Q$ положительно определена.*

В теореме 3.4.2 было установлено, что матрица R в (3.4.18) положительно определена для всех m тогда и только тогда, когда обе матрицы P и $P + Q$ положительно определены. Поэтому, основываясь на утверждении теоремы 3.4.3, мы можем переформулировать теорему 3.4.2 следующим образом.

3.4.4. Теорема. *Пусть матрица $A = P - Q$ симметрична и положительно определена, а матрица P симметрична и невырожденна. Тогда при $H = P^{-1}Q$ матрица R в (3.4.18) симметрична и положительно определена для всех $m \geqslant 1$ в том и только в том случае, если матрица P положительно определена и $\rho(H) < 1$.*

Как отмечалось выше, для того, чтобы R была симметричной для всех m, должна быть симметричной и матрица P. Тогда в силу теоремы 3.4.4 для положительной определенности матрицы M требуется, чтобы положительно определенной была матрица P, а также чтобы выполнялось условие $\rho(H) < 1$. Последнее условие влечет за собой сходимость итерационного метода, определенного расщеплением $P - Q$. В частности, поскольку метод Якоби может не сходиться для положительно определенной матрицы A, соответствующая матрица M не всегда будет знакоопределенной для четных m.

С другой стороны, для метода SSOR мы можем показать, предполагая, как и ранее, что матрица A симметрична и положительно определена, что соответствующая матрица M всегда будет положительно определенной. Если $A = D - L - L^T$, то, как показано в приложении 2, матрицы P и Q для итераций SSOR имеют следующий вид:

$$P = \frac{1}{\omega(2-\omega)}(D - \omega L)\,D^{-1}(D - \omega L^T),$$
$$Q = \frac{1}{\omega(2-\omega)}((1-\omega)D + \omega L)\,D^{-1}((1-\omega)D + \omega L^T). \tag{3.4.19}$$

Поскольку матрица D положительно определена, матрицы P и Q будут положительно определенными при любом $0 < \omega < 2$, за исключением $\omega = 1$, когда матрица Q будет лишь положительно полуопределенной (см. упражнение 3.4.5). В любом слу-

чае обеспечивается положительная определенность матрицы $P+Q$ и в силу теоремы 3.4.2 справедлив следующий результат.

3.4.5. Теорема. *Если матрица A симметрична и положительно определена и $0<\omega<2$, то матрицы R в (3.4.18) и M в (3.4.14), полученные с помощью метода SSOR, положительно определены для всех m.*

Изучим теперь влияние предобусловливаний типа (3.4.14) на скорость сходимости. С помощью (3.4.11) мы можем оценить $\operatorname{cond}(\hat{A})$, если получим оценки наибольшего и наименьшего собственных значений матрицы $M^{-1}A$. Учитывая (3.4.14) и подставляя вместо матрицы A ее расщепление $P-Q$, получаем

$$M^{-1}A=(I+H+\ldots+H^{m-1})P^{-1}(P-Q)=$$
$$=(I+H+\ldots+H^{m-1})(I-H)=I-H^m. \quad (3.4.20)$$

Если $\lambda_1, \ldots, \lambda_n$ — собственные значения матрицы H, то отсюда следует, что собственными значениями матрицы $M^{-1}A$ будут $1-\lambda_i^m$, $i=1, \ldots, n$. В дальнейшем мы будем предполагать, что матрица P симметрична и положительно определена. Тогда, как и в доказательстве теоремы 3.4.2, собственные значения матрицы H вещественны. Будем предполагать, что они упорядочены по возрастанию: $\lambda_1 \leqslant \ldots \leqslant \lambda_n$. Если ввести обозначение $\delta=\min|\lambda_i|$, то нетрудно показать (упражнение 3.4.6), что

$$\operatorname{cond}(\hat{A})=\frac{\lambda_{\max}(M^{-1}A)}{\lambda_{\min}(M^{-1}A)}=$$
$$=\begin{cases}\dfrac{1-\lambda_1^m}{1-\lambda_n^m}, & \text{если } \lambda_1\geqslant 0 \text{ или } \lambda_1<0 \text{ и } m \text{ нечетно};\\ \dfrac{1-\delta^m}{1-\lambda_n^m}, & \text{если } \lambda_1<0,\ |\lambda_n|\geqslant|\lambda_1| \text{ и } m \text{ четно};\\ \dfrac{1-\delta^m}{1-\lambda_1^m}, & \text{если } \lambda_1<0,\ |\lambda_1|\geqslant|\lambda_n| \text{ и } m \text{ четно}.\end{cases} \quad (3.4.21)$$

Для предобусловливания по методу SSOR нам потребуется только первый случай, указанный в (3.4.21).

3.4.6. Теорема. *Пусть A — симметричная положительно определенная матрица и $0<\omega<2$. Пусть $\lambda_1 \leqslant \ldots \leqslant \lambda_n$ — собственные значения матрицы перехода H метода SSOR. Тогда $\lambda_1 \geqslant 0$ и для m-шагового предобусловливания по методу SSOR*

число обусловленности матрицы $\hat{A}$ задается формулой

$$\operatorname{cond}(\hat{A}) = \frac{1-\lambda_1^m}{1-\lambda_n^m} \tag{3.4.22}$$

и является строго убывающей функцией от m.

Доказательство. Как уже отмечалось, при $\omega \neq 1$ матрицы P и Q в SSOR положительно определены. В этом случае по теореме П.2.7 собственные значения матрицы H положительны. Если же $\omega = 1$, то матрица Q лишь положительно полуопределена и собственные значения матрицы H неотрицательны (упражнение 3.4.8). В любом случае $\lambda_1 \geqslant 0$, и тогда (3.4.22) вытекает из первого соотношения (3.4.21). Чтобы показать, что $\operatorname{cond}(\hat{A})$ представляет собой строго убывающую функцию от m, заметим, что в силу теоремы П.2.11 итерации SSOR сходятся, откуда следует $\lambda_n < 1$. Таким образом, получаем

$$\frac{1-\lambda_1^{m+1}}{1-\lambda_n^{m+1}} = \frac{(1-\lambda_1)(1+\ \ldots\ +\lambda_1^m)}{(1-\lambda_n)(1+\ \ldots\ +\lambda_n^m)} < \\ < \frac{(1-\lambda_1)(1+\ \ldots\ +\lambda_1^{m-1})}{(1-\lambda_n)(1+\ \ldots\ +\lambda_n^{m-1})} = \frac{1-\lambda_1^m}{1-\lambda_n^m}. \tag{3.4.23}$$

Доказательство неравенства мы оставим в качестве упражнения 3.4.7. □

Рассуждения, послужившие для доказательства теоремы 3.4.6, позволяют убедиться в том, что величина $\operatorname{cond}(\hat{A})$ является убывающей функцией от m для любого итерационного метода, для которого $0 \leqslant \lambda_1 \leqslant \ \ldots\ \leqslant \lambda^n < 1$, где λ_i — собственные значения его матрицы перехода H. В противном случае поведение $\operatorname{cond}(\hat{A})$ может быть другим, как показывает следующий пример применения m-шагового предобусловливания по методу Якоби к дискретному уравнению Пуассона (3.1.5). Собственные значения матрицы перехода метода Якоби равны

$$\frac{1}{2}\left(\cos\frac{j\pi}{N+1} + \cos\frac{k\pi}{N+1}\right), \quad j,\ k = 1,\ \ldots,\ N$$

(упражнение 3.1.21).

В частности, наибольшее и наименьшее собственные значения удовлетворяют соотношениям

$$\lambda_n = \cos\frac{\pi}{N+1} = -\cos\frac{N\pi}{N+1} = -\lambda_1,$$

и справедливо также

$$\delta = \begin{cases} \cos\dfrac{N\pi}{2(N+1)}, & \text{если } N \text{ четно,} \\ 0, & \text{если } N \text{ нечетно.} \end{cases}$$

Отсюда в силу (3.4.21) получаем

$$\operatorname{cond}(\widehat{A}) = \begin{cases} \dfrac{1-\lambda_1^m}{1-\lambda_n^m} = \dfrac{1+\lambda_n^m}{1-\lambda_n^m}, & \text{если } m \text{ нечетно,} \\ \dfrac{1-\delta^m}{1-\lambda_n^m}; & \text{если } m \text{ четно.} \end{cases}$$

Например, при $N = 44$ имеем $\lambda_n \approx 0.9976$ и $\delta \approx 0$, поэтому

$$\operatorname{cond}(\widehat{A}) \approx \begin{cases} \dfrac{1+(0{,}9976)^m}{1-(0{,}9976)^m}, & \text{если } m \text{ нечетно,} \\ \dfrac{1}{1-(0{,}9976)^m}, & \text{если } m \text{ четно.} \end{cases}$$

Соответствующие величины для нескольких первых значений m приведены в табл. 3.4.1. Заметим, что число обусловленности

Таблица 3.4.1. Числа обусловленности для предобусловливания по Якоби

m	1	2	3	4
$\operatorname{cond}(\widehat{A})$	830	208	276	104

для $m = 1$ совпадает с числом обусловленности самой матрицы A. Это происходит из-за того, что все элементы на главной диагонали матрицы A равны 4, и один шаг предобусловливания по методу Якоби приводит лишь к масштабированию матрицы A путем деления каждого ее элемента на 4 и, таким образом, не влияет на ее число обусловленности (упражнение 3.4.10). Из табл. 3.4.1 видно, что $\operatorname{cond}(\widehat{A})$ не убывает монотонно при возрастании m. Влияние этого обстоятельства на поведение метода сопряженных градиентов с m-шаговым предобусловливанием Якоби в применении к дискретному уравнению Пуассона выражается, вообще говоря, в увеличении числа итераций при переходе от использования $m-1$ шагов к использованию m шагов при нечетном m. В качестве примера в табл. 3.4.2

Таблица 3.4.2. Число итераций для m-шагового предобусловливания по Якоби в методе PCG

m	0	1	2	3	4	5	6	7	8
Число итераций	45	45	23	36	21	30	18	26	16

приводятся значения числа итераций, требуемых для решения рассматриваемой системы при $N = 89$.

Полиномиальное предобусловливание

Матрица R в (3.4.18) представляет собой полином специального вида от матрицы H, умноженный на P^{-1}. Указанное обстоятельство наводит на мысль о рассмотрении полиномов от H общего вида и, соответственно, матрицы R, определенной соотношением

$$R = (\alpha_0 I + \alpha_1 H + \ldots + \alpha_{m-1} H^{m-1}) P^{-1}. \qquad (3.4.24)$$

Теперь наша цель заключается в том, чтобы выбрать значения $\alpha_0, \ldots, \alpha_{m-1}$, при которых сохраняется положительная определенность матрицы R и минимизируется отношение наибольшего собственного значения матрицы $RA = M^{-1}A$ к наименьшему.

С вычислительной точки зрения, параметры α_i вводятся в итерационный процесс во время накопления линейных комбинаций итерационных приближений следующим образом. Если на k-й итерации метода PCG вспомогательный итерационный процесс представлен соотношением $\mathbf{r}^{i+1} = H\mathbf{r}^i + P^{-1}\mathbf{r}$, где $\mathbf{r}^0 = 0$, то вычисление $\tilde{\mathbf{r}}$ осуществляется по формуле

$$\tilde{\mathbf{r}}^m = (\ldots((\mu_0 \mathbf{r}^1) + \mu_1 \mathbf{r}^2) + \ldots + \mu_{m-1}\mathbf{r}^m), \qquad (3.4.25)$$

где скобки показывают, что вектор $\tilde{\mathbf{r}}^m$ накапливается в результате вычисления триад по мере возникновения итерационных приближений $\mathbf{r}^i$, причем требуется лишь один дополнительный вектор памяти. Константы μ_i в (3.4.25) связаны с величинами α_i в (3.4.24) соотношением

$$\mu_i + \mu_{i+1} + \ldots + \mu_{m-1} = \alpha_i, \quad i = 0, \ldots, m-1, \qquad (3.4.26)$$

т. е. $\mu_{m-1} = \alpha_{m-1}$, $\mu_{m-2} = \alpha_{m-2} - \mu_{m-1}$, и т. д.

Формулы (3.4.26) можно вывести следующим образом. Как уже отмечалось, справедливо представление $\mathbf{r}^{i+1} = (I + \ldots + H^i)P^{-1}\mathbf{r}$. Если подставить его в (3.4.25) и приравнять коэффициенты при членах $H^i P^{-1}\mathbf{r}$ в (3.4.24) и (3.4.25), то по-

лучаются равенства (3.4.26). Проверку этого факта мы оставляем в качестве упражнения 3.4.12.

Обсудим теперь, как можно получить параметры $\alpha_0, \ldots, \alpha_{m-1}$. В условиях теоремы 3.4.2 матрица R, заданная соотношением (3.4.24), остается симметричной, так как введение параметров α_i не затрагивает обоснования симметрии. Поскольку $H = I - P^{-1}A$,

$$RA = \left(\alpha_0 I + \alpha_1 (I - P^{-1}A) + \ldots + \alpha_{m-1}(I - P^{-1}A)^{m-1}\right) P^{-1}A,$$

т. е. матрица RA представляет собой полином от $P^{-1}A$. Нужно выбрать α_i так, чтобы собственные значения матрицы RA были положительными. Поскольку матрица R симметрична, а матрица A симметрична и положительно определена, то по теореме П.2.7 R, а значит, и M, также положительно определены. Кроме того, хотелось бы выбрать α_i таким образом, чтобы собственные значения матрицы RA были как можно ближе к единице. Вот один из возможных способов. Пусть собственные значения матрицы $P^{-1}A$ лежат в интервале $[a, b]$, и пусть полином q определен как

$$q(\lambda) = \left(\alpha_0 + \alpha_1 (1 - \lambda) + \ldots + \alpha_{m-1}(1 - \lambda)^{m-1}\right) \lambda.$$

Тогда $RA = q(P^{-1}A)$ и спектр матрицы RA принадлежит отрезку $\{q(\lambda) : \lambda \in [a, b]\}$. Таким образом, одним из критериев выбора α_i, подобным минимизации отношения наибольшего собственного значения матрицы RA к наименьшему, может служить минимизация интеграла $\int_a^b (1 - q(\lambda))^2 d\lambda$ по $\alpha_0, \ldots, \alpha_{m-1}$ [1]).

Эффект применения полиномиального предобусловливания продемонстрирован в табл. 3.4.3, где приведены результаты вычислений, связанных с решением двумерной задачи расчета напряжений [2]). В первом столбце указано количество шагов m предобусловливания SSOR, а значение m, помеченное буквой P, означает применение полиномиального предобусловливания. Через I обозначается число итераций, а через T — время счета

---

[1]) Ничто не мешает, конечно, построить полином q, минимизирующий значение $\varkappa = \lambda_{\max}(RA)/\lambda_{\min}(RA)$; при известных a и b он легко выражается через полиномы Чебышёва первого рода. Однако здесь такой подход часто обнаруживает свою несостоятельность, и гораздо эффективнее оказывается другой выбор q, который, конечно, не обеспечивает минимизацию $\varkappa$, но зато позволяет получить более выгодное с точки зрения сходимости метода сопряженных градиентов распределение собственных значений матрицы RA (см. [Johnson O. et al., 1983]). — *Прим. перев.*

[2]) Здесь, вероятно, имеется в виду задача о нагруженной плоской мембране. — *Прим. перев.*

Таблица 3.4.3. Результаты применения полиномиального предобусловливания

m	I	T	I	T	I	T
0	112	0.133	157	0.213	536	3.293
1	52	0.129	66	0.184	214	2.373
2	38	0.143	50	0.208	152	2.428
2P	31	0.116	40	0.167	118	1.885
3	31	0.155	39	0.216	124	2.585
3P	24	0.121	30	0.167	88	1.836
4	28	0.176	36	0.249	108	2.780
4P	22	0.138	24	0.166	67	1.726
5	25	0.188	33	0.274	97	2.969
5P	19	0.143	20	0.167	56	1.716
6	23	0.202	30	0.290	89	3.158
6P	18	0.159	18	0.175	47	1.670
7P					43	1.739
8P					36	1.634

на векторном компьютере CDC CYBER 203. Результаты приводятся для трех задач последовательно возрастающего размера, и можно видеть, что с увеличением размера задачи степень полинома, позволяющего получить наименьшее время вычислений, также возрастает.

Прием Айзенштата

В § 3.2 мы описали прием Конрада — Валяха, позволяющий уменьшить объем вычислений при выполнении итераций метода SSOR. Эту процедуру можно использовать для предобусловливания по методу SSOR. Опишем теперь другой способ, который будем называть *приемом Айзенштата*, также позволяющий сократить вычислительные затраты. Основное предположение заключается в том, что матрица-предобусловливатель M имеет вид

$$M = ST^{-1}S^{\mathrm{T}}, \tag{3.4.27}$$

где обычно предполагается, что S — нижнетреугольная матрица, а T — диагональная. Например, для одношагового предобусловливания SSOR получаем, используя (3.4.14) и (3.4.19), что $M = P$ и

$$S = \widehat{D} - L, \quad \widehat{D} = \omega^{-1}D, \quad S = \omega^{-1}(2-\omega)D. \tag{3.4.28}$$

Идея приема Айзенштата заключается в применении другого предобусловленного метода сопряженных градиентов к системе

$$\hat{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}, \quad \hat{A} = S^{-1}AS^{-\mathrm{T}}, \quad \hat{\mathbf{x}} = S^{\mathrm{T}}\mathbf{x}, \quad \hat{\mathbf{b}} = S^{-1}\mathbf{b}. \tag{3.4.29}$$

Нетрудно убедиться (упражнение 3.4.9), что применение метода PCG с матрицей-предобусловливателем M, заданной в (3.4.27), эквивалентно применению метода PCG с матрицей-предобусловливателем T^{-1} к системе $\hat{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}$. Для новой системы метод PCG принимает следующий вид:

$\hat{\mathbf{r}}^0 = S^{-1}(\mathbf{b} - A\mathbf{x}^0)$, $\hat{\mathbf{p}}^0 = \mathbf{q}^0 = T\hat{\mathbf{r}}^0$;

для $k = 0, 1, \ldots$

$$\hat{\alpha}_k = -(\hat{\mathbf{r}}^k, \mathbf{q}^k)/(\hat{\mathbf{p}}^k, \hat{A}\hat{\mathbf{p}}^k), \tag{3.4.30}$$

$\mathbf{x}^{k+1} = \mathbf{x}^k - \hat{\alpha}_k S^{-\mathrm{T}}\hat{\mathbf{p}}^k$,

$\hat{\mathbf{r}}^{k+1} = \hat{\mathbf{r}}^k + \hat{\alpha}_k \hat{A}\hat{\mathbf{p}}^k$;

$\mathbf{q}^{k+1} = T\hat{\mathbf{r}}^{k+1}$;

$\hat{\beta}_k$, $\hat{\mathbf{p}}^{k+1}$ вычисляются как обычно.

Заметим, что в (3.4.30) мы оставляем в соответствующей формуле исходные итерационные приближения $\mathbf{x}^k$ вместо $\hat{\mathbf{x}}^k$. К этому обстоятельству мы скоро вернемся.

Ключевым моментом здесь является выражение произведения $\hat{A}\hat{\mathbf{p}}$ через исходную матрицу A. Если $A = D - L - L^{\mathrm{T}}$, то матрицу A можно представить в виде

$$A = S + S^{\mathrm{T}} - K, \quad K = S + S^{\mathrm{T}} + L + L^{\mathrm{T}} - D. \tag{3.4.31}$$

Полагая $\mathbf{t} = S^{-\mathrm{T}}\hat{\mathbf{p}}$, получаем

$$\hat{A}\hat{\mathbf{p}} = S^{-1}AS^{-\mathrm{T}}\hat{\mathbf{p}} = S^{-1}(S + S^{\mathrm{T}} - K)S^{-\mathrm{T}}\hat{\mathbf{p}} = \mathbf{t} + S^{-1}(\hat{\mathbf{p}} - K\mathbf{t}). \tag{3.4.32}$$

Таким образом, для формирования произведения $\hat{A}\hat{\mathbf{p}}$ требуется вычисление $K\mathbf{t}$ и решение систем

$$S^{\mathrm{T}}\mathbf{t} = \hat{\mathbf{p}}, \quad S\mathbf{w} = \hat{\mathbf{p}} - K\mathbf{t}. \tag{3.4.33}$$

Но соответствующие системы с матрицами коэффициентов S и S^{T} должны решаться при выполнении предобусловливания на каждой итерации с использованием матрицы M, определенной в (3.4.27). Таким образом, прием Айзенштата позволяет получить более эффективную формулировку алгоритма при условии, что умножение матрицы K на вектор потребует меньших

затрат, чем умножение матрицы A. Заметим, что величина $S^{-T}\hat{\mathbf{p}}^k$, при помощи которой осуществляется пересчет приближения $\mathbf{x}^k$, — это просто текущее значение $\mathbf{t}$, вычисляемое в (3.4.33). Поскольку оно известно, мы можем работать с исходными векторами итерационных приближений $\mathbf{x}^k$ без каких-либо дополнительных вычислений.

Когда прием Айзенштата применяется к одношаговому предобусловливанию SSOR, матрица S определяется соотношением (3.4.28), и в силу (3.4.31)

$$K = \hat{D} - L + \hat{D} - L^T + L + L^T - D = \omega^{-1}(2-\omega)D. \quad (3.4.34)$$

Таким образом, дополнительная вычислительная работа по формированию вектора $\hat{A}\hat{\mathbf{p}}$ сводится лишь к умножению диагональной матрицы на вектор и двум векторным сложениям.

Параллельная и векторная реализация

Обсудим теперь параллельную и векторную реализацию рассматривавшихся до сих пор методов PCG. Основным методом такого типа является m-шаговый метод SSOR-PCG, но мы начнем с m-шагового метода Якоби-PCG. В предыдущем параграфе мы уже говорили о реализации исходного метода сопряженных градиентов, а в § 3.1 — о реализации метода Якоби. Вопрос теперь заключается в комбинировании этих двух методик.

Рассмотрим сначала векторный компьютер, обеспечивающий эффективную обработку длинных векторов. Тогда итерации метода Якоби, при помощи которых реализуется предобусловливание, должны выполняться с использованием как можно более длинных векторов. Для дискретных краевых задач это достигается с помощью приемов, описанных в § 3.1. Как отмечалось в предыдущем параграфе, эти же приемы можно применить для вычисления матрично-векторного произведения $A\mathbf{p}^k$, требующегося при выполнении итераций метода сопряженных градиентов.

Для предобусловливания по методу SSOR итерации метода Якоби нужно заменить на итерации SSOR. В § 3.2 показано, что для выполнения итераций SSOR с использованием операций над длинными векторами нужно вводить красно-черное упорядочение точек сетки. Для задач более общего вида в случае, когда красно-черное упорядочение не дает должного эффекта, нужно использовать многоцветные упорядочения, описанные в том же параграфе.

Рассмотрим теперь параллельные системы с локальной памятью. Для реализации предобусловливания как по методу

Якоби, так и по методу SSOR распределим неизвестные по процессорам так, как описано в § 3.1 и показано на рис. 3.1.4. Тем же самым образом осуществим распределение векторов **p** и **r**, используемых на итерациях метода сопряженных градиентов. Умножение $A\mathbf{p}^k$, вычисление скалярных произведений и триад на итерациях метода сопряженных градиентов будем производить так же, как в § 3.3, а предобусловливание выполнять при помощи тех же вычислений, что и для метода Якоби в § 3.1 или для метода SSOR в § 3.2.

Неполная факторизация Холесского

Перейдем к изложению другого важного подхода к предобусловливанию, который основан на *неполных факторизациях* матрицы A. Если LL^{T} — факторизация Холесского симметричной положительно определенной матрицы A и матрица A разреженная, то, вообще говоря, множитель L оказывается гораздо менее разреженным, чем матрица A, из-за возникающего заполнения. *Неполной факторизацией Холесского* мы будем называть соотношение вида

$$A = LL^{\mathrm{T}} + R, \tag{3.4.35}$$

где L является нижнетреугольной матрицей, но $R \neq 0$. Одним из способов получения таких неполных факторизаций может служить подавление заполнения или его части в процессе факторизации Холесского. Приведем следующий простой пример. Матрица

$$L = 2^{-1/2}\begin{bmatrix} 2 & 0 & 0 \\ 1 & 3^{1/2} & 0 \\ 1 & -3^{-1/2} & 2^{3/2}3^{-1/2} \end{bmatrix}, \tag{3.4.36}$$

представляет собой множитель Холесского для матрицы

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 0 & 2 \end{bmatrix} \tag{3.4.37}$$

и, таким образом, позиция (3, 2) заполняется. Если мы хотим получить неполную факторизацию с нулевыми элементами в тех же позициях, что и у матрицы A, то можно заменить нулями элементы множителя Холесского, расположенные в тех же позициях, что и нулевые элементы матрицы A. Так, неполный множитель для матрицы (3.4.37) будет совпадать с матрицей L, определенной в (3.4.36), за исключением элемента в позиции (3, 2), который будет равен нулю. Недостаток этого

подхода заключается в том, что приходится выполнять всю вычислительную работу, связанную с разложением Холесского, а это именно те вычислительные затраты (сопровождаемые к тому же расходом памяти), которые мы хотим устранить. Поэтому возникает следующее требование: не вычислять элементы L в позициях, где расположены нулевые элементы матрицы A. Это приводит к принципу *неполной факторизации Холесского без заполнения* (или $IC(0)$*-принципу*):

$$\begin{aligned} &\text{если } a_{ij} \neq 0, \quad \text{выполнить вычисление } l_{ij} \text{ по методу Холесского;} \\ &\text{если } a_{ij} = 0, \quad \text{положить } l_{ij} = 0. \end{aligned} \tag{3.4.38}$$

Алгоритм, построенный в соответствии с принципом (3.4.38), приведен на рис. 3.4.1. Заметим, что если удалить условный оператор, получится алгоритм точной факторизации Холесского.

$$\begin{aligned} &l_{11} = (a_{11})^{1/2} \\ &\text{Для } i \text{ от } 2 \text{ до } n \\ &\quad \text{Для } j \text{ от } 1 \text{ до } i-1 \\ &\quad\quad \text{Если } a_{ij} = 0 \text{ то } l_{ij} = 0 \text{ иначе} \\ &\quad\quad\quad l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}\right) / l_{jj} \\ &\quad l_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2\right)^{1/2} \end{aligned} \tag{3.4.39}$$

Рис. 3.4.1. Неполная факторизация Холесского

Если применить алгоритм неполной факторизации Холесского, показанный на рис. 3.4.1, к матрице (3.4.37), то получится неполный множитель

$$L = 2^{-1/2} \begin{bmatrix} 2 & 0 & 0 \\ 1 & 3^{1/2} & 0 \\ 1 & 0 & 3^{1/2} \end{bmatrix}.$$

Заметим, что элемент в позиции (3, 3) изменился по сравнению с (3.4.36) вследствие того, что элемент (3, 2) был приравнен к нулю.

Принцип отсутствия заполнения представляет собой частный случай более общего подхода к определению неполных факторизаций. Пусть $S = \{(i, j)\}$ — заданное множество индексов, где i и j принимают значения между 1 и n. Тогда (3.4.38)

заменяется на следующую схему:

$$\begin{aligned} &\text{если } (i, j) \in S, \text{ выполнить вычисление } l_{ij} \text{ по} \\ &\qquad\qquad \text{методу Холесского (3.4.39);} \\ &\text{если } (i, j) \notin S, \text{ положить } l_{ij} = 0. \end{aligned} \tag{3.4.40}$$

Если $S = \{(i, j): a_{ij} \neq 0\}$, то (3.4.40) сводится к принципу отсутствия заполнения (3.4.38). Однако при подходящем выборе S принцип отсутствия заполнения можно ослабить, допустив некоторую степень заполнения, и есть много возможностей это сделать. Например, если A — диагонально разреженная матрица, то одним из распространенных подходов является введение нескольких дополнительных диагоналей в структуру неполного множителя L помимо диагоналей самой матрицы A

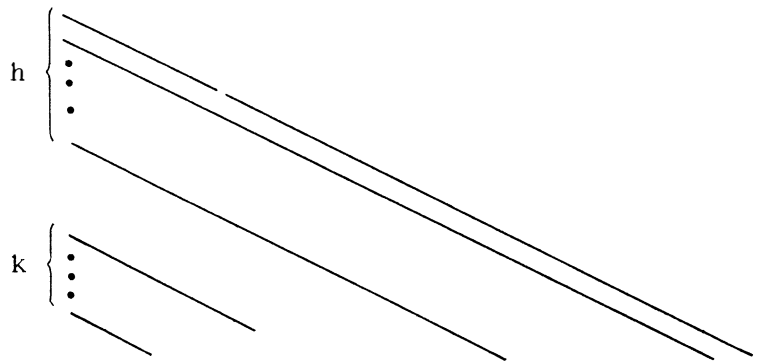


Рис. 3.4.2. Структура заполнения, соответствующая IC(h, k)-принципу

(конечно, здесь учитываются только те диагонали матрицы A, которые расположены ниже главной диагонали). В частности, для пятидиагональной матрицы, такой, как матрица дискретного уравнения Пуассона (3.1.7), *IC(h, k)-принцип* допускает наличие $h + k$ диагоналей, расположенных ниже главной диагонали матрицы L. Иллюстрацией служит рис. 3.4.2, где предполагается, что A имеет две ненулевые поддиагонали. Тогда допускается заполнение $h - 1$ дополнительных диагоналей, смежных с первой поддиагональю, и $k - 1$ дополнительных диагоналей, смежных со второй поддиагональю. Таким образом, IC(1, 1)-принцип в точности совпадает с принципом отсутствия заполнения. Возможны и другие стратегии заполнения, соответствующие различным множествам S.

Если LL^T является неполным разложением, то матрица $R = A - LL^T$ всегда отлична от нулевой. Однако нулевые элементы матрицы R могут располагаться заранее определенным

образом. Неполное разложение называется *регулярным* относительно индексного множества S, если $r_{ij} = 0$ для всех $(i, j) \in S$. Это выполняется всегда, когда неполное разложение вычисляется способом (3.4.40) и, следовательно, является регулярным в силу соотношений (3.4.39), служащих для вычисления l_{ij}, из которых вытекает

$$r_{ij} = a_{ij} - \sum_{k=1}^{j} l_{ik} l_{jk} = 0. \qquad (3.4.41)$$

Однако неполные факторизации, не соответствующие принципу (3.4.40), не обязательно регулярны.

В отличие от полного разложения Холесского, неполное разложение симметричной положительно определенной матрицы возможно не всегда. Попытка его вычисления может потерпеть неудачу из-за необходимости извлечения квадратного корня из отрицательного числа при вычислении диагонального элемента или, в случае применения разложения типа LDL^{T}, свободного от квадратных корней, матрица D может оказаться незнакоопределенной. Приведем теперь (без доказательства) результат, устанавливающий условия выполнимости неполного разложения Холесского. Напомним, что *M-матрицей* называется такая матрица, внедиагональные элементы которой неположительны, а все элементы обратной к ней матрицы неотрицательны (см. приложение 2, определение П.2.12). Матрица A с положительными диагональными элементами называется *H-матрицей*, если матрица $\hat{A}$ с элементами $\hat{a}_{ij} = -|a_{ij}|$, $i \neq j$, и $\hat{a}_{ii} = a_{ii}$ является M-матрицей.

3.4.7. Теорема. *Если A является симметричной H-матрицей, то для любого множества S неполная факторизация* (3.4.40) *выполнима.*

Если условия теоремы 3.4.7 не выполнены, неполное разложение симметричной положительно определенной матрицы может потерпеть неудачу. Для преодоления подобных ситуаций разработаны некоторые приемы. Предположим, что на i-м шаге получилось $l_{ii}^2 \leqslant 0$, т. е. либо требуется извлечение квадратного корня из отрицательного числа, либо (в случае $l_{ii}^2 = 0$) матрица L вырождена. Одним из простейших приемов является замена l_{ii}^2 на некоторое положительное число; например, можно выбрать значение предыдущего диагонального элемента $l_{i-1,\,i-1}^2$. Другая стратегия заключается в замене l_{ii}^2 на $\left(\sum_{j<i} |l_{ij}|\right)^2$, что позволяет обеспечить диагональное преобладание в новой i-й строке матрицы L.

Еще один подход основан на теореме 3.4.7 и носит название *стратегии сдвига*. Рассмотрим матрицу $\hat{A} = A + \alpha I$. Ясно, что при некотором выборе $\alpha > 0$ матрица $\hat{A}$ будет иметь строгое диагональное преобладание. Поскольку матрица со строгим диагональным преобладанием, имеющая положительные диагональные элементы и неположительные внедиагональные элементы, является M-матрицей (см. приложение 2, утверждение П.2.16), то $\hat{A}$ представляет собой H-матрицу, и поэтому неполная факторизация Холесского выполнима.

Неполное разложение матриц специального вида

В общем случае неполная факторизация может потребовать значительных вычислительных затрат. Однако в некоторых специальных случаях такую факторизацию можно получить очень легко. Будем работать со свободным от корней разложением Холесского вида LDL^{T} и применять стратегию, основанную на принципе отсутствия заполнения. Тогда алгоритм неполной факторизации принимает вид, показанный на рис. 3.4.3.

Для i от 1 до n

$$d_i = a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 d_k \qquad (3.4.42)$$

Для j от 1 до $i-1$

Если $a_{ij} = 0$ то $l_{ij} = 0$ иначе

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_k l_{jk}\right) \Big/ d_j \qquad (3.4.43)$$

Рис. 3.4.3. Неполное LDL^{T}-разложение без заполнения

Рассмотрим теперь блочно-трехдиагональную матрицу

$$A = \begin{bmatrix} A_1 & B_1^{\mathrm{T}} & & & \\ B_1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & B_{N-1}^{\mathrm{T}} \\ & & & B_{N-1} & A_N \end{bmatrix}, \qquad (3.4.44)$$

где A_i — трехдиагональные матрицы $N \times N$, а B_i — диагональные матрицы $N \times N$; таким образом, A представляет собой пятидиагональную матрицу. Напомним, что именно такую форму имеет матрица дискретного уравнения Пуассона (3.1.5), так же как и матрица более общего вида (3.1.17). При факторизации без заполнения матрица L будет иметь ненулевые элементы только в тех поддиагональных позициях, которые содержат ненулевые элементы в матрице A. Рассмотрим сначала элемент $l_{i,\,i-N}$, отвечающий некоторому элементу подматрицы B_i. Поскольку все элементы i-й строки матрицы L, расположенные слева от $l_{i,\,i-N}$, равны нулю, то в силу (3.4.43)

$$l_{i,\,i-N} = a_{i,\,i-N}/d_{i-N}. \tag{3.4.45}$$

Аналогично, в силу принципа отсутствия заполнения единственным ненулевым элементом слева от $l_{i,\,i-1}$ является $l_{i,\,i-N}$, и поэтому

$$l_{i,\,i-1} = (a_{i,\,i-1} - l_{i,\,i-N} d_{i-N} l_{i-1,\,i-N})/d_{i-1}.$$

Но $l_{i-1,\,i-N}$ — это элемент в $(i-1)$-й строке, расположенный непосредственно над элементом $l_{i,\,i-N}$, и поэтому он должен быть равен нулю. Отсюда получаем

$$l_{i,\,i-1} = a_{i,\,i-1}/d_{i-1}. \tag{3.4.46}$$

Элементы матрицы D задаются в соответствии с (3.4.42) как

$$d_i = a_{ii} - l^2_{i,\,i-N} d_{i-N} - l^2_{i,\,i-1} d_{i-1}, \tag{3.4.47}$$

поскольку единственными внедиагональными элементами матрицы L в i-й строке будут $l_{i,\,i-1}$ и $l_{i,\,i-N}$. Используя (3.4.45) и (3.4.46), можно также записать (3.4.47) в виде

$$d_i = a_{ii} - a^2_{i,\,i-N}/d_{i-N} - a^2_{i,\,i-1}/d_{i-1}. \tag{3.4.48}$$

Таким образом, для пятидиагональной матрицы вида (3.4.44) неполная факторизация без заполнения очень легко получается из (3.4.45), (3.4.46) и либо (3.4.47), либо (3.4.48). Заметим, что аналогичные формулы справедливы для семидиагональной матрицы, возникающей при дискретизации трехмерного уравнения типа Пуассона (упражнение 3.4.14). С другой стороны, такие формулы уже не справедливы, если матрицы B_i в (3.4.44) являются трехдиагональными, как в случае, когда рассматриваются уравнения (3.2.24) (упражнение 3.4.15).

Поддиагональные элементы матрицы L в точности совпадают с соответствующими элементами матрицы A, разделенными на подходящие элементы матрицы D. По этой причине

оказывается возможным записать такую факторизацию в виде

$$L_1 D_1 L_1^{\mathrm{T}}, \quad \operatorname{diag}(L_1) = D_1^{-1}. \tag{3.4.49}$$

В этом представлении поддиагональные элементы матрицы L_1 совпадают с соответствующими элементами матрицы A (упражнение 3.4.16). Если используется (3.4.49), может оказаться выгодным масштабировать матрицу A так, чтобы выполнялось $D_1 = I$. Тогда матрица-предобусловливатель M принимает вид $M = (I + \hat{L})(I + \hat{L}^{\mathrm{T}})$, где $\hat{L}$ — строго нижнетреугольная матрица. Детали такого масштабирования см. в упражнении 3.4.17.

Блочные методы

Обратимся теперь к рассмотрению блочных неполных факторизаций и предположим, что A — блочно-трехдиагональная матрица (3.4.44). Нетрудно проверить (упражнение 3.4.18), что блочное разложение (точное) матрицы A имеет вид

$$A = (D + L) D^{-1} (D + L^{\mathrm{T}}), \tag{3.4.50}$$

где

$$L = \begin{bmatrix} 0 & & & \\ B_1 & \ddots & & \\ & \ddots & \ddots & \\ & & B_{N-1} & 0 \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & & \\ & \ddots & \\ & & D_N \end{bmatrix}, \tag{3.4.51}$$

а матрицы D_i задаются соотношениями

$$D_1 = A_1, \quad D_1 = A_i - B_{i-1} D_{i-1}^{-1} B_{i-1}^{\mathrm{T}}, \quad i = 2, \ldots, N. \tag{3.4.52}$$

Заменим теперь (3.4.52) на формулы

$$D_1 = A_1, \quad D_i = A_i - B_{i-1} C_{i-1} B_{i-1}^{\mathrm{T}}, \quad i = 2, \ldots, N, \tag{3.4.53}$$

где матрица C_{i-1} является некоторой аппроксимацией для D_{i-1}^{-1}. Тогда матрица

$$M = (D + L) D^{-1} (D + L^{\mathrm{T}}), \tag{3.4.54}$$

где матрица задана формулой 3.4.51, представляет собой приближенное, или неполное, разложение матрицы A. В зависимости от того, как именно осуществляется аппроксимация

матриц C_i, возможны различные версии неполного блочного разложения.

Предположим теперь, что матрицы A_i трехдиагональные, а матрицы B_i диагональные, причем A является M-матрицей и справедливы условия

$$\sum_{j \neq k} |a_{kj}| \leqslant a_{kk}, \quad k = 1, \ldots, n,$$

со строгим неравенством хотя бы для одного значения i. При этих предположениях мы будем аппроксимировать матрицы, обратные к матрицам D_i, при помощи трехдиагональных матриц C_i. В этом случае в силу (3.4.53) матрицы D_i также будут трехдиагональными (однако матрицы D_i^{-1} будут в общем случае плотными). Одной из отправных точек аппроксимации матрицы D_i^{-1} может служить следующая теорема, которую мы приведем без доказательства.

3.4.8. Теорема. *Пусть T — неприводимая невырожденная симметричная трехдиагональная матрица $m \times m$. Тогда существуют два таких вектора u и v, что*

$$T^{-1} = \begin{bmatrix} u_1v_1 & u_1v_2 & \ldots & u_1v_m \\ u_1v_2 & u_2v_2 & \ldots & u_2v_m \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ u_1v_m & u_2v_m & \ldots & u_mv_m \end{bmatrix}.$$

Если, кроме того, T является M-матрицей со строгим диагональным преобладанием, то

$$u_1 < u_2 < \ldots < u_m, \quad v_1 > v_2 > \ldots > v_m. \tag{3.4.55}$$

Неравенства (3.4.55) показывают, что элементы матрицы T^{-1} уменьшаются по мере удаления от главной диагонали, и это наводит на мысль о том, что аппроксимация T^{-1} посредством матриц с небольшой шириной ленты может оказаться вполне удовлетворительной. Можно вывести достаточно простые рекуррентные соотношения для u_i и v_i, однако даже если мы захотим вычислить лишь диагональную часть матрицы T^{-1}, нам потребуются все значения u_i и v_i. Опишем более простой метод аппроксимации.

Пусть $a_1, \ldots, a_m$ — диагональные элементы матрицы T, а $b_1, \ldots, b_{m-1}$ — внедиагональные элементы. Если $T = LL^{\mathrm{T}}$ —

факторизация Холесского матрицы T, то

$$L = \begin{bmatrix} \gamma_1 & & & & \\ -\delta_1 & \ddots & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & -\delta_{m-1} & \gamma_m \end{bmatrix}, \tag{3.4.56}$$

где

$$\gamma_1^2 = a_1, \quad \delta_1\gamma_1 = -b_1, \quad \delta_{i-1}^2 + \gamma_i^2 = a_i, \quad \delta_i\gamma_i = -b_i, \quad i \geqslant 2.$$

Получаем $T^{-1} = L^{-\mathrm{T}}L^{-1}$, и нетрудно проверить (упражнение 3.4.19), что первые две диагонали матрицы L^{-1} имеют вид

$$\begin{bmatrix} \gamma_1^{-1} & & & \\ \eta_1 & \gamma_2^{-1} & & \\ & \ddots & \ddots & \\ & & \eta_{m-1} & \gamma_m^{-1} \end{bmatrix}, \tag{3.4.57}$$

где

$$\eta_i = \frac{\delta_i}{\gamma_i\gamma_{i+1}}, \quad 1 \leqslant i \leqslant m-1.$$

Таким образом, первые две диагонали матрицы L^{-1} вычисляются довольно просто. Если V — аппроксимация L, образованная этими двумя диагоналями, то матрица $C = V^{\mathrm{T}}V$ является трехдиагональной и может служить аппроксимацией T^{-1}.

Метод неполного разложения Холесского — сопряженных градиентов

Рассмотрим теперь использование неполных факторизаций Холесского в качестве предобусловливателей для метода сопряженных градиентов. Пусть (3.4.35) — любая неполная факторизация с невырожденной матрицей L. Выберем теперь матрицу-предобусловливатель M в предобусловленном методе сопряженных градиентов PCG (3.4.9) равной

$$M = LL^{\mathrm{T}}. \tag{3.4.58}$$

Поскольку матрица L невырожденна, матрица M симметрична и положительно определена. Решение систем типа $M\tilde{\mathbf{r}} = \mathbf{r}$ в

алгоритме PCG выполняется при помощи последовательного решения двух треугольных систем

$$L\mathbf{x} = \mathbf{r}, \quad L^{\mathrm{T}}\tilde{\mathbf{r}} = \mathbf{x}. \tag{3.4.59}$$

Заметим, что неполная факторизация выполняется лишь один раз перед началом итераций PCG и множитель L должен храниться для использования при решении систем (3.4.59) на каждой итерации метода PCG.

Указанный выше выбор матрицы M определяет общий класс *методов неполного разложения Холесского — сопряженных градиентов*, или *методов ICCG* (incomplete Choleski conjugate gradient). Дальнейшая детализация метода зависит от способа получения неполной факторизации Холесского. Так, например, если неполная факторизация получена на основе принципа отсутствия заполнения (3.4.38), то соответствующий метод сопряженных градиентов обозначается через ICCG(0) или, в соответствии с рис. 3.4.2, через ICCG(1, 1).

Как уже указывалось, иногда оказывается удобным рассматривать неполное разложение Холесского в форме, свободной от корней, где (3.4.35) заменено на

$$A = LDL^{\mathrm{T}} + R; \tag{3.4.60}$$

здесь D — положительно определенная диагональная матрица, а матрица L теперь имеет диагональные элементы, равные единице. Если взять $M = LDL^{\mathrm{T}}$, то прямая и обратная подстановка (3.4.59) заменяются на формулы

$$L\mathbf{z} = \mathbf{r}, \quad D\mathbf{x} = \mathbf{z}, \quad L^{\mathrm{T}}\tilde{\mathbf{r}} = \mathbf{x}. \tag{3.4.61}$$

Параллельная и векторная реализация метода ICCG

Параллельная или векторная реализация метода сопряженных градиентов как такового остается такой же, как в § 3.3. Проблема заключается в реализации неполной факторизации и, что еще важнее, прямой и обратной подстановок (3.4.59) или (3.4.61), поскольку их нужно выполнять на каждой итерации. Векторизуемость и параллелизм алгоритмов факторизации Холесского, описанных в гл. 2, оказываются совершенно неудовлетворительными для больших разреженных матриц, для которых мы хотим использовать итерации метода сопряженных градиентов. Аналогичные реализации неполной факторизации сталкиваются с теми же трудностями. Опишем некоторые приемы, позволяющие их преодолеть.

Рассмотрим сначала блочно-трехдиагональную матрицу (3.4.44). Для простоты будем предполагать, что все матрицы

A_i и B_i имеют размеры $N \times N$ и что все матрицы A_i являются ленточными с полушириной ленты α, а матрицы B_i — ленточными с полушириной ленты β. Как мы уже видели, матрица A дискретного уравнения Пуассона (3.1.5) представляет собой матрицу именно такого вида при $\alpha = 1$ и $\beta = 0$, в то время как для матрицы уравнений (3.2.24) имеем $\alpha = \beta = 1$.

Предположим, что мы получили неполную факторизацию без заполнения LL^T матрицы A. Таким образом, $L + L^T$ имеет

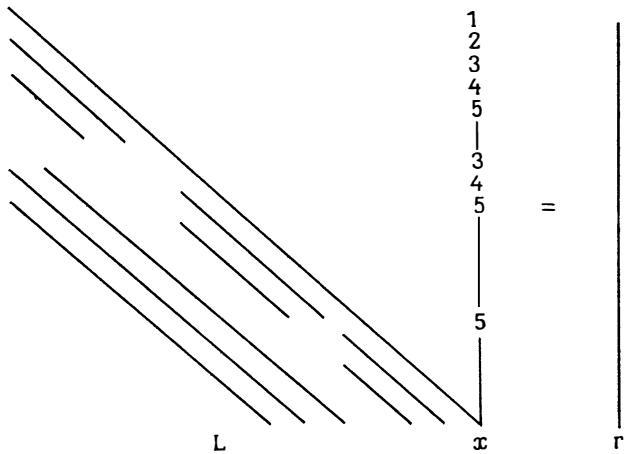


Рис. 3.4.4. Решение системы $L\mathbf{x} = \mathbf{r}$

ту же структуру расположения ненулевых элементов, что и A, и мы можем представить L в виде

$$L = \begin{bmatrix} L_1 & & & \\ K_1 & L_2 & & \\ & \ddots & \ddots & \\ & & K_{N-1} & L_N \end{bmatrix},$$

где все матрицы L_i и K_i имеют размер $N \times N$ с полушириной ленты соответственно α и β. В (3.4.45)—(3.4.47) мы показали, как можно получить такие факторизации в случае $\alpha = 1$ и $\beta = 0$. Сейчас мы сосредоточимся на решении систем (3.4.59).

Проиллюстрируем предлагаемую идею схемой, показанной на рис. 3.4.4, где предполагается, что $\alpha = 2$ и $\beta = 1$. Цифры, проставленные в столбце для вектора x, указывают номер стадии процесса вычислений, на которой соответствующее уравнение может быть решено. На шагах 1 и 2 решаются первые два

уравнения. На шаге 3 решается третье уравнение, но одновременно может быть решено также и $(N+1)$-е уравнение, так как уже доступны компоненты решения x_1 и x_2. На шаге 4 можно решить четвертое и $(N+2)$-е уравнения. Тогда на шаге 5 можно решить пятое, $(N+3)$-е и $(2N+1)$-е уравнения и т. д. Через каждые два шага дополнительно может быть параллельно решено еще одно уравнение, и такие добавления возможны до тех пор, пока не будет достигнуто максимальное значение, составляющее около $N/2$ уравнений. Заметим, что мы можем начать решение второго блока уравнений на шаге 3, третьего блока на шаге 5, и т. д., вплоть до N-го блока на шаге $2N-1$. Таким образом, решение будет завершено за $3N-2$ шагов. Поскольку нужно решать N^2 уравнений, средняя степень параллелизма составит $N^2/(3N-2) \approx N/3$. В упражнении 3.4.20 рассматривается случай, когда блочный порядок равен M и каждый блок представляет собой матрицу $N \times N$.

Описанная схема проиллюстрирована для значений полуширины $\alpha = 2$ и $\beta = 1$. Заметим, что моменты времени, в которые можно последовательно начинать решение блоков уравнений, зависят только от β, но не от α; в частности, через каждые $\beta+1$ шагов можно начать обработку еще одного блока. Для дискретного уравнения Пуассона (3.1.5) $\beta = 0$, т. е. обработку нового блока уравнений можно начинать на каждом шаге, в то время как для уравнений (3.2.24) $\beta = 1$, и поэтому обработка нового блока может начинаться на каждом втором шаге. Ключевым моментом описанного процесса является «развязывание» уравнений каждого блока, возможное после того, как вычислены некоторые неизвестные, и такая стратегия не будет работать для матриц с плотно заполненной лентой. Отметим сходство рассмотренной процедуры с алгоритмом Лори — Самеха, описанным в § 2.3.

Решение верхнетреугольной системы $L^T\tilde{\mathbf{r}} = \mathbf{x}$ можно осуществить аналогичным образом, выполняя обработку в обратном порядке.

Полезно проинтерпретировать описанный процесс с использованием разностной сетки, на которой задано решаемое уравнение. Сделаем это для уравнения (3.2.24). Сеточный шаблон соответствующей дискретизации указан на рис. 3.2.7. Нижней и верхней треугольным частям матрицы A отвечают сеточные шаблоны, показанные на рис. 3.4.5, где буквой P помечена центральная точка шаблона. Поскольку неполные множители L и L^T имеют ту же структуру заполнения, что и нижняя и верхняя треугольные части матрицы A, мы можем связать их с шаблонами, приведенными на рис. 3.4.5. В частности, решение

уравнения системы $L\mathbf{x} = \mathbf{r}$, отвечающего точке сетки P, требует, чтобы решение уже было известно в соседних западной, южной, юго-восточной и юго-западной точках. Соответственно решение уравнения системы $L^T\tilde{\mathbf{r}} = \mathbf{x}$, отвечающего точке сетки P, требует известных значений в соседних восточной, северной, северо-восточной и северо-западной точках.

Рис. 3.4.5. Шаблоны, соответствующие матрицам L и L^T

Рассмотрим теперь обход узлов сетки в процессе решения системы $L\mathbf{x} = \mathbf{r}$, когда каждое уравнение решается сразу же, как только становятся известны требуемые предыдущие значения компонент решения. Порядок, в котором можно решать эти

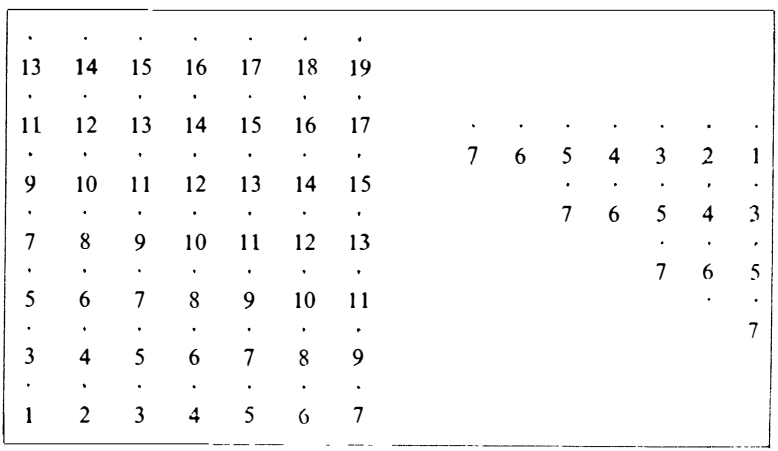


Рис. 3.4.6. Порядок решения систем $L\mathbf{x} = \mathbf{r}$ и $L^T\tilde{\mathbf{r}} = \mathbf{x}$

уравнения, показан на рис. 3.4.6a, где одной и той же цифрой, скажем k, помечены те точки, в которых решение соответствующих уравнений может быть выполнено параллельно на k-м шаге. Порядок решения для системы $L^T\tilde{\mathbf{r}} = \mathbf{x}$ показан (не столь детально) на рис. 3.4.6b; в этом случае мы передвигаемся по сетке в обратном направлении. Если сетка имеет размеры $N \times N$, то описанный процесс эквивалентен процессу для

системы, изображенной на рис. 3.4.4. Заметим, что порядок решения на рис. 3.4.6 аналогичен диаграмме времен пересчета компонент итерационного приближения на рис. 3.2.16 для метода SOR с реорганизованными потоками данных.

Многоцветная реализация ICCG

Рассмотрим теперь реализацию алгоритмов ICCG для компьютеров, ориентированных на обработку длинных векторов. Эта же реализация позволяет получить алгоритмы, обладающие очень высокой степенью параллелизма. Для некоторых задач, возникающих при дискретизации уравнений в частных производных, можно получить эффективную реализацию, используя многоцветные упорядочения точек сетки.

Чтобы проиллюстрировать предлагаемый подход, предположим сначала, что матрица коэффициентов A имеет вид

$$A=\begin{bmatrix} D_1 & C^{\mathrm{T}} \\ C & D_2 \end{bmatrix}, \tag{3.4.62}$$

где матрицы D_1 и D_2 диагональные, а C диагонально разреженная. Напомним (см. § 3.2), что матрица такого вида может возникнуть при дискретизации уравнения в частных производных, если использовать красно-черное упорядочение точек сетки. Точное разложение Холесского, свободное от корней, для матрицы A принимает вид

$$\begin{bmatrix} I & 0 \\ L_3 & I \end{bmatrix}\begin{bmatrix} \hat{D}_1 & 0 \\ 0 & \hat{D}_2 \end{bmatrix}\begin{bmatrix} L_1^{\mathrm{T}} & L_3^{\mathrm{T}} \\ 0 & L_2^{\mathrm{T}} \end{bmatrix}=\begin{bmatrix} D_1 & C^{\mathrm{T}} \\ C & D_2 \end{bmatrix}, \tag{3.4.63}$$

где L_1 и L_2 — нижнетреугольные матрицы с единицами на главной диагонали, а $\hat{D}_i$ — диагональные матрицы. Приравнивая соответствующие члены в (3.4.63), получаем

$$L_1\hat{D}_1L_1^{\mathrm{T}}=D_1, \quad L_3\hat{D}_1L_1^{\mathrm{T}}=C, \quad L_3\hat{D}_1L_3^{\mathrm{T}}+L_2\hat{D}_2L_2^{\mathrm{T}}=D_2. \tag{3.4.64}$$

Из первого соотношения (3.4.64) вытекают (упражнение 3.4.21) равенства $L_1=I$ и $\hat{D}_1=D_1$. Таким образом,

$$L_1=I, \quad \hat{D}_1=D_1, \quad L_3=CD_1^{-1}. \tag{3.4.65}$$

Рассмотрим теперь факторизацию IC(0) матрицы (3.4.62), для которой введем обозначения

$$M=\begin{bmatrix} L_1 & 0 \\ L_3 & L_2 \end{bmatrix}\begin{bmatrix} \hat{D}_1 & 0 \\ 0 & \hat{D}_2 \end{bmatrix}\begin{bmatrix} L_1^{\mathrm{T}} & L_3^{\mathrm{T}} \\ 0 & L_2^{\mathrm{T}} \end{bmatrix}. \tag{3.4.66}$$

Поскольку соотношения (3.4.65) показывают, что точное разложение Холесского позволяет получить блоки L_1 и L_3 без заполнения, то L_1, L_3 и $\bar{D}_1$, определенные в (3.4.65), представляют собой в точности те блоки, которые возникают при неполном разложении Холесского (3.4.66). Кроме того, блок L_2 в неполной факторизации должен быть равен единичной матрице, так как заполнение не допускается, и поэтому

$$\hat{D}_2 = \bar{D}_2 \equiv \text{диагональная часть } (D_2 - CD_1^{-1}C^{\mathrm{T}}). \quad (3.4.67)$$

Таким образом, множители факторизации IC(0) вида (3.4.66) задаются формулами

$$L = \begin{bmatrix} I & 0 \\ CD_1^{-1} & I \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & 0 \\ 0 & \bar{D}_2 \end{bmatrix}. \quad (3.4.68)$$

Для получения матрицы $\bar{D}_2$ нужны только те операции, которые необходимы для вычисления диагональных элементов.

Если разбить векторы $\mathbf{r}$, $\mathbf{x}$ и $\mathbf{z}$ в системах (3.4.61) согласованно с A, то эти системы принимают следующий вид:

$$\begin{bmatrix} I & 0 \\ CD_1^{-1} & I \end{bmatrix}\begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix} \quad \begin{bmatrix} D_1 & 0 \\ 0 & \bar{D}_2 \end{bmatrix}\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix},$$
$$\begin{bmatrix} I & D_1^{-1}C^{\mathrm{T}} \\ 0 & I \end{bmatrix}\begin{bmatrix} \tilde{\mathbf{r}}_1 \\ \tilde{\mathbf{r}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}. \quad (3.4.69)$$

Решая их, получаем формулы

$$\begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 - CD_1^{-1}\mathbf{r}_1 \end{bmatrix}, \quad \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} D_1^{-1}\mathbf{z}_1 \\ \bar{D}_2^{-1}\mathbf{z}_2 \end{bmatrix},$$
$$\begin{bmatrix} \tilde{\mathbf{r}}_1 \\ \tilde{\mathbf{r}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 - D_1^{-1}C^{\mathrm{T}}\mathbf{x}_2 \\ \mathbf{x}_2 \end{bmatrix},$$

или

$$\tilde{\mathbf{r}}_2 = \bar{D}_2^{-1}(\mathbf{r}_2 - CD_1^{-1}\mathbf{r}_1), \quad \tilde{\mathbf{r}}_1 = D_1^{-1}\mathbf{r}_1 - D_1^{-1}C^{\mathrm{T}}\tilde{\mathbf{r}}_2. \quad (3.4.70)$$

Заметим, что формирование вектора $\hat{\mathbf{r}}$ требует только умножения на вектор диагональных матриц или диагонально разреженных матриц. Таким образом, решение треугольных систем (3.4.61) сводится к операциям векторного и параллельного вычисления.

Если матрицу коэффициентов нельзя привести к красно-черной форме (3.4.62), но можно представить в виде c-цветной

матрицы (см. § 3.2), то можно применить ту же самую технику. Пусть

$$A=\begin{bmatrix} D_1 & C_{21}^{\mathrm{T}} & \dots & C_{c,1}^{\mathrm{T}} \\ C_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & C_{c,c-1}^{\mathrm{T}} \\ C_{c,1} & \dots & C_{c,c-1} & D_c \end{bmatrix}, \tag{3.4.71}$$

где D_i — диагональные матрицы, а C_{ij} — диагонально разреженные. Аналогично (3.4.63), блочная факторизация Холесского, свободная от корней, задается для матрицы A соотношением

$$A=\begin{bmatrix} L_{11} & & \\ \vdots & \ddots & \\ L_{c,1} & \dots & L_{c,c} \end{bmatrix}\begin{bmatrix} \hat{D}_1 & & \\ & \ddots & \\ & & \hat{D}_c \end{bmatrix}\begin{bmatrix} L_{11}^{\mathrm{T}} & \dots & L_{c,1}^{\mathrm{T}} \\ & \ddots & \vdots \\ & & L_{c,c}^{\mathrm{T}} \end{bmatrix}, \tag{3.4.72}$$

где все диагональные элементы матриц L_{ii} равны единице. Отсюда следует (упражнение 3.4.22), что матрицы L_{ij} и D_i удовлетворяют соотношениям

$$L_{jj}\hat{D}_jL_{jj}^{\mathrm{T}}=D_j-\sum_{k=1}^{j-1}L_{jk}\hat{D}_kL_{jk}^{\mathrm{T}},\quad j=1,\ \dots,\ c, \tag{3.4.73}$$

и

$$L_{ij}=\left(C_{ij}-\sum_{k=1}^{j-1}L_{ik}\hat{D}_kL_{jk}^{\mathrm{T}}\right)\left(\hat{D}_jL_{jj}^{\mathrm{T}}\right)^{-1},\quad i>j. \tag{3.4.74}$$

Для точного разложения Холесского вычисления, отвечающие соотношениям (3.4.73), требуют выполнения разложения Холесского для матрицы, равной правой части (3.4.73), чтобы получить множители L_{jj} и $\hat{D}_j$. Однако для разложения IC(0) матрицы A диагональные множители L_{ii} должны быть равны единичным матрицам и, поскольку $\hat{D}_j$ — диагональные матрицы, (3.4.73) принимает вид

$$\hat{D}_j=D_j-\text{диагональная часть}\left(\sum_{k=1}^{j-1}L_{jk}\hat{D}_kL_{jk}^{\mathrm{T}}\right),\quad j=1,\ \dots,\ c. \tag{3.4.75}$$

Так как ненулевые значения допускаются лишь для тех элементов L_{ij}, которые расположены в позициях, отвечающих

диагоналям матрицы C_{ij}, то возникновение ненулевых элементов в других диагоналях в (3.4.74) подавляется:

$$L_{ij} = \left(C_{ij} - \text{диагонали} \left(\sum_{k=1}^{j-1} L_{ik} \hat{D}_k L_{jk}^{\mathrm{T}} \right) \right) \hat{D}_j^{-1}, \quad i > j. \quad (3.4.76)$$

Уравнения (3.4.75) и (3.4.76), обобщающие соотношения (3.4.68), задают факторизацию матрицы A типа IC(0). Заметим, что вычисление этой факторизации требует лишь умножений диагонально разреженных матриц. Заметим также, что в (3.4.75) должны выполняться лишь умножения, необходимые для вычисления диагональных элементов, и, аналогично, в (3.4.76) должны выполняться лишь умножения, необходимые для вычисления элементов, расположенных в позициях, которые соответствуют ненулевым диагоналям матриц C_{ij}.

Треугольные системы (3.4.61) также можно решить с использованием только матричных умножений Система $L\mathbf{z} = \mathbf{r}$ с векторами $\mathbf{z}$ и $\mathbf{r}$, разбиение которых согласовано с разбиением матрицы L, записывается в виде

$$\begin{bmatrix} I & & & \\ L_{21} & \ddots & & \\ \vdots & \ddots & \ddots & \\ L_{c,1} & \cdots & L_{c,c-1} & I \end{bmatrix} \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_c \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_c \end{bmatrix}, \quad (3.4.77)$$

так как диагональные блоки равны единичным матрицам. Таким образом,

$$\mathbf{z}_j = \mathbf{r}_j - \sum_{k=1}^{j-1} L_{jk} \mathbf{z}_k, \quad j = 1, \ldots, c,$$

так что вычисление вектора $\mathbf{z}$ требует только умножений диагонально разреженных матриц на векторы. Система $L^{\mathrm{T}} \tilde{\mathbf{r}} = \mathbf{x}$ решается аналогично.

Рассмотрим теперь вопросы, связанные с реализацией на векторном компьютере. Предположим, что матрица A хранится по диагоналям, то есть хранятся только ненулевые диагонали матриц C_{ij} и D_j. Множитель L хранится так же. Если матрица A возникает при дискретизации двумерной задачи на сетке

$N \times N$, то матрица A имеет размер $N^2 \times N^2$. Таким образом, длины векторов при выполнении неполного разложения и при решении треугольных систем не превосходят $O(N^2/c)$, причем эти длины уменьшаются по мере удаления рассматриваемой диагонали C_{ij} или L_{ij} от главной диагонали. Для большинства задач такое уменьшение не будет чрезмерным, и в качестве ориентировочного значения средней длины вектора мы можем принять $l = O(N^2/(2c))$. Если $N = 100$, то при $c = 2$, т. е. для красно-черного упорядочения, $l = O(2500)$. Для бо́льших значений c величина l оказывается, конечно, меньше, однако, скажем, для $c = 6$ величина l все еще близка к 1000.

Относительно метода сопряженных градиентов можно заметить, что вычисление произведения $A\mathbf{p}$ можно по-прежнему выполнять при помощи длинных векторных операций, если ненулевые диагонали матрицы A проведены сквозь матрицы C_{ij}. Так, например, главная диагональ A имеет длину N^2.

Прием Айзенштата для ICCG

Прием Айзенштата, обсуждавшийся ранее в связи с методом SSOR, можно использовать и совместно с предобусловливанием по методу неполного разложения Холесского. Предположим, что матрица A масштабирована так, что все ее диагональные элементы равны единице. Тогда при использовании красно-черного упорядочения неполные множители Холесского (3.4.68) принимают вид

$$L = \begin{bmatrix} I & 0 \\ C & I \end{bmatrix}, \quad D = \begin{bmatrix} I & 0 \\ 0 & \bar{D}_2 \end{bmatrix},$$

и матрица K в (3.4.31) равна

$$K = L + L^{\mathrm{T}} - \begin{bmatrix} 0 & 0 \\ C & 0 \end{bmatrix} - \begin{bmatrix} 0 & C^{\mathrm{T}} \\ 0 & 0 \end{bmatrix} - I = I.$$

Таким образом, умножение $K\mathbf{t}$ в результате приема Айзенштата «исчезает» и можно получить произведение $A\widehat{\mathbf{p}}^k$, не выполняя практически никакой вычислительной работы сверх той, которая уже была затрачена на предобусловливание. Однако по мере увеличения количества цветов ситуация становится менее выигрышной. В общем случае только первый блочный столбец матрицы K будет нулевым вне главной диагонали, и относительная эффективность приема Айзенштата будет уменьшаться с увеличением количества цветов.

Блочное предобусловливание Якоби и декомпозиция области

В качестве вспомогательного итерационного процесса (3.4.16) можно использовать блочные (или полинейные) итерации Якоби, получая тем самым m-шаговый блочный метод Якоби-PCG (см. упражнение 3.4.4); аналогичную конструкцию можно построить и на основе метода SSOR. Опишем теперь комбинацию блочного предобусловливания Якоби с методом декомпозиции области. Предположим, что для области, на которой задано уравнение в частных производных, проведена декомпозиция (см., например, рис. 3.1.14). Как и в § 3.1, предположим, что каждая точка сетки принадлежит некоторой подобласти; таким образом, множества-разделители отсутствуют. Если $A = (A_{ij})$ — блочная форма матрицы коэффициентов, отвечающая принятой декомпозиции области, и количество подобластей равно p, то матрица-предобусловливатель для одношагового блочного метода Якоби-PCG равна $M = \mathrm{diag}(A_{11}, \ldots, A_{pp})$. Мы можем интерпретировать это блочное предобусловливание Якоби как некоторую простую разновидность предобусловливания посредством декомпозиции области, где предобусловливание выполняется путем решения задач в подобластях при значениях неизвестных в соседних подобластях, взятых с предыдущей итерации метода сопряженных градиентов.

Подобный подход к предобусловливанию потенциально обладает исключительно хорошими параллельными и векторными свойствами,— при условии, что рассматриваемая задача имеет подходящую структуру. Например, как было показано в § 3.1, вычисления, требуемые для выполнения блочной итерации Якоби, могут быть выполнены при помощи сквозной векторизации множества систем. При этом предполагается, что все матрицы A_{ii} имеют один и тот же размер и одну и ту же структуру (например, трехдиагональную). Для параллельных вычислений подобласти нужно выбирать таким образом, чтобы вычислительные затраты на решение задач в подобластях были сбалансированы.

Хотя одношаговый блочный метод Якоби — сопряженных градиентов имеет хорошие параллельные свойства, скорость сходимости итераций может оказаться досадно низкой. Например, для дискретного уравнения Пуассона и одношагового линейного предобусловливателя Якоби число обусловленности уменьшается (по сравнению с поточечным методом Якоби) всего лишь в два раза (упражнение 3.4.11). Поэтому может оказаться полезным использование подобластей другого вида, например квадратов, как показано на рис. 3.1.14. Но в этом

случае приходится решать в каждой из подобластей задачу того же типа, что и исходная. Таким образом, если исходная сетка имеет размеры, скажем, 1000×1000 и используется 16 подобластей (рис. 3.1.14), то каждая подобласть будет иметь размеры 250×250, и решение таких задач может потребовать очень больших затрат. Один из подходов к возникающей проблеме заключается в приближенном решении задач в подобластях при помощи неполной факторизации $L_i L_i^{\mathrm{T}}$ блоков A_{ii}. При этом на шаге предобусловливания метода PCG решаются системы

$$L_i \mathbf{x}_i = \mathbf{r}_i, \quad L_i^{\mathrm{T}} \tilde{\mathbf{r}}_i = \mathbf{x}_i, \quad i = 1, \ldots, p,$$

причем каждый процессор решает свою пару систем.

Упражнения к параграфу 3.4

1. Показать, что вектор $\tilde{\mathbf{r}}$, определенный соотношением (3.4.15), совпадает с вектором $\mathbf{r}^m$ из (3.4.16) при $\mathbf{r}^0 = 0$

2. Для метода SOR показать, что матрица M в (3.4.14) является, вообще говоря, несимметричной при всех $m \geqslant 1$ для любого $0 < \omega < 2$

3. Пусть B и C — симметричные матрицы $n \times n$. Показать, что произведения $B(CB)^m$ представляют собой симметричные матрицы для любого положительного целого m.

4. Сформулировать m-шаговый блочный метод Якоби-PCG, аналогичный поточечному методу (3.4.17)

5. Показать, что если матрица D положительно определена и $0 < \omega < 2$, то матрицы P и Q в (3.4.19) являются симметричными и положительно определенными, за исключением случая $\omega = 1$, когда матрица Q оказывается лишь положительно полуопределенной

6. Убедиться в справедливости формул (3.4.21).

7. Показать, что если a, b, c и d — такие вещественные числа, что $cb < ad$, то

$$\frac{a+c}{b+d} < \frac{a}{b}.$$

Использовать эту оценку для доказательства неравенства в (3.4.23) при $0 \leqslant \lambda_1 \leqslant \lambda_n$.

8. В предположениях теоремы 3.4.6 показать, что собственные значения матрицы перехода метода SSOR неотрицательны при $\omega = 1$.

9. Показать, что предобусловленный метод сопряженных градиентов, примененный к системе $A\mathbf{x} = \mathbf{b}$ с матрицей-предобусловливателем M, заданной в (3.4.27), эквивалентен методу PCG, примененному к системе $\hat{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}$, определенной в (3.4.29), с матрицей-предобусловливателем T^{-1}.

10. Предположим, что к решению системы $A\mathbf{x} = \mathbf{b}$ с матрицей A, заданной в (3.1.7), применяется одношаговый метод Якоби-PCG. Показать, что скорость сходимости итераций будет такой же, как если бы предобусловливание не применялось. (Указание: сначала покажите, что получающийся итера-

ционный метод совпадает с методом CG без предобусловливания, примененным к системе $\hat{A}\mathbf{x} = \hat{\mathbf{b}}$, где D — диагональная часть матрицы A и $\hat{A} = D^{-1}A$, а $\hat{\mathbf{b}} = D^{-1}\mathbf{b}$; затем покажите, что, поскольку матрица D равна единичной матрице, умноженной на константу, то числа обусловленности матриц A и $\hat{A}$ совпадают.)

11. Рассмотрим одношаговый блочный метод Якоби-PCG для системы $A\mathbf{x} = \mathbf{b}$ в случае, когда A — матрица (3.1.7), отвечающая дискретному уравнению Пуассона. Тогда

$$B = M^{-1}A = \begin{bmatrix} I & -T^{-1} & & & \\ -T^{-1} & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -T^{-1} \\ & & & -T^{-1} & I \end{bmatrix} = I + C,$$

где C является кронекеровским произведением вида $(T - 4I) \otimes T^{-1}$. Используя упражнение 3.1.19, показать, что собственные значения матриц T^{-1} и $T - 4I$ равны $(4 - 2\cos kh)^{-1}$ и $-2\cos kh$, $k = 1, \ldots, N$, соответственно, где $h = \pi/(N+1)$. Затем использовать тот факт, что (см., например, [Ortega, 1987a, p. 237]) собственные значения матрицы $(T - 4I) \otimes T^{-1}$ равны всевозможным произведениям собственных значений матриц $T - 4I$ и T^{-1}, чтобы установить формулы для собственных значений матрицы B вида

$$\frac{2\cos jh}{-4 + 2\cos kh}, \qquad j, k = 1, \ldots, N.$$

Вывести отсюда, что число обусловленности $\mathrm{cond}(B)$, т. е. отношение наибольшего собственного значения к наименьшему, равно

$$\mathrm{cond}(B) = 1/(1 - \cos h) \approx 2h^{-2},$$

где приближенное равенство справедливо для малых значений h. Показать, что число обусловленности предобусловленной матрицы при $h \to 0$ лишь в два раза меньше числа обусловленности исходной матрицы A.

12. Проверить в деталях вывод соотношений (3.4.26).

13. Показать, что в результате применения алгоритма, приведенного на рис. 3.4.1, к трехдиагональной матрице получается ее точное разложение Холесского. Вообще, показать, что если указанный алгоритм применяется к ленточной матрице, не содержащей нулевых элементов внутри ленты, то в результате получается точное разложение Холесского.

14. Пусть

$$A = \begin{bmatrix} A_1 & C_1 & & & \\ C_1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & C_{N-1} \\ & & & C_{N-1} & A_N \end{bmatrix},$$

$$A_i = \begin{bmatrix} A_{i1} & B_{i1} & & & \\ B_{i1} & \cdot & \cdot & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & B_{i,N-1} \\ & & & B_{i,N-1} & A_{i,N} \end{bmatrix},$$

где каждый блок A_{il} представляет собой трехдиагональную матрицу $N \times N$, блок B_{ij} — диагональную матрицу $N \times N$, а каждая матрица C_i является диагональной и имеет размер $N^2 \times N^2$. Таким образом, A — семидиагональная матрица $N^3 \times N^3$. Показать, что множители L и D неполного разложения IC(0) удовлетворяют соотношениям

$$l_{i,i-1} = a_{i,i-1}/d_{i-1},$$
$$l_{i,i-N} = a_{i,i-N}/d_{i-N},$$
$$l_{i,i-N^2} = a_{i,i-N^2}/d_{i-N^2},$$
$$d_i = a_{ii} - l^2_{i,i-N^2}d_{i-N^2} - l^2_{i,i-N}d_{i-N} - l^2_{i,i-1}d_{i-1}.$$

15. Предположим, что матрицы B_i в (3.4.44) являются трехдиагональными. Показать, что в этом случае соотношение (3.4.46) уже не имеет места.

16 Пусть LDL^{T}, где $\mathrm{diag}(L) = I$, является свободным от корней разложением Холесского (полным или неполным), и пусть то же самое разложение представлено в виде $L_1D_1L_1^{\mathrm{T}}$, где $\mathrm{diag}\,(L_1) = D_1^{-1}$. Показать, что $D_1 = D^{-1}$ и $LD^{1/2} = L_1D_1^{1/2}$. Используя полученные соотношения, убедиться в том, что неполное разложение матрицы (3.4.44) можно представить в виде (3.4 49), где поддиагональные элементы матрицы L_1 равны соответствующим поддиагональным элементам матрицы A.

17. Пусть $L_1D_1L_1^{\mathrm{T}}$ — разложение, описанное в упражнении 16. Показать, что соответствующее разложение $\widehat{L}_1\widehat{D}_1\widehat{L}_1^{\mathrm{T}}$ матрицы $\widehat{A} = D_1^{1/2}AD_1^{1/2}$ обладает свойством $\widehat{D}_1 = I$.

18. Убедиться в том, что формулы (3.4.50)—(3.4.52) определяют точное блочное разложение Холесского матрицы A, определенной в (3.4.44).

19. Показать, что первые две диагонали матрицы L^{-1}, соответствующей (3 4.56), задаются соотношениями (3.4 57).

20. Предположим, что матрица (3 4 44) образована M^2 блоками размера $N \times N$, т. е. матрица A имеет размер $MN \times MN$. Показать, что по схеме, приведенной на рис. 3 4.4, обработка M-го блока начинается на шаге $2M - 1$ и решение вычисляется полностью на шаге $2M + N - 2$, так что средняя степень параллелизма оказывается равной $MN/(2M + N - 2)$.

21. Показать, что если L — нижнетреугольная матрица, а LL^{T} — диагональная, то L — также диагональная матрица Вывести отсюда, что первое из соотношений (3.4 64) влечет за собой равенства $L_1 = I$ и $\widehat{D}_1 = D_1$.

22. Убедиться в справедливости соотношений (3.4.73) и (3.4.74).

Литература и дополнения к параграфу 3.4

1. Хотя сама идея предобусловливания восходит к работе [Hestenes, 1956], основная часть исследований в этой области была выполнена после того, как в начале 70-х годов было осознано потенциальное значение метода сопряженных градиентов как итерационного метода. Основополагающей работой по предобусловливанию является [Concus, Golub, O'Leary, 1976]; см. также статью [Axelsson, 1976].

2. Подход к предобусловливанию для параллельных и векторных компьютеров, основанный на использовании усеченного разложения в ряд, впервые был применен в работе [Dubois et al., 1979] в случае, когда матрица P в (3.4.15) равна диагональной части матрицы A, иначе говоря, для той ситуации, когда в качестве вспомогательного итерационного процесса используется метод Якоби Затем в статье [Johnson et al, 1983] было рассмотрено полиномиальное предобусловливание, опять для случая, когда P представляет собой диагональную часть матрицы A. В работах [Adams, 1982, 1983, 1985] предобусловливание при помощи усеченных рядов изучалось в контексте использования m шагов вспомогательного итерационного метода, включая, в частности, методы Якоби и SSOR. Теоремы 3.4.2, 3.4.5 и 3.4.6 основываются на результатах этой работы Там же были рассмотрены полиномиальные предобусловливания для матриц P более общего вида, возникающих, в частности, при использовании метода SSOR, таблица 3.4.2 заимствована из этих работ. См. также работу [Saad, 1985], где можно найти общий анализ методов полиномиального предобусловливания, и работу [Chen, 1982], где дается детальное исследование способов масштабирования с приложениями к полиномиальному предобусловливанию. В статье [Vaughan, Ortega, 1987] описана реализация метода SSOR-PCG на гиперкубическом мультипроцессоре Intel iPSC

3 Прием Айзенштата описан в работе [Eisenstat, 1981] Родственная идея обсуждается в работе [Bank, Douglas, 1985]. Анализ и сравнение процедур Бэнка — Дугласа, Айзенштата и Конрада — Валяха приводится в работе [Ortega, 1987c], где рассматриваются как стационарные итерационные методы, так и метод сопряженных градиентов

4 Идея неполной факторизации восходит по крайней мере к работе [Varga, 1960]. В статье [Meijerink, van der Vorst, 1977] дается детальное изложение метода ICCG, включая формулировку метода IC(h, k). Понятие регулярной неполной факторизации было введено в работе [Robert, 1982]. Неполная «встречная» факторизация [1]) рассматривается в работе [van der Vorst, 1987].

5. Теорема 3.4.7 была доказана для M-матриц в [Meijerink, van der Vorst, 1977] и распространена на H-матрицы в [Manteuffel 1980]. В последней работе обсуждается также стратегия сдвига, причем отмечается, что не всегда обязательно выбирать значения α настолько большим, чтобы матрица $\alpha I + A$ имела диагональное преобладание

6. В [Kershaw, 1978] предложена стратегия, обеспечивающая диагональное преобладание в треугольном сомножителе неполного разложения путем замены текущего значения l_{ii} на величину $\sum |l_{ij}|$. Сходная идея выдвинута в работе [Gustafsson, 1978], где предложен так называемый модифицированный

[1]) В оригинале incomplete «twisted» factorization; структурно этот алгоритм близок к методу встречной прогонки для решения трехдиагональных систем. — *Прим. перев.*

метод ICCG (сокращенно MICCG). Можно также построить семейство модифицированных методов, зависящее от параметра α, изменяющегося от нуля (что соответствует исходному методу ICCG) до единицы (что дает метод MICCG). Выбор значения α, немного меньшего единицы, позволяет добиться многократного улучшения скорости сходимости итераций. Более подробно с этим подходом можно ознакомиться в работах [Ashcraft, Grimes, 1987] и [Axelsson, Lindskog, 1986].

7. Изложение неполной факторизации (3.4.45)—(3.4.48) матрицы (3.4.44) основано на работе [van der Vorst, 1982], где также дается соотношение (3.4.49).

8. При обсуждении блочных методов (3.4.50) — (3.4.57) мы следуем подходу, принятому в [Concus, Golub, Meurant, 1985]. В работе [Meurant, 1984] рассматриваются соответствующие векторизованные алгоритмы. Большое количество статей посвящено исследованию различных неполных блочных факторизаций, как правило, в предположении, что матрица A блочно-трехдиагональная, см. например, [Axelsson, 1985, 1986], [Eisenstat et al., 1984], [Jordan T., 1982], [Kershaw, 1982], [Reiter, Rodrigue, 1984], [Rodrigue, Wolitzer, 1984]. В последних четырех статьях основное внимание уделяется построению эффективных алгоритмов для векторных компьютеров типа CRAY. См. также работу [Axelsson, Polman, 1986].

9. Подход к решению системы $L\mathbf{x} = \mathbf{r}$, проиллюстрированный рис. 3.4.4, принадлежит ван дер Ворсту [van der Vorst, 1983], а его обобщение описано в работе [Ashcraft, 1985a].

10. Использование многоцветных раскрашиваний для ICCG было предложено в работе [Schreiber, Tang, 1982]. Дальнейшее развитие этого подхода можно найти в [Poole, 1986] и [Poole, Ortega, 1987]. Применению этой идеи к предобусловливанию как по методу неполного разложения Холесского, так и по методу SSOR посвящена работа [Nodera, 1984]. Хотя многоцветное раскрашивание позволяет достичь очень высокой степени параллелизма, его применение может повлечь за собой уменьшение скорости сходимости итераций по сравнению с естественным упорядочением [1]).

11. Использование неполной факторизации для аппроксимации одношагового блочного метода Якоби-PCG предложено в работе [Ashcraft, 1987b], где приводятся также различные экспериментальные результаты. Блочное предобусловливание Якоби изучается также в работе [Kowalik, Kumar, 1982]. Мы отождествили такое блочное предобусловливание Якоби с методом декомпозиции области, однако в большинстве работ по методу декомпозиции области используются множества-разделители и рассмотрение начинается с уравнений с матрицей, приведенной к диагонально-окаймленной форме (3.3.18). В этом случае открываются две основные возможности: метод PCG применяется либо непосредственно к системе (3.3.18), либо к редуцированной системе $\hat{A}\mathbf{x}_s = \hat{\mathbf{b}}$ с матрицей, равной дополнению Шура, где $\hat{A}$ и $\hat{\mathbf{b}}$ определены в (3.3.19). Очень хороший обзор этих двух подходов и другие указания на литературу можно найти в статье [Keyes, Gropp, 1987]. Там же сообщается и об экспериментальных расчетах, выполненных на гиперкубическом мультипроцессоре Intell iPSC. Среди других работ посвященных методу декомпозиции области, можно назвать [Lichnewsky, 1984], где используется многоцветное раскрашивание внутри подобластей; [Gonzalez, Wheeler, 1987], где рассматривается обобщенное уравнение Пуассона с краевыми условиями Неймана; [Nour-Omid, Park, 1987], где сообщается об экспериментальных расчетах плоской задачи

[1]) Прежде всего это относится к красно-черному упорядочению. — *Прим. перев.*

о напряжениях на гиперкубическом мультипроцессоре; а также [Rodrigue, 1986], где обсуждается декомпозиция области типа процесса Шварца.

12 В литературе рассматривались и другие подходы к предобусловливанию В статье [McBryan, van de Velde, 1985] в качестве предобусловливателей рассматривались процедуры быстрого решения дискретного уравнения Пуассона, основанные на использовании быстрого преобразования Фурье, а также итерации метода переменных направлений ADI В работе [Nour-Omid, Parlett, 1987] использовалось «поэлементное предобусловливание», которое, возможно, окажется полезным инструментом решения конечноэлементных задач

13 В ряде других работ сообщается о численных результатах и об экспериментальном сравнении различных методов. В статье [Kightley, Jones, 1985] приводятся результаты вычислений на компьютере CRAY-1 для некоторых предобусловливаний, примененных к четырем трехмерным тестовым задачам. Две из них представляют собой уравнение Пуассона, в одном случае с краевыми условиями Дирихле, а в другом — с краевыми условиями Неймана. Интересно, что для этих двух задач простое диагональное масштабирование оказалось наилучшим из всех способов предобусловливания в смысле общих затрат времени Две другие тестовые задачи были взяты из гидродинамики Для одной из них диагональное масштабирование тоже работало очень хорошо, уступая лишь одной из версий метода ICCG, и то лишь процентов на десять. В работе [Knightley, Thompson, 1987] проводится сравнение ряда многосеточных программ с методом ICCG на компьютере CRAY-1. В качестве тестовых задач используются двумерные дискретные задачи Дирихле для уравнения Пуассона в квадратной области Многосеточные методы продемонстрировали на этих задачах лучшую производительность. В работе [Chan, 1985] проводится сравнение предобусловливания SSOR с многосеточным методом и с прямым методом для разреженных систем при решении двумерных задач. В статьях [Seager, 1986a, b] оцениваются накладные затраты, возникающие при крупноблочном и при мелкозернистом распараллеливании предобусловленного метода сопряженных градиентов на четырехпроцессорном компьютере CRAY X-MP

14. Как мы видели в § 3.3, метод сопряженных градиентов можно интерпретировать как метод минимизации и, таким образом, его область применимости фактически оказывается ограниченной симметричными задачами. Однако было предпринято большое число попыток обобщить этот метод на случай несимметричных систем уравнений и, в частности, разработать такие методы, которые по возможности сохраняли бы достоинства метода сопряженных градиентов Некоторые из этих методов были опробованы на векторных компьютерах Например, в работе [Oppe, Kincaid, 1987] сообщается о тестовых расчетах пяти модельных задач, возникающих при моделировании подземных месторождений, с помощью различных предобусловленных методов типа сопряженных градиентов. Из этих пяти задач только одна имела симметричную положительно определенную матрицу коэффициентов; две другие задачи были «почти симметричны», а последние две — сильно несимметричны В качестве предобусловливателей рассматривались точечный и блочный методы Якоби, метод SSOR, полиномиальный метод, неполное разложение Холесского и модифицированное неполное разложение Холесского. Применялись также некоторые полуитерационные методы. Прогоны соответствующих программ выполнялись на компьютерах CYBER 205 и CRAY X-MP. См. также статьи [Kincaid et al., 1986a, b].

Приложение 1

LU-разложение и разложение Холесского: ijk-формы

В этом приложении приводятся некоторые подробности об обсуждавшихся в гл. 2 ijk-формах LU-разложения и разложения Холесского. Общая основа этих форм — тройка вложенных циклов Для, показанная на рис. П.1.1.

Мы будем считать, что A допускает устойчивое LU-разложение, так что выбор главного элемента не нужен. Разложение Холесского будем записывать как в обычной форме LL^{T}, так

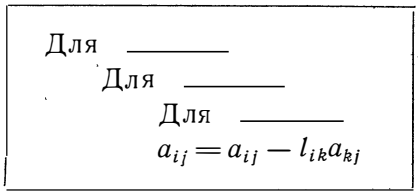


Рис. П.1.1. Тройка вложенных циклов для LU-разложения

и в форме LDL^{T}, отвечающей схеме без квадратных корней, или симметризованной форме гауссова исключения SGE (Symmetric Gaussian Elimination. — *Перев.*); в последнем случае L — нижнетреугольная матрица с единицами на главной диагонали. Форма LDL^{T} позволяет естественно перейти от LU-разложения к разложению Холесского. В случае применения метода Холесского или метода SGE будем предполагать, что хранится только нижний треугольник матрицы A, который постепенно замещается элементами матрицы L (а также матрицы D, если рассматривается метод SGE).

Векторные компьютеры

Псевдокоды для kij- и kji-форм LU-разложения приведены на рис. 2.1.1 и 2.1.2. Математически обе формы можно задать с помощью вычитания *внешних произведений*, поэтому иногда их называют *LU-разложением в форме внешних произведений.* Сказанное подтверждается следующим представлением первого

шага LU-разложения:

$$A=\begin{bmatrix} a_{11} & \mathbf{a}_1^{\mathrm{T}} \\ \hat{\mathbf{a}}_1 & A_2 \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ \mathbf{l}_1 & I \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & \hat{A}_2 \end{bmatrix}\begin{bmatrix} a_{11} & \mathbf{u}_1 \\ 0 & I \end{bmatrix}=\begin{bmatrix} a_{11} & \mathbf{u}_1 \\ a_{11}\mathbf{l}_1 & \hat{A}_2+\mathbf{l}_1\mathbf{u}_1 \end{bmatrix}$$

Таким образом, модификация подматрицы A_2 реализуется (математически) путем вычитания внешнего произведения $\mathbf{l}_1\mathbf{u}_1$ первого столбца L и первой строки U. В обеих формах kij и kji матрица L вычисляется по столбцам, а матрица U — по строкам, т. е. на k-м шаге определяются k-й столбец в L и $(k+1)$-я строка в U (первая строка матрицы U совпадает с первой строкой A). Однако конкретный способ вычислений в этих двух алгоритмах различен. В форме kij строки U вычисляются посредством векторных операций, а элементы L — посредством скалярных операций. В форме kji столбцы L могут вычисляться с помощью векторных операций, тогда как элементы строк матрицы U вычисляются по одному в результате векторных операций над столбцами.

Формы *kij* и *kji* алгоритмов Холесского и SGE

Псевдокоды двух форм алгоритма SGE приведены на рис. П.1.2. В скобках указаны изменения, необходимые для того, чтобы превратить эти тексты в псевдокоды соответствующих форм алгоритма Холесского.

$(l_{11}=a_{11}^{1/2})$	$(l_{11}=a_{11}^{1/2})$
Для $k=1$ до $n-1$	Для $k=1$ до $n-1$
Для $i=k+1$ до n	Для $s=k+1$ до n
$l_{ik}=a_{ik}/a_{kk}\ (a_{ik}/l_{kk})$	$l_{sk}=a_{sk}/a_{kk}\ (a_{sk}/l_{kk})$
Для $j=k+1$ до i	Для $j=k+1$ до n
	Для $i=j$ до n
$a_{ij}=a_{ij}-l_{ik}a_{jk}$	$a_{ij}=a_{ij}-l_{ik}a_{jk}$
$(a_{ij}-l_{ik}l_{jk})$	$(a_{ij}-l_{ik}l_{jk})$
$(l_{k+1,\,k+1}=a_{k+1,\,k+1}^{1/2})$	$(l_{k+1,\,k+1}=a_{k+1,\,k+1}^{1/2})$

Рис. П.1.2. Формы kij и kji SGE-алгоритма и алгоритма Холесского: a) kij; b) kji

В обеих программах предполагается, что хранится только нижняя половина матрицы A. Поэтому основная формула модификации $a_{ij}=a_{ij}-l_{ik}a_{kj}$, использовавшаяся в LU-разложении при хранении матрицы полностью, заменяется в алгоритме

SGE на $a_{ij} = a_{ij} - l_{ik}a_{jk}$. В программах, реализующих метод Холесского, место a_{jk} в этом основном арифметическом операторе занимает l_{jk}. Хотя в программах это явно не указано, в методе Холесского элементы матрицы L могут замещать соответствующие элементы матрицы A. Однако в методе SGE такие замещения приходится иногда откладывать до момента, когда первоначальные значения элементов A становятся ненужными для дальнейших вычислений. Различные ijk-формы метода SGE имеют в этом отношении разные свойства. В форме kji на рис. П.1.2 весь k-й столбец матрицы A должен быть использован для модификаций, прежде чем можно будет записать на его место k-й столбец матрицы L. Отметим, что в то время как в программе метода Холесского матрица L хранится полностью, программа метода SGE хранит только ее нижнюю *строго* треугольную часть; в конце разложения диагональ матрицы A превращается в диагональ D. Независимо от того, когда происходит замещение элементов матрицы A элементами L, мы будем считать, что L и A имеют одну и ту же структуру хранения. Таким образом, во внутреннем цикле формы kij алгоритма SGE i-я строка матрицы A модифицируется с помощью k-го столбца этой же матрицы, а в программе метода Холесского используются i-я строка в A и k-й столбец в L. В обоих случаях нужен доступ и к строкам, и к столбцам. Для форм kji требуется доступ только к столбцам матриц A и L.

Как и в LU-разложении с полным хранением матрицы, формы kij и kji методов SGE и Холесского являются алгоритмами безотлагательных модификаций. Векторная команда внутреннего цикла во всех четырех случаях — это триада, хотя средние длины векторов теперь короче, чем в LU-разложении. Как указано в (2.1.17), средняя длина вектора для каждого из четырех алгоритмов равна $O(n/3)$.

Формы kij и kji снова можно интерпретировать как алгоритмы внешних произведений. Например, первый шаг метода Холесского допускает представление

$$A = \begin{bmatrix} a_{11} & \mathbf{a}_1^{\mathrm{T}} \\ \mathbf{a}_1 & A_2 \end{bmatrix} = \begin{bmatrix} a_{11}^{1/2} & 0 \\ \mathbf{l}_1 & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \hat{A}_2 \end{bmatrix} \begin{bmatrix} a_{11}^{1/2} & \mathbf{l}_1^{\mathrm{T}} \\ 0 & I \end{bmatrix}.$$

Таким образом, модифицированная подматрица $\hat{A}_2$ получается из A_2 путем вычитания внешнего произведения $\mathbf{l}_1\mathbf{l}_1^{\mathrm{T}}$. В книге [George, Liu, 1981] формы kij и kji названы *подматричным алгоритмом Холесского*. В обеих этих формах вычисляется по одному столбцу матрицы L за шаг. Для псевдокодов, представленных на рис. П.1.2, характер доступа к данным показан на рис. 2.1.11.

Отложенные модификации: формы *ikj* и *jki* *LU*-разложения

Форма *ikj* *LU*-разложения аналогична форме *kij* с одним исключением: модификации последующих строк матрицы A откладываются до тех пор, пока эти строки не подвергнутся окончательной обработке. Так же соотносятся формы *jki* и *kji*; поэтому формы *ikj* и *jki* называются *алгоритмами с отложенными модификациями*. Их псевдокоды приведены на рис. П.1.3. (Напомним, что псевдокод для формы *jki* показан и на рис. 2.1.3.)

```
Для i = 2 до n                    Для j = 2 до n
    Для k = 1 до i − 1                Для s = j до n
        l_ik = a_ik/a_kk                  l_{s,j−1} = a_{s,j−1}/a_{j−1,j−1}
        Для j = k + 1 до n            Для k = 1 до j − 1
                                          Для i = k + 1 до n
        a_ij = a_ij − l_ik a_kj               a_ij = a_ij − l_ik a_kj
```

Рис. П.1.3. Формы *ikj* и *jki* *LU*-разложения a) *ikj*, b) *jki*

На i-м шаге алгоритма *ikj* выполняются все необходимые модификации i-й строки матрицы A; в результате получаются i-е строки матриц L и U. Таким образом, i-я строка матрицы U вычисляется как линейная комбинация ее предыдущих строк:

$$\mathbf{u}_i = \sum_{k=1}^{i-1} l_{ik}\mathbf{u}_k$$

(используются лишь те позиции, которые войдут в i-ю строку U). Во внутреннем цикле требуется доступ к строкам матрицы A, а элементы матрицы L дают коэффициенты линейных комбинаций. Форма *jki* представляет собой соответствующий столбцово ориентированный алгоритм. В ней нужен доступ к столбцам матриц A и L. Основной операцией снова является вычисление линейных комбинаций, только роль коэффициентов в них теперь играют элементы a_{kj}. На j-м шаге после внесения требуемых изменений в матрицу A будут получены j-е столбцы матриц L и U (в действительности, согласно тексту на рис. П.1.3b, j-й столбец L будет вычислен на $(j+1)$-шаге. — *Перев.*). Характер доступа к данным для форм *ikj* и *jki* показан на рис. 2.1.4. Заметим, что средняя длина векторов в операциях внутреннего цикла этих двух форм снова равна $O\left(\frac{2}{3}n\right)$.

Формы *ikj* и *jki* алгоритмов SGE и Холесского

На рис. П.1.4 приведены псевдокоды этих двух форм; характер доступа к данным показан на рис. 2.1.11.

В алгоритмах *ikj* и *jki* основной операцией снова является вычисление линейных комбинаций, и средняя длина векторов равна $O\left(\frac{1}{3}n\right)$. Если речь идет о форме *ikj* алгоритма Холесского, то во внутреннем цикле модифицируются элементы i-й строки матрицы A, при этом используются столбцы матрицы L; таким образом, нужен доступ и к строкам, и к столбцам. То же

```
(l11 = a11^1/2)                    (l11 = a11^1/2)
Для i = 2 до n                     Для j = 2 до n
    Для k = 1 до i − 1                 Для s = j до n
        lik = aik/akk (aik/lkk)            ls, j−1 = as, j−1/aj−1, j−1 (as, j−1/lj−1, j−1)
        Для j = k + 1 до i             Для k = 1 до j − 1
                                           Для i = j до n
            aij = aij − lik ajk                aij = aij − lik ajk
                  (aij − lik ljk)                    (aij − lik ljk)
    (lii = a^1/2 ii)               (ljj = a^1/2 jj)
```

Рис. П.1.4. Формы *ikj* и *jki* SGE-алгоритма и алгоритма Холесского: a) *ikj*; b) *jki*

самое справедливо в отношении алгоритма SGE, хотя здесь требуются строки матрицы A, хранимые вследствие симметрии как столбцы (т. е. вместо столбцов L, использовавшихся в алгоритме Холесского, теперь берутся столбцы A. — *Перев.*). Заметим, что в алгоритме SGE нельзя записать L на место A, поскольку для будущих модификаций необходимы элементы именно исходной матрицы A. Матрица L вычисляется по строкам, и данная форма представляет собой один из вариантов метода, названного в книге [George, Liu, 1981] *строчным алгоритмом Холесского*. В основе алгоритма лежит идея окаймления; это значит, что если $A_{i-1} = \hat{L}_{i-1}\hat{L}_{i-1}^{\mathrm{T}}$ есть разложение $(i-1)$-й главной подматрицы, то разложение подматрицы A_i имеет вид

$$A_i = \begin{bmatrix} A_{i-1} & \mathbf{a}_i \\ \mathbf{a}_i^{\mathrm{T}} & a_{ii} \end{bmatrix} = \begin{bmatrix} \hat{L}_{i-1} & 0 \\ \mathbf{l}_i^{\mathrm{T}} & l_{ii} \end{bmatrix} \begin{bmatrix} \hat{L}_{i-1}^{\mathrm{T}} & \mathbf{l}_i \\ 0 & l_{ii} \end{bmatrix}.$$

Таким образом, очередная строка матрицы L вычисляется с помощью соотношений $\hat{L}_{i-1}\mathbf{l}_i = \mathbf{a}_i$, $l_{ii} = (a_{ii} - \mathbf{l}_i^{\mathrm{T}}\mathbf{l}_i)^{1/2}$.

Для формы *jki* нужен доступ к столбцам матриц A и L. В алгоритме SGE элементы a_{jk} играют роль коэффициентов линейных комбинаций; они принадлежат j-й строке матрицы A, а часть этой матрицы, лежащая выше j-й строки, может быть замещена матрицей L. Столбцы L вычисляются по одному за шаг, и данная форма представляет собой один из вариантов метода, названного в книге [George, Liu, 1981] *столбцовым алгоритмом Холесского.*

Формы *ijk* и *jik* *LU*-разложения

Эти формы основаны на явном LU-разложении с использованием скалярных произведений. Их псевдокоды приведены на рис. П.1.5. Отметим, что в форме *ijk* цикл j состоит из двух частей; в первой вычисляются элементы i-й строки матрицы L,

```
Для i = 2 до n                          Для j = 2 до n
    Для j = 2 до i                          Для s = j до n
        l_{i,j-1} = a_{i,j-1}/a_{j-1,j-1}       l_{s,j-1} = a_{s,j-1}/a_{j-1,j-1}
        Для k = 1 до j - 1                  Для i = 2 до j
            a_ij = a_ij - l_ik a_kj             Для k = 1 до i - 1
    Для j = i + 1 до n                              a_ij = a_ij - l_ik a_kj
        Для k = 1 до i - 1                  Для i = j + 1 до n
                                                Для k = 1 до j - 1
            a_ij = a_ij - l_ik a_kj                 a_ij = a_ij - l_ik a_kj
```

Рис. П.1.5. Формы *ijk* и *jik* LU-разложения: a) *ijk*; b) *jik*

во второй — элементы i-й строки матрицы U. Вычисления внутреннего цикла представляют собой скалярные произведения i-й строки L и столбцов A. Аналогично, в первом цикле i формы *jik* вычисляется j-й столбец матрицы U, а во втором цикле i модифицируется оставшаяся часть j-го столбца матрицы A. Основной векторной операцией является скалярное произведение; его сомножители — строки матрицы L и j-й столбец матрицы U. Характер доступа к данным для обеих форм показан на рис. 2.1.4. Заметим, что все необходимые модификации производятся на шаге, на котором вычисляется текущая строка (или столбец); таким образом, данные формы тоже принадлежат к числу алгоритмов с отложенными модификациями. В форме *ijk* матрицы L и U вычисляются по строкам, а в форме *jik* — по столбцам. Наконец, средние длины векторов в скалярных произведениях снова равны $O\left(\frac{2}{3}n\right)$.

Формы *ijk* и *jik* алгоритмов SGE и Холесского

Псевдокоды этих форм приведены на рис. П.1.6, а характер доступа к данным показан на рис. 2.1.11.

Внутренний цикл формы *ijk* — это вычисление скалярного произведения i-й и j-й строк матрицы L в случае метода Холесского и i-й строки L и j-й строки A в случае метода SGE. Поскольку строки L вычисляются по одной за шаг, то форма

```
(l_11 = a_11^1/2)
Для i = 2 до n
    Для j = 2 до i
        l_i,j-1 = a_i,j-1/a_j-1,j-1
                  (a_i,j-1/l_j-1,j-1)
    Для k = 1 до j - 1
        a_ij = a_ij - l_ik a_jk
               (a_ij - l_ik l_jk)
    (l_ii = a_ii^1/2)
```

```
(l_11 = a_11^1/2)
Для j = 2 до n
    Для s = j до n
        l_s,j-1 = a_s,j-1/a_j-1,j-1
                  (a_s,j-1/l_j-1,j-1)
    Для i = j до n
        Для k = 1 до j - 1
            a_ij = a_ij - l_ik a_jk
                   (a_ij - l_ik l_jk)
        (l_jj = a_jj^1/2)
```

Рис. П.1.6. Формы *ijk* и *jik* алгоритмов SGE и Холесского: a) *ijk*; b) *jik*

ijk метода Холесского представляет собой еще один вариант строчного алгоритма. Отметим, что в методе SGE для модификаций i-го шага нужны все предыдущие строки матрицы A, поэтому нельзя записать L на место A. В форме *jik* внутренний цикл — это скалярное произведение i-й строки L и либо j-й строки L, либо j-й строки A, т. е. совпадает с внутренним циклом формы *ijk*. Однако L вычисляется по столбцам, на каждом шаге — очередной столбец, поэтому форма *jik* представляет собой еще один вариант столбцового алгоритма Холесского. Заметим, что в методе SGE j-я строка матрицы A нужна для j-го шага, поэтому L может замещать A только в позициях, находящихся выше j-й строки. Обе формы — *ijk* и *jik* — представляют собой алгоритмы с отложенными модификациями, а средняя длина векторов в скалярных произведениях равна $O\left(\frac{1}{3}n\right)$.

Параллельные компьютеры

Теперь мы обсудим *ijk*-формы для параллельных систем. Составляющие такую систему p процессоров могут обладать, но могут и не обладать векторными свойствами. Мы рассмотрим

только случай систем с локальной памятью и будем использовать. для хранения матрицы циклическую слоистую схему, строчную или столбцовую. В дальнейшем P_i обозначает i-й процессор системы, а $P(i)$ — процессор, хранящий i-ю строку (или i-й столбец) матрицы A. Начнем с LU-разложения.

LU-разложение

Основные программы для форм *kij* и *kji* LU-разложения приведены на рис. 2.1.1 и 2.1.2, но, чтобы быть полезными для параллельных систем, они нуждаются в соответствующей интерпретации. Рассмотрим для начала программу формы *kij*. Будучи ориентированной на векторную (или последовательную) машину, она фактически предполагает строгий порядок обработки, невыгодный для параллельных систем. Суть формы *kij* состоит в том, что строки (а не столбцы) модифицируются, как только это становится возможным, и представляется естественным выполнять параллельно как можно большее число этих модификаций. Так, в программе на рис. 2.1.1 k-й шаг должен быть закончен, прежде чем начнется $(k+1)$-й шаг. Однако это требование в случае параллельной системы является обременительным. Действительно, в то время как одни процессоры еще завершают k-й шаг, другие могут уже быть свободными; для достижения максимальной эффективности таким процессорам следует позволить начать операции $(k+1)$-го шага. Итак, процессорам будет разрешено переходить к следующему шагу сразу после того, как закончены вычисления данного шага. Теперь мы более подробно рассмотрим конкретные формы.

Форма LU-kij, r

Обозначим через *LU-kij, r* форму *kij* LU-разложения при использовании строчной слоистой схемы; аналогичным образом будут обозначаться другие формы. На рис. П.1.7 приведен псевдокод, описывающий работу отдельного процессора при прямолинейной реализации алгоритма *LU-kij, r*; имя mygows относится к множеству индексов строк, приписанных к данному процессору. В случае использования строчной слоистой схемы каждый процессор вычисляет множитель, а затем модифицирует хранящуюся в нем строку. Все процессоры могут выполнять эти действия одновременно, только при обработке нижней подматрицы $p \times p$ процессоры один за другим начнут прекращать работу. Чтобы выполнить модификации k-го шага, каждому процессору требуется строка k; следовательно, строка k должна быть разослана всем процессорам до начала k-го шага.

В алгоритме на рис. П.1.7 синхронизация процессоров на каждом шаге происходит автоматически, поскольку процессор $P(k+1)$ закончит шаг k не раньше других процессоров и должен еще распространить информацию. Такая организация нежелательна, потому что заставляет процессоры простаивать

```
Для k = 1 до n − 1 выполнить
    если k принадлежит множеству myrows то
        распространить строку k
    иначе
        принять строку k
    для всех строк i > k из myrows
        вычислить l_ik
        модифицировать строку i
```

Рис. П.1.7. Простейшая *LU-kij, r* программа для отдельного процессора

в ожидании очередной ведущей строки. Поэтому мы позволим процессору $P(k+1)$ распространить строку $k+1$, как только ее модификация завершена, и лишь затем приступить к моди-

```
распространить строку 1
    для k = 1 до n − 1
        если k не принадлежит множеству myrows то
            принять строку k
        для всех строк i > k из myrows
            l_ik = a_ik/a_kk
        для j = k + 1 до n
            a_ij = a_ij − l_ik a_kj
        если i = k + 1 и i ≠ n то
            распространить строку k + 1
```

Рис. П.1.8. *LU-kij, r* программа для отдельного процессора

фикации прочих его строк. Конфликты при обменах для подобной стратегии опережающей рассылки маловероятны, так как строка $k+1$ не будет готова для распространения, прежде чем будет получена и использована строка k. Псевдокод этого алгоритма, который мы будем рассматривать как основную реализацию формы *LU-kij, r*, приведен на рис. П.1.8.

Алгоритм на рис. П.1.8 можно улучшить, если начинать вычисление множителей еще тогда, когда продолжается прием ведущей строки. Например, вычислять множители l_{i1} можно уже при получении элемента a_{11} — вместо того чтобы ожидать приема полной первой строки. В любом случае задержки обменов представляются вполне приемлемыми.

Для параллельно-векторных машин модификации производятся посредством векторных операций над строками. Для этого нужно, разумеется, чтобы все элементы требуемых строк были доступны. На k-м шаге векторы имеют полную длину $n-k$; в отличие от этого, для некоторых других форм длины векторов равны $O((n-k)/p)$.

Форма LU-kij, c

Рассмотрим теперь форму *kij* со столбцовым слоистым хранением, в наших обозначениях форму *LU-kij, c*. На k-м шаге в $P(k)$ будут вычислены все множители. Согласно алгоритму на рис. 2.1.1, каждый множитель используется для модификации соответствующей строки, прежде чем вычисляется следующий множитель. Однако ясно, что если процессор $P(k)$ не даст приоритета вычислению полного k-го столбца множителей перед модификацией остальных хранящихся в нем столбцов, то это вызовет задержку в работе других процессоров. Множители могут вычисляться и распространяться сегментами; сегмент длины 1 соответствует распространению каждого множителя сразу после его вычисления. Хотя этот прием позволяет другим процессорам приступить к модификациям насколько возможно рано, стоимость обменов будет гораздо выше, чем для алгоритма *LU-kij, r*.

С другой стороны, сегментом может быть и полный столбец множителей. Это повлечет за собой задержки на каждом шаге на время вычисления множителей, но ситуацию можно поправить (за исключением первого шага) дополнительной модификацией алгоритма *LU-kij, c*, а именно включением в него стратегии опережающего вычисления и рассылки. Конкретнее, получив k-й столбец множителей, процессор $P(k+1)$ должен отдать приоритет вычислению и рассылке $(k+1)$-го столбца множителей. Это требует дальнейшего отступления от ортодоксальной схемы *kij*, поскольку столбец $k+1$ модифицируется полностью, прежде чем процессор $P(k+1)$ займется модификацией строк оставшейся части своей подматрицы. Подобный вариант алгоритма был бы гораздо более естественным в контексте формы *kji*.

В случае использования векторных процессоров вычисление всего столбца множителей можно выполнить с помощью векторной команды деления. Однако, поскольку каждая строка в форме *kij*, *c* распределена между *p* процессорами, длины векторов при строчных операциях будут в *p* раз меньше, чем в форме *kij*, *r*. Такие длины векторов мы назовем *частичными*. Как указано выше, форма *kij*, *r* допускает векторы полной длины во всех операциях пересчета.

Форма LU-kji, r

Форма *kji* отличается от формы *kij* тем, что модификации производятся по столбцам, а не по строкам. Мы снова начинаем со строчной слоистой схемы хранения. Псевдокод на рис. 2.1.2 можно адаптировать для параллельной системы следующим образом. На *k*-м шаге полный столбец множителей вычисляется параллельно всеми процессорами. Затем оставшаяся подматрица в *A* модифицируется столбец за столбцом. Вычисление множителей и модификации требуют, чтобы строка *k* была доступна всем процессорам, поэтому до начала *k*-го шага процессор $P(k)$ должен распространить строку *k*. Стоимость обменов такая же, как в простейшей версии алгоритма *LU-kij*, *r*; в действительности одинаковой будет и общая стоимость, поскольку на каждом шаге процессоры делают в обоих случаях одни и те же операции с одними и теми же элементами; различается только порядок операций.

Чтобы устранить ненужные задержки, связанные с ожиданием данных, мы снова применим стратегию опережающего вычисления и рассылки. В начале *k*-го шага процессор $P(k+1)$ пересчитает и распространит строку $k+1$, прежде чем модифицировать остальные части своих столбцов. У этого алгоритма нет никаких преимуществ перед алгоритмом *LU-kij*, *r*: время работы такое же, а программа несколько сложнее, поскольку элементы строки $k+1$ модифицируются в ином порядке, чем остальная часть матрицы коэффициентов. Для векторных процессоров форма *kji*, *r* имеет тот недостаток, что используются лишь частичные длины векторов

Форма LU-kji, c

Алгоритм *LU-kji*, *c* в своей простейшей форме страдает тем же недостатком, что и простейшая версия алгоритма *LU-kij*, *r*, т. е. нежелательной синхронизацией в конце каждого шага. Однако можно применить стратегию, аналогичную той, что была применена для формы *LU-kij*, *r*: на *k*-м шаге, сразу после модификации столбца $k+1$, процессор $P(k+1)$ вычисляет и

распространяет $(k+1)$-й столбец множителей и лишь затем продолжает модификацию остальных своих столбцов. На каждом шаге, кроме первого, это сократит время задержки для процессоров, ожидающих получения очередного столбца множителей. Поскольку используются полные длины векторов, как скалярная, так и векторная версии алгоритма *kji, c* кажутся привлекательными (несмотря на задержку в самом начале алгоритма).

Формы *ikj* и *jki* *LU*-разложения: отложенные модификации

Псевдокоды для форм *ikj* и *jki* приведены на рис. П.1.3. Сущность формы *ikj* заключается в том, что модификация каждой строки откладывается до того момента, когда она должна

Для $i=2$ до n вычислить i-ю строку матрицы L выполнить все модификации для i-й строки матрицы A	Для $j=2$ до n вычислить $(j-1)$-й столбец матрицы L выполнить все модификации для j-го столбца матрицы A

Рис. П.1.9. Параллельные формы *ikj* и *jki* *LU*-разложения

получить завершающую обработку. То же самое верно для формы *jki*, если говорить о столбцах вместо строк. Это отражено на рис. П.1.9. Теперь мы перейдем к более подробному обсуждению алгоритмов.

Форма LU-ikj, r

Если строго придерживаться стратегии отложенных модификаций, то, поскольку используется строчная схема хранения, на каждом шаге будет работать только один процессор: сначала P_2 пересчитает вторую строку и разошлет ее всем процессорам, затем P_3 пересчитает третью строку, и т. д. Чтобы не простаивать, процессоры могут начать обработку прочих строк. После того как в P_2 завершена модификация строки 2, она рассылается всем процессорам, и для P_3 приоритет приобретает вычисление третьей строки и ее рассылка. Таким образом, каждый процессор отдает приоритет стратегии отложенных модификаций, но во время ожидания данных он пересчитывает другие

строки. В случае использования векторных процессоров длины строк будут полными.

Возможна и такая вариация на данную тему: $P(1)$ распространяет строку 1, после чего остальные процессоры начинают модификацию своих первых строк, в то время как $P(1)$ пересчитывает строку $p+1$. Затем $P(2)$ распространяет новую строку 2, и другие процессоры пересчитывают с ее помощью следующие $p-1$ строк; тем временем $P(2)$ модифицирует строку $p+2$ с помощью строки 1, и т. д. В результате в каждый момент модифицируется лента из p строк, и по мере того как строки в верхней части ленты принимают завершенный вид, внизу добавляются новые строки, так что лента как бы движется сквозь матрицу. Хотя эта стратегия позволяет процессорам работать параллельно, она имеет серьезный изъян. Когда в нижней части ленты появляются новые строки, их нужно модифицировать с помощью всех предыдущих строк. Следовательно, нужно либо хранить в каждом процессоре строки с меньшими номерами, либо распространять их снова каждый раз, когда они нужны. В любом случае требуются большие дополнительные затраты на хранение или обмены.

Ясно, что форма ikj, r окажется не очень полезной, если строго соблюдать стратегию отложенных модификаций. С указанными выше изменениями она становится более похожа на форму kij, r (за исключением более сложного программирования). Для некоторых векторных процессоров отложенные модификации имеют то достоинство, что уменьшают число записей в память; для других к этому добавляется уменьшение количества индексных вычислений и считываний из памяти. Но вряд ли это окупает усложнение алгоритма

Форма LU-ikj, c

В этой форме отложенные модификации выполняются по строкам, а матрица хранится по столбцам. На i-м шаге процессор 1 вычисляет множитель l_{i1} и рассылает его всем процессорам. Каждый процессор пересчитывает с помощью строки 1 свою порцию строки i; затем процессор 2 вычисляет и рассылает множитель l_{i2}, после чего все процессоры пересчитывают с помощью строки 2 свою порцию строки i, и т. д. Так продолжается до тех пор, пока i-я строка не будет пересчитана полностью. Каждая модификация строки i выполняется всеми процессорами параллельно, однако распространение множителей вызывает задержки в алгоритме.

Другой вариант организации процесса предусматривает вычисление и рассылку множителей до пересчета остальной части

строки. Таким образом, вычисление и пересылки множителей совмещаются с пересчетом строк, что представляет собой некоторое отступление от стратегии отложенных модификаций. Понятно, что запрограммировать этот вариант сложнее, а длины векторов (в случае векторных процессоров) будут лишь частичными. Мы приходим к заключению, что в данной форме алгоритма привлекательного мало.

Форма LU-jki, r

В форме *jki* отложенные модификации выполняются по столбцам. Для параллельной системы и строчной схемы хранения алгоритм работает следующим образом. На j-м шаге параллельно вычисляется весь $(j-1)$-й столбец множителей (для этого все процессоры должны иметь доступ к диагональному элементу). Затем для k, растущего от 1 до $j-1$, процессор $P(k)$ распространяет элемент a_{kj}; эти элементы используются всеми процессорами как коэффициенты при параллельном пересчете jго столбца. Как только процессор $P(k+1)$ пересчитал элемент $a_{k+1,j}$, он распространяет его новое значение, прежде чем начать пересчет любых других элементов столбца j, которые в нем хранятся. Хотя в этой форме налицо высокая степень параллелизма, стоимость обменов высока, поскольку каждый элемент из U распространяется в качестве коэффициента некоторой линейной комбинации столбцов. Кроме того, в случае векторных процессоров недостатком являются лишь частичные длины векторов.

Форма LU-jki, c

Поскольку используется столбцовая схема хранения, строгое соблюдение предписаний стратегии отложенных модификаций сделало бы эту форму бесполезной: при пересчете некоторого столбца все процессоры, кроме одного, простаивали бы. Единственный способ придать этой форме практический смысл — крутой отход от философии отложенных модификаций. Например, на первом шаге в P_1 вычисляются множители и рассылаются остальным процессорам. Все процессоры выполняют пересчет своих столбцов, пока P_2 не закончит модификацию второго столбца, не вычислит соответствующие множители и не разошлет их. После этого P_2 может заняться пересчетом остальных своих столбцов, а P_3 завершает модификацию третьего столбца, и т. д. Таким образом, приоритет отдается отложенным модификациям столбцов, приобретающих окончательный вид, но одновременно другие процессоры выполняют

текущие модификации. Ситуация аналогична той, что складывалась для формы ikj, r; как и для этой последней, мы можем сконструировать версию алгоритма с *движущейся лентой*. Длины векторов в данном случае полные. Как и в случае формы ikj, r, мы заключаем, что данный алгоритм может представлять интерес только для некоторых типов векторных процессоров.

Формы *ijk* и *jik*: алгоритмы скалярных произведений

Псевдокоды для форм *ijk* и *jik* приведены на рис. П.1.5. Интерпретация этих форм дана на рис. П.1.10.

Для $i = 2$ до n вычислить i-е строки матриц L и U с помощью скалярных произведений i-й строки L и столбцов U	Для $j = 2$ до n вычислить $(j-1)$-й столбец в L и j-й столбец в U с помощью скалярных произведений i-й строки L и столбцов U

Рис. П.1.10. Параллельные формы *ijk* и *jik* LU-разложения

Форма LU-ijk, r

Поскольку матрица A хранится по строкам, для скалярного произведения строки из L и столбца из U потребуются данные из одного процессора для строки и из ряда процессоров для столбца. Понадобятся пересылки необходимых элементов строки L соответствующим процессорам, после чего в каждом процессоре будут вычислены частичные скалярные произведения, а скалярное произведение в целом будет получено путем межпроцессорного сложения по методу сдваивания. Количество обменов в данном алгоритме столь велико, что его употребление не имеет особого смысла. (Кроме того, вначале очень велик дисбаланс загрузки процессоров; так, при вычислении второй строки матрицы U используется только первый процессор. Однако к концу алгоритма загруженность процессоров становится равномерной.)

Форма LU-ijk, c

Здесь в скалярном произведении участвуют столбец U, хранимый каким-то одним процессором, и строка L, распределенная между несколькими процессорами. Снова потребуются пере-

сылки необходимых сегментов столбца другим процессором, и дальше все будет идти так же, как в форме ijk, r. Таким образом, данная форма имеет те же недостатки, что и предыдущая.

Формы LU-jik, r и LU-jik, c

У этих двух форм те же недостатки, что и у двух форм *jik*.

Напомним, что свойства различных форм LU-разложения в случае использования параллельных компьютеров сведены в таблицу 2.2.1.

Разложение Холесского

Теперь мы рассмотрим разложение Холесского $A = LL^{\mathrm{T}}$. Через CH-kij, r и CH-kij, c будем обозначать форму *kij* метода Холесского со слоистой схемой хранения, соответственно срочной или столбцовой; аналогичные обозначения используются для других форм. Обсудим более подробно различные формы метода. Псевдокоды для форм *kij* и *kji* (алгоритмов безотлагательных модификаций) приведены на рис. П.1.2.

Форма CH-kij, r

В этой форме используется слоистая строчная схема хранения. На k-м шаге процессор $P(k)$ вычисляет и рассылает элемент l_{kk}. Для каждого $i (k < i \leqslant n)$ процессор $P(i)$ вычисляет l_{ik} и модифицирует i-ю строку матрицы A. Для этой модификации нужны значения элементов $l_{k+1,\,k}, \ldots, l_{ik}$, следовательно, каждому процессору требуется доступ к большей части k-го столбца матрицы L. Таким образом, объем обменов в алгоритме весьма значителен. Однако по крайней мере часть этих обменов можно совместить с вычислениями, если организовать следующим образом процесс. Когда процессоры получают значение l_{kk}, они параллельно вычисляют свои элементы k-го столбца матрицы L и затем распространяют их. Ожидая получения этих данных, они могут начать модификации своих строк, пользуясь теми элементами k-го столбца, которые хранятся в них самих. Заметим, что все процессоры попытаются начать распространение приблизительно в одно и то же время, что для некоторых типов архитектуры повлечет к серьезным конфликтам в каналах связи.

После первого шага дальнейшее усовершенствование можно получить за счет следующей стратегии опережающего вычисления и рассылки. Все процессоры вначале вычисляют и рассылают элементы следующего столбца матрицы L и лишь затем завершают текущие модификации. Таким образом, на k-м

шаге процессор $P(k+1)$ вычисляет и сразу распространяет значение $l_{k+1,\,k+1}$, после чего все процессоры вычисляют и распространяют свои элементы $(k+1)$-го столбца матрицы L, а потом заканчивают модификации k-го шага. Эта стратегия представляет собой довольно сильный отход от ортодоксальной формы *kij*; она более естественно выглядит в контексте формы *kji*, *r*, рассматриваемой ниже. В случае применения векторных процессоров в данной форме могут использоваться векторы полной длины. Однако если нельзя осуществить стратегию опережающего вычисления и совмещение вычислений с обменами, то форма *kij*, *r* уступает некоторым другим формам.

Форма CH-kij, c

Пусть теперь матрица хранится в соответствии со слоистой столбцовой схемой. На k-м шаге процессор $P(k)$ вычисляет и распространяет элементы l_{ik} k-го столбца матрицы L. Это можно сделать разными способами. Если числа l_{ik} распространяются по одному, то остальные процессоры могут начать модификации своих частей строк, как только получено необходимое значение l_{ik}; по аналогии с формой *kij*, *r*, при модификации i-й строки нужны элементы $l_{k+1,\,k}, \ldots, l_{ik}$. При такой организации процесса потребуется $O\left(\frac{1}{2}n^2\right)$ рассылок, т. е. чрезмерно много. С другой стороны, можно несколько отступить от ортодоксальной формы *kij*, вычисляя и распространяя одновременно несколько элементов l_{ik}. Это уменьшит число рассылок, но, возможно, за счет увеличения задержек при ожидании данных процессорами. В предельном случае можно распространять сразу весь k-й столбец матрицы L, и не исключено, что это наилучшая стратегия, если сочетать ее, как и для формы *kij*, *r*, с опережающей рассылкой: на k-м шаге процессор $P(k+1)$ должен вычислить и распространить полный $(k+1)$-й столбец матрицы L, и только затем завершить свои модификации k-го шага. Это опять-таки довольно сильное отклонение от схемы *kij*. Тот же эффект можно получить для формы *kji*, обсуждаемой в следующем пункте. В случае использования векторных процессоров на первом шаге возникнут дополнительные задержки при ожидании достаточного количества множителей, но на последующих шагах положение, возможно, удастся исправить за счет стратегии опережающего вычисления. Однако длины векторов будут лишь частичными.

Форма CH-kji, r

В форме *kji* модификации также выполняются сразу, но на этот раз по столбцам. Последовательность действий для формы

kji, r такова. На первом шаге $P(1)$ вычисляет и распространяет элемент l_{11}, после чего все процессоры могут вычислять свою часть первого столбца матрицы L. Далее возможны две стратегии обменов. Поскольку модификация j-го столбца требует, чтобы каждый процессор располагал элементом l_{j1} в дополнение к хранимым в нем элементам j-го столбца, то можно распространять элементы первого столбца один за другим. Так, $P(2)$ распространяет l_{21}, и тогда все процессоры могут модифицировать свои части второго столбца, в то время как $P(3)$ распространяет l_{31}, и т. д. Общее число рассылок для этой стратегии равно $O\left(\frac{1}{2}n^2\right)$

Вторая стратегия состоит в том, что все процессоры одновременно распространяют свои части k-го столбца; это потребует в общей сложности $O(pn)$ рассылок. Как и в случае формы *kij, r*, можно применить опережающее вычисление, т. е. вычислить и распространить элементы $(k+1)$-го столбца матрицы L до завершения модификаций k-го шага. Если можно совместить обмены и вычисления, то такая организация процесса будет вполне удовлетворительной. Отметим, что она является отступлением от строгого метода *kji*, но меньшим, чем это было для формы *kij*, поскольку модификации производятся по столбцам. Длины векторов для формы *kji, r* будут лишь частичными.

Форма CH-kji, c

Обсуждение формы *kji, c* (матрица хранится по столбцам) почти полностью повторяет обсуждение формы *kij, c* с тем исключением, что теперь модификация $(k+1)$-го столбца не является таким уж сильным отклонением от ортодоксального метода *kji*; в самом деле, это всего лишь изменение последовательности обменов, а не вычислений. Как и в предыдущих случаях, крайне желательно воспользоваться опережающей рассылкой и на k-м шаге как можно быстрее пересчитать и распространить $(k+1)$-й столбец матрицы L. При параллельно-векторной реализации длины векторов будут полными. Понятно, что среди всех вариантов форм *kij* и *kji* форма *kji, c* является наилучшей

Формы *ikj* и *jki*: отложенные модификации

Псевдокоды для форм *ikj* и *jki* приведены на рис. П.1.4. В первой из них откладываются модификации строк, во второй — модификации столбцов. Обсудим эти формы более подробно.

Форма CII-ikj, r

Здесь, как и в случае LU-разложения, для обеспечения приемлемого баланса загрузки процессоров придется довольно сильно отклоняться от стратегии отложенных модификаций. Технику «движущейся ленты», обсуждавшуюся для LU-разложения, можно применить и для метода Холесского, но опять-таки мы в лучшем случае получим алгоритм, не превосходящий алгоритма kij, r. Единственным исключением является случай векторных процессоров, для которого прием отложенных модификаций важен.

Форма CII-ikj, c

К хранению матрицы по столбцам форма ikj приспособлена несколько лучше, поскольку в модификацию строки i вовлечены все процессоры. В частности, произведения $l_{ik}l_{jk}$ могут быть вычислены параллельно. Однако эти произведения должны вычитаться из элемента a_{ij}, находящегося в процессоре $P(j)$. Ясно, что потребуется большое количество обменов. Таким образом, данная форма не очень привлекательна.

Форма CH-jki, r

Эта форма предусматривает отложенные модификации столбцов при строчной схеме хранения. Ее свойства схожи со свойствами формы kji, r с тем исключением, что все модификации j-го столбца производятся на j-м шаге; для k-й из этих модификаций требуется k-й столбец матрицы L (от элемента l_{jk} вниз) и элемент l_{jk} как коэффициент. Поэтому в начале шага элементы j-й строки L можно либо полностью разослать до выполнения модификаций, либо рассылать по одному; тогда каждая модификация начинается при получении очередного множителя. В этом последнем варианте потребуется больше времени на обмены. Длины векторов будут лишь частичными.

Форма CH-jki, c

Хранение осуществляется по столбцам, но проблемы здесь в сущности те же, что и для формы ikj, r. Чтобы получить достаточно равномерную загруженность процессоров, приходится сильно отступать от стратегии отложенных модификаций.

Формы *ijk* и *jik*: алгоритмы скалярных произведений

Псевдокоды для этих форм приведены на рис. П.1.6. В форме *ijk* производятся отложенные модификации строк, а внутренний цикл представляет собой скалярное произведение двух строк матрицы L.

Форма CH-ijk, r

На i-м шаге этого алгоритма в процессоре $P(i)$ из элементов i-й строки вычитаются скалярные произведения строк $1, \ldots, i-1$ с текущей i-й строкой. Поскольку i-я строка постоянно пересчитывается, эти скалярные произведения нельзя вычислить параллельно, следовательно, при выполнении модификаций в каждый момент будут действовать самое большее два процессора. Если допустить отступление от ортодоксальной схемы, чтобы другие процессоры могли производить операции более поздних шагов, то для этого потребуется или каждый раз заново посылать любую строку, если она нужна для пересчета строки с бо́льшим номером, или же хранить все строки в каждом процессоре. Это сильно увеличит затраты на обмены или количество потребляемой памяти.

Форма CH-ijk, c

При использовании слоистой столбцовой схемы хранения ситуация становится лишь немногим лучше. Теперь строки распределены между процессорами и частичные скалярные произведения могут вычисляться параллельно. Однако затем эти частичные скалярные произведения нужно суммировать с помощью процедуры сдваивания, и стоимость обменов становится недопустимо высокой. Мы заключаем, что ни форма *ijk, r*, ни форма *ijk, c* не могут конкурировать с другими формами.

Формы CH-jik, r и CH-jik, c

В форме *jik* также вычисляются скалярные произведения строк матрицы L, но теперь L строится по столбцам, а не по строкам. На j-м шаге вычисляется j-столбец L и требуются скалярные произведения j-й строки L со всеми последующими строками. Начнем со строчной слоистой схемы хранения. Прежде всего нужно разослать всем процессорам j-ю строку L, тогда скалярные произведения можно будет вычислять параллельно. Таким образом, используется та же числовая информация, что и в случае формы *jki, r*. Различие в том, что в алгоритме *jki* пересчет j-го столбца осуществляется с помощью линейной

комбинации других столбцов, тогда как в алгоритме *jik* для этой цели применяются скалярные произведения строк. Для последовательных машин оба алгоритма эквивалентны. Однако в случае использования векторных процессоров в форме *jik, r* вычисляются скалярные произведения векторов полной длины, а в форме *jki, r* приходится оперировать только с векторами частичной длины. Аналогичным образом нетрудно убедиться, что форма *jik, c* (слоистая столбцовая схема хранения) страдает теми же недостатками, что и форма *jki, c*.

Напомним, что свойства параллельных форм метода Холесского сведены в таблицу 2.2.2.

Приложение 2

Сходимость итерационных методов

В этом приложении мы обсудим некоторые основные теоремы о сходимости итерационных методов. Большинство их этих результатов являются классическими и приводятся в монографиях [Varga, 1962] и [Young, 1971]. Рассмотрим сначала итерационный метод

$$\mathbf{x}^{k+1} = H\mathbf{x}^k + \mathbf{d}, \quad k = 0, 1, \ldots, \tag{П.2.1}$$

отыскания приближенного решения линейной системы

$$A\mathbf{x} = \mathbf{b}. \tag{П.2.2}$$

Будем предполагать, что матрица A размера $n \times n$ невырождена и что $\hat{\mathbf{x}}$ — единственное решение системы (П.2.2). Тогда итерационный метод (П.2.1) называют *согласованным* с системой (П.2.2), если

$$\hat{\mathbf{x}} = H\hat{\mathbf{x}} + \mathbf{d}. \tag{П.2.3}$$

Для согласованного метода мы можем, вычитая (П.2.3) из (П.2.1), получить основное уравнение для погрешности

$$\mathbf{e}^{k+1} = H\mathbf{e}^k, \quad k = 0, 1, \ldots, \tag{П.2.4}$$

где $\mathbf{e}^i = \mathbf{x}^i - \hat{\mathbf{x}}$ — погрешность на i-м шаге. Это уравнение эквивалентно следующему:

$$\mathbf{e}^k = H^k\mathbf{e}^0, \quad k = 1, 2, \ldots. \tag{П.2.5}$$

Если мы потребуем, чтобы метод (П.2.1) сходился для любого начального приближения $\mathbf{x}^0$, то эквивалентным этому будет требование, чтобы $\mathbf{e}^k \to 0$ при $k \to \infty$ для любого $\mathbf{e}^0$. Последовательно выбирая $\mathbf{e}^0$ равным координатным векторам $\mathbf{e}_1$, $\mathbf{e}_2$, ... (где $\mathbf{e}_i$ — вектор с компонентами, равными нулю, за исключением i-й компоненты, равной единице), мы видим, что условие $\mathbf{e}^k \to 0$ при $k \to \infty$ для любого $\mathbf{e}^0$ эквивалентно условию $H^k \to 0$ при $k \to \infty$. Если $\lambda_1, \ldots, \lambda_n$ — собственные значения матрицы

H и $\rho(H) = \max_i |\lambda_i|$ — спектральный радиус матрицы H, то имеет место следующий основной результат, доказательство которого дается, например, в монографии Ортеги [Ortega, 1987a, теорема 5.3.4].

П.2.1. Теорема. *Если H — матрица $n \times n$, то $H^k \to 0$ при $k \to \infty$ в том и только в том случае, если $\rho(H) < 1$. Таким образом, согласованный итерационный метод* (П.2.1) *сходится для любого $\mathbf{x}^0$ к единственному решению системы* (П.2.2) *в том и только в том случае, если $\rho(H) < 1$.*

Согласованные итерационные методы возникают естественным образом при следующем подходе. Пусть

$$A = P - Q \qquad (П.2.6)$$

— расщепление матрицы A, и предположим, что матрица P невырожденна. Легко убедиться в том, что процесс

$$\mathbf{x}^{k+1} = P^{-1}Q\mathbf{x}^k + P^{-1}\mathbf{b}, \quad k = 0, 1, \ldots, \qquad (П.2.7)$$

является согласованным итерационным методом. Для таких методов вопрос о сходимости сводится к установлению того, что $\rho(P^{-1}Q) < 1$. Полезным инструментом для этого может служить следующий результат, доказательство которого дается, например, в [Ortega, 1987a, теорема 5.4.2].

П.2.2. Теорема Стейна. *Если H — вещественная матрица $n \times n$, то $\rho(H) < 1$ в том и только в том случае, если существует такая симметричная положительно определенная матрица B, что матрица $B - H^tBH$ положительно определена.*

Будем говорить, что вещественная, не обязательно симметричная матрица C является *положительно определенной*, если $\mathbf{x}^tC\mathbf{x} > 0$ для всех вещественных векторов $\mathbf{x} \neq 0$. Это эквивалентно требованию, чтобы симметричная часть матрицы C, равная $(C + C^t)/2$, была положительно определенной в обычном смысле. Мы используем введенное понятие следующим образом.

П.2.3. Определение. Расщепление $A = P - Q$ называется *P-регулярным*, если матрица P невырожденна, а матрица $P + Q$ положительно определена.

Теперь мы можем сформулировать следующую теорему сходимости.

П.2.4. Теорема о *P*-регулярном расщеплении. *Если A — симметричная положительно определенная матрица и $A = P - Q$ — ее P-регулярное расщепление, то $\rho(P^{-1}Q) < 1$.*

Доказательство. Пусть $H = P^{-1}Q$ и $C = A - H^{\mathrm{T}}AH$. Тогда в силу $P^{-1}Q = I - P^{-1}A$ имеем

$$\begin{aligned} C &= A - (I - P^{-1}A)^{\mathrm{T}} A (I - P^{-1}A) = \\ &= (P^{-1}A)^{\mathrm{T}} A + AP^{-1}A - (P^{-1}A)^{\mathrm{T}} AP^{-1}A = \\ &= (P^{-1}A)^{\mathrm{T}} (P + P^{\mathrm{T}} - A) P^{-1}A = \\ &= (P^{-1}A)^{\mathrm{T}} (P^{\mathrm{T}} + Q)(P^{-1}A). \end{aligned} \qquad (\text{П.2.8})$$

По предположению матрица $P + Q$ положительно определена, и поэтому матрица $P^{\mathrm{T}} + Q$ также положительно определена. Таким образом, поскольку матрица C конгруэнтна матрице $P^{\mathrm{T}} + Q$, то она также положительно определена. Тогда по теореме Стейна П.2.2 получаем, что если A положительно определена, то $\rho(H) < 1$. □

Существует два различных обращения утверждения П.2.4, каждое из которых может оказаться полезным в некоторых случаях. Первое из них формулируется следующим образом:

П.2.5. Первое обращение теоремы о *P*-регулярном расщеплении. *Пусть матрица A симметрична и невырожденна, $A = P - Q$ является P-регулярным расщеплением и $\rho(P^{-1}Q) < 1$. Тогда матрица A положительно определена.*

Доказательство. Снова положим $H = P^{-1}Q$, так что $\rho(H) < 1$. Тогда для любого $\mathbf{x}^0$, согласно П.2.1, последовательность $\mathbf{x}^k = H\mathbf{x}^{k-1}$, $k = 1, 2, \ldots$, сходится к нулю. Предположим теперь, что матрица A не является положительно определенной. Тогда существует вектор $\mathbf{x}^0 \neq 0$, такой, что $(\mathbf{x}^0)^{\mathrm{T}} A\mathbf{x}^0 \leqslant 0$

и

$$\mathbf{y}^0 = P^{-1}A\mathbf{x}^0 \neq 0.$$

Таким образом, из (П.2.8) получаем

$$(x^0)^{\mathrm{T}} C\mathbf{x}^0 = (\mathbf{y}^0)^{\mathrm{T}} (P^{\mathrm{T}} + Q)\mathbf{y}^0 > 0$$

в силу предположения о P-регулярности расщепления. Кроме того, из (П.2.8) видно, что матрица C положительно определена, так что для любого $k \geqslant 0$

$$0 \leqslant (\mathbf{x}^k)^{\mathrm{T}} C\mathbf{x}^k = (\mathbf{x}^k)^{\mathrm{T}} A\mathbf{x}^k - (\mathbf{x}^{k+1})^{\mathrm{T}} A\mathbf{x}^{k+1},$$

причем для $k = 0$ мы уже показали, что неравенство будет строгим. Поэтому

$$(\mathbf{x}^{k+1})^{\mathrm{T}} A\mathbf{x}^{k+1} \leqslant (\mathbf{x}^k)^{\mathrm{T}} A\mathbf{x}^k \leqslant \ldots \leqslant (\mathbf{x}^1)^{\mathrm{T}} A\mathbf{x}^1 < (\mathbf{x}^0)^{\mathrm{T}} A\mathbf{x}^0 \leqslant 0.$$

Однако полученные соотношения противоречат тому факту, что $\mathbf{x}^k \to 0$ при $k \to \infty$, откуда следует, что матрица A должна быть положительно определенной. □

Второе обращение теоремы П.2.4 применяется в том случае, когда сама матрица P также симметрична и положительно определена. В этом случае из сходимости следует не только то, что A положительно определена, но и то, что расщепление является P-регулярным. Заметим, что утверждения П.2.4 и П.2.6 включают в себя результат 3.4.3, приводившийся в основном тексте.

П.2.6. Второе обращение теоремы о *P*-регулярном расщеплении. *Предположим, что матрица $A = P - Q$ симметрична и невырожденна, матрица P симметрична и положительно определена и $\rho(P^{-1}Q) < 1$. Тогда матрицы A и $P + Q$ положительно определены.*

Для доказательства теоремы П.2.6 нам потребуется следующее утверждение о собственных значениях произведений симметричных матриц (его доказательство можно найти, например, в [Ortega, 1987a, теорема 6.2.3]).

П.2.7. Теорема. *Пусть B и C — вещественные симметричные матрицы. Если одна из них положительно определена, то их произведение BC имеет вещественные собственные значения. Если обе матрицы B и C положительно определены, то BC имеет положительные собственные значения. Обратно, если матрица BC имеет положительные собственные значения и одна из матриц (B или C) положительно определена, то обе матрицы B и C положительно определены.*

Доказательство теоремы П.2.6. Положим, как и ранее, $H = P^{-1}Q = I - P^{-1}A$. Так как матрица P^{-1} положительно определена, из теоремы П.2.7 следует, что собственные значения $\lambda_1, \ldots, \lambda_n$ матрицы H вещественны, и в силу $\rho(H) < 1$ должны выполняться неравенства

$$-1 < \lambda_i < 1, \quad i = 1, \ldots, n. \qquad (\text{П.2.9})$$

Таким образом, собственные значения матрицы $P^{-1}A$ положительны и в силу утверждения П.2.7 матрица A положительно определена. Кроме того, существует матрица $(I - H)^{-1}$, и матрица $G = (I - H)^{-1}(I + H)$ имеет собственные значения $(1 + \lambda_i)/(1 - \lambda_i)$, $i = 1, \ldots, n$, которые положительны в силу

(П.2.9). Можно представить матрицу G в виде

$$G = (I - P^{-1}Q)^{-1}(I + P^{-1}Q) =$$
$$= (P^{-1}(P - Q))^{-1} P^{-1}(P + Q) = A^{-1}(P + Q),$$

и тогда из П.2.7 следует, что $P + Q$ положительно определена, поскольку положительно определена A^{-1}. □

Заметим, что остается открытым вопрос о справедливости П.2.6 при отсутствии предположения о симметричности матрицы P.

Применим теперь приведенные общие утверждения к исследованию некоторых конкретных итерационных методов. Заметим сначала, что теорема 3.1.2 является непосредственным следствием утверждений П.2.4 и П.2.6, если положить $P = D$ и $Q = B$.

П.2.8. Сходимость итераций метода Якоби. *Если матрица $A = D - B$ симметрична и невырожденна, а диагональная часть D матрицы A положительно определена, то итерации метода Якоби сходятся для любого начального приближения $\mathbf{x}^0$ тогда и только тогда, когда положительно определена каждая из матриц A и $D + B$.*

Докажем теперь теорему 3.2.3 в несколько усиленной форме.

П.2.9. Теорема Островского — Райха. *Если матрица A симметрична, невырожденна и имеет положительные диагональные элементы, то при $\omega \in (0, 2)$ итерации метода SOR сходятся при любом $\mathbf{x}^0$ тогда и только тогда, когда матрица A положительно определена.*

Доказательство. Применим сначала теорему П.2.4. Если $A = D - L - L^T$, то матрица перехода метода SOR имеет вид

$$H_\omega = (D - \omega L)^{-1}((1 - \omega) D + \omega L^T) \qquad (П.2.10)$$

и легко показать, что P и Q задаются соотношениями

$$P = \omega^{-1}(D - \omega L), \quad Q = \omega^{-1}((1 - \omega) D + \omega L^T).$$

Так как диагональные элементы матрицы D положительны, матрица P невырожденна и остается только убедиться в том, что матрица $P + Q$ положительно определена. Действительно, симметричная часть матрицы $P + Q$ равна

$$\frac{1}{2}(P + P^T) + \frac{1}{2}(Q + Q^T) = \frac{1}{2\omega}(2D - \omega L - \omega L^T) +$$
$$+ \frac{1}{2\omega}(2(1 - \omega) D + \omega L + \omega L^T) = \frac{2 - \omega}{\omega} D$$

и, следовательно, положительно определена при $\omega \in (0, 2)$. Обратное утверждение следует непосредственно из теоремы П.2.5. □

Теорема П.2.9 описывает весь диапазон допустимых вещественных значений параметра ω итерационного метода SOR. Это вытекает из следующего результата, справедливого также и для несимметричных матриц.

П.2.10. Лемма Кахана. *Пусть матрица A имеет ненулевые диагональные элементы. Тогда спектральный радиус матрицы перехода метода SOR* (П.2.10) (*где матрица L заменяется на U*) *удовлетворяет неравенству*

$$\rho(H_\omega) \geqslant |\omega - 1|. \qquad \text{(П.2.11)}$$

Доказательство. Так как матрица L является строго нижнетреугольной, то $\det D^{-1} = \det(D - \omega L)^{-1}$, и мы получаем

$$\det H_\omega = \det D^{-1} \det((1-\omega) D + \omega U) =$$
$$= \det((1-\omega) I + \omega D^{-1} U) = (1-\omega)^n,$$

поскольку матрица $D^{-1}U$ является строго верхнетреугольной. Но определитель $\det H_\omega$ равен произведению всех собственных значений матрицы H_ω, откуда легко следует (П.2.11).

Теорема П.2.4 приложима также и к методу SSOR. Чтобы найти правильное расщепление $A = P - Q$, в котором P — симметричная матрица, изучим более общую процедуру попеременных итераций:

$$S\mathbf{x}^{k+\frac{1}{2}} = (S - A)\mathbf{x}^k + \mathbf{b}, \qquad \text{(П.2.12a)}$$

$$S^T\mathbf{x}^{k+1} = (S^T - A)\mathbf{x}^{k+\frac{1}{2}} + \mathbf{b}. \qquad \text{(П.2.12b)}$$

Комбинируя два уравнения (П.2.12), получаем одношаговую итерацию

$$\mathbf{x}^{k+1} = (I - S^{-T}A)(I - S^{-1}A)\mathbf{x}^k + ((I - S^{-T}A)S^{-1} + S^{-T})\mathbf{b}. \qquad \text{(П.2.13)}$$

Если предполагается, что этот итерационный процесс соответствует расщеплению $A = P - Q$, то мы должны получить

$$P^{-1} = ((I - S^{-T}A)S^{-1} + S^{-T}) = S^{-T}(S + S^T - A)S^{-1},$$

или

$$P = S(S + S^T - A)^{-1} S^T. \qquad \text{(П.2.14)}$$

Полученная матрица P, очевидно, симметрична, и соответствующая матрица Q имеет вид

$$Q = P - A = S(S + S^T - A)^{-1} S^T - A. \qquad (\text{П.2.15})$$

Матрицы P и Q, заданные соотношениями (П.2.14) и (П.2.15), удовлетворяют равенству

$$P^{-1}Q = I - P^{-1}A = I - S^{-T}(S + S^T - A)S^{-1}A =$$
$$= I - S^{-T}A - S^{-1}A + S^{-T}AS^{-1}A = (I - S^{-T}A)(I - S^{-1}A),$$

т. е., как и следовало ожидать, получается матрица перехода схемы (П.2.13).

Для метода SSOR имеем $S = \omega^{-1}D - L$ и (П.2.12a) принимает вид

$$(\omega^{-1}D - L)\mathbf{x}^{k+\frac{1}{2}} = (\omega^{-1}D - L - D + L + L^T)\mathbf{x}^k + \mathbf{b} =$$
$$= ((\omega^{-1} - 1)D + L^T)\mathbf{x}^k + \mathbf{b}.$$

Аналогичное соотношение получаем из (П.2.12b). Заметим, что матрица S определена путем вынесения за скобку множителя ω из выражения $D - \omega L$, так что в (П.2.12) $\mathbf{b}$ не умножается на ω. Для такой матрицы S соотношение (П.2.14) принимает следующий вид:

$$P = (\omega^{-1}D - L)(\omega^{-1}D - L + \omega^{-1}D - L^T - D + L + L^T)^{-1}(\omega^{-1}D - L^T) =$$
$$= \omega^{-2}(D - \omega L)((2\omega^{-1} - 1)D)^{-1}(D - \omega L^T) =$$
$$= \frac{1}{\omega(2 - \omega)}(D - \omega L)D^{-1}(D - \omega L^T). \qquad (\text{П.2.16})$$

Нетрудно убедиться в том, что соответствующая матрица Q в (П.2.15) имеет вид

$$Q = \frac{1}{\omega(2 - \omega)}((1 - \omega)D + \omega L)D^{-1}((1 - \omega)D + \omega L^T). \qquad (\text{П.2.17})$$

Заметим, что матрицы P и Q в формулах (П.2.16), (П.2.17) можно получить из выражения для матрицы перехода метода SSOR

$$H = (D - \omega L^T)^{-1}((1 - \omega)D + \omega L)(D - \omega L)^{-1}((1 - \omega)D + \omega L^T). \qquad (\text{П.2.18})$$

Два средних множителя в (П.2.18) можно преобразовать к виду

$$((1-\omega)D+\omega L)(D-\omega L)^{-1}=\left((1-\omega)I+\omega LD^{-1}\right)\left(I-\omega LD^{-1}\right)^{-1}=$$
$$=\left(I-\omega LD^{-1}\right)^{-1}\left((1-\omega)I+\omega LD^{-1}\right)=$$
$$=D(D-\omega L)^{-1}((1-\omega)D+\omega L)D^{-1},$$

так как сомножители вида $(I+B)^{-1}(aI+bB)$ перестановочны. Таким образом, получаем

$$H=\left(D-\omega L^{\mathrm{T}}\right)^{-1}D(D-\omega L)^{-1}((1-\omega)D+\omega L)D^{-1}((1-\omega)D+\omega L^{\mathrm{T}}),$$

т. е. те же два сомножителя P^{-1} и Q с точностью до константы $\omega(2-\omega)$.

Если матрица A положительно определена и $\omega\in(0, 2)$, то матрица P в (П.2.16) также положительно определена, а матрица Q в (П.2.17) положительно определена за исключением случая $\omega=1$, когда она положительно полуопределена. В любом случае матрица $P+Q$ положительно определена, т. е. $A=P-Q$ является P-регулярным расщеплением. Таким образом, можно применить обе теоремы П.2.4 и П.2.5 и суммировать свойства сходимости метода SSOR следующим образом.

П.2.11. Теорема о сходимости метода SSOR. *Если матрица A симметрична и невырожденна, а ее диагональные элементы положительны, то при $0<\omega<2$ итерации метода SSOR сходятся для любого $\mathbf{x}^0$ в том и только в том случае, если A положительно определена.*

Заметим, что если A — симметричная положительно определенная матрица, то собственные значения матрицы перехода метода SSOR вещественны и положительны при $\omega\neq 1$. Это вытекает из утверждения П.2.7 и положительной определенности P и Q. (Если $\omega=1$, то эти собственные значения всего лишь неотрицательны.) Аналогичным образом из П.2.7 вытекает вещественность собственных значений матрицы перехода метода Якоби.

Регулярные расщепления и диагональное преобладание

Предыдущие результаты были получены при условии симметричности и положительной определенности матрицы коэффициентов A. Другой стандартный подход к выводу теорем о сходимости итераций методов Якоби и Гаусса — Зейделя использует предположение о том, что матрица A имеет диагональное преобладание или является M-матрицей. В этих случаях предположение о симметрии не является необходимым. Напомним

следующие определения, которые будут использованы в дальнейшем.

П.2.12. Определения. Вещественная матрица A размера $n \times n$ называется

a) *имеющей диагональное преобладание*, если

$$|a_{ii}| \geqslant \sum_{j \neq i} |a_{ij}|, \quad i = 1, \ldots, n, \tag{П.2.19}$$

и *имеющей строгое диагональное преобладание*, если в (П.2.19) строгие неравенства выполняются для всех i;

b) *приводимой*, если существует такая матрица перестановки P, что

$$PAP^{\mathrm{T}} = \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix},$$

и *неприводимой*, если она не является приводимой;

c) *неприводимой с диагональным преобладанием*, если она неприводима, имеет диагональное преобладание и в (П.2.19) строгое неравенство выполняется хотя бы для одного значения i;

d) *неотрицательной* $(A \geqslant 0)$, если все ее элементы неотрицательны, и *положительной* $(A > 0)$, если все ее элементы положительны [1]);

e) M-матрицей, если $a_{ij} \leqslant 0$ для всех $i \neq j$ и, кроме того, $A^{-1} \geqslant 0$.

Рассмотрим сначала следующие классы расщеплений.

П.2.13. Определение. Представление $A = P - Q$ называется *слабо регулярным расщеплением*, если матрица P невырождена, $P^{-1} \geqslant 0$ и $P^{-1}Q \geqslant 0$, и *регулярным расщеплением*, если $P^{-1} \geqslant 0$ и $Q \geqslant 0$.

Ясно, что всякое регулярное расщепление является также слабо регулярным. Основная теорема о сходимости для таких расщеплений формулируется следующим образом.

П.2.14. Теорема о слабо регулярном расщеплении. *Предположим, что матрица A невырождена, матрица A^{-1} неотрицательна и $A = P - Q$ является слабо регулярным расщеплением. Тогда $\rho(P^{-1}Q) < 1$.*

---

[1]) Необходимо четко отличать введенное свойство положительности (неотрицательности) от упоминавшихся выше свойств положительной определенности и положительной полуопределенности; несмотря на созвучие терминов, они обозначают никак не связанные между собой свойства матриц. Подчеркнем также, что вплоть до конца этого пункта матричные неравенства понимаются как поэлементные (см. П.2.12d). — *Прим. перев.*

Доказательство. Положим $H = P^{-1}Q$. Тогда $H \geqslant 0$ и, поскольку

$$(I + H + \ldots + H^m)(I - H) = I - H^{m+1}, \quad P^{-1} = (I - H)A^{-1}$$

и $A^{-1} \geqslant 0$, получаем

$$0 \leqslant (I + H + \ldots + H^m)P^{-1} = (I - H^{m+1})A^{-1} \leqslant A^{-1}$$

для всех $m \geqslant 0$. Поскольку $P^{-1} \geqslant 0$, каждая строка матрицы P^{-1} должна содержать по крайней мере один положительный элемент; отсюда следует, что все элементы матрицы $I + H + \ldots + H^m$ ограничены при $m \to \infty$. Поэтому в силу $H \geqslant 0$ указанный матричный ряд сходится и, следовательно, $H^k \to 0$ при $k \to \infty$. Таким образом, в силу теоремы П.2.1 заключаем, что $\rho(H) < 1$. □

Теперь можно применить утверждение П.2.14 к итерациям Якоби и Гаусса — Зейделя в случае, когда A является M-матрицей. Пусть

$$A = D - L - U, \quad B = L + U, \tag{П.2.20}$$

где D — диагональная часть матрицы A, $(-L)$ — ее строго нижняя треугольная часть, а $(-U)$ — строго верхняя треугольная часть. Если A является M-матрицей, то $B \geqslant 0$ и

$$I = (D - B)A^{-1} = DA^{-1} - BA^{-1} \leqslant DA^{-1},$$

что доказывает положительность диагональных элементов матрицы D. Таким образом, итерации Якоби и Гаусса — Зейделя определены корректно.

П.2.15. Теорема. *Пусть A является M-матрицей. Тогда итерации Якоби и Гаусса — Зейделя сходятся для любого $\mathbf{x}^0$ к единственному решению системы $A\mathbf{x} = \mathbf{b}$.*

Доказательство. Для итераций Якоби расщепление $A = D - B$, очевидно, является регулярным, так что применима теорема П.2.14. Для итераций Гаусса — Зейделя расщепление имеет вид $A = (D - L) - U$. Так как матрица D невырожденна, то $D - L$ также невырожденна и

$$(D - L)^{-1} = (I + D^{-1}L + (D^{-1}L)^2 + \ldots + (D^{-1}L)^{n-1})D^{-1} \geqslant 0, \tag{П.2.21}$$

где ряд является конечным в силу того, что матрица $D^{-1}L$ — строго нижнетреугольная. Таким образом, $A = (D - L) - U$ является регулярным расщеплением и снова можно применить теорему П.2.14. □

Сформулируем теперь теоремы сходимости, применимые в том случае, когда матрица A имеет диагональное преобладание. Сначала нам потребуется следующий результат из теории M-матриц.

П.2.16. Лемма. *Пусть матрица A имеет строгое диагональное преобладание или является неприводимой с диагональным преобладанием, и пусть $a_{ii} > 0$, $i = 1, \ldots, n$, и $a_{ij} \leqslant 0$, $i \neq j$. Тогда A является M-матрицей.*

Доказательство. Пусть $H = I - D^{-1}A$ — матрица перехода итераций Якоби. Тогда в силу утверждения [Ortega, 1987a, теорема 6.1.6] A будет M-матрицей, если $\rho(H) < 1$. В случае, когда матрица A имеет строгое диагональное преобладание, $\rho(H) \leqslant \|H\|_\infty < 1$, и если матрица A неприводима с диагональным преобладанием, то оценка $\rho(H) < 1$ следует из утверждения [Ortega, 1987a, теорема 6.1.10]. □

П.2.17. Теорема о диагональном преобладании. *Пусть матрица A имеет строгое диагональное преобладание или является неприводимой с диагональным преобладанием. Тогда как итерации Якоби, так и итерации Гаусса — Зейделя сходятся для любого $\mathbf{x}^0$.*

Доказательство. Пусть

$$\hat{A} = |D| - |L| - |U|,$$

где $|L|$ обозначает матрицу с элементами, равными модулям соответствующих элементов матрицы L (аналогично для матриц $|D|$ и $|U|$). Ясно, что если A имеет строгое диагональное преобладание или является неприводимой с диагональным преобладанием, то тем же свойством обладает и матрица $\hat{A}$. Таким образом, в силу П.2.16 $\hat{A}$ является M-матрицей, и в силу П.2.15

$$\rho\left(|D|^{-1}(|L| + |U|)\right) < 1,$$
$$\rho\left((|D| - |L|)^{-1}|U|\right) < 1.$$

Но

$$\left|D^{-1}(L+U)\right| \leqslant |D|^{-1}(|L| + |U|)$$

и

$$\left|(D-L)^{-1}U\right| \leqslant (|D| - |L|)^{-1}|U|,$$

где второе неравенство следует из (П.2.21). Затем из теорем сравнения для спектральных радиусов (если $|B| \leqslant C$, то $\rho(B) \leqslant \rho(C)$; см., например, [Varga, 1962]) следует, что

$$\rho\left(D^{-1}(L+U)\right) \leqslant \rho\left(|D|^{-1}(|L| + |U|)\right) < 1,$$
$$\rho\left((D-L)^{-1}U\right) \leqslant \rho\left((|D| - |L|)^{-1}|U|\right) < 1. \qquad \square$$

Сходимость метода ADI

Докажем теперь теорему 3.1.3 о сходимости итераций метода ADI

$$(\alpha I + H)\,\mathbf{x}^{k+\frac{1}{2}} = (\alpha I - V)\,\mathbf{x}^{k} + \mathbf{b}, \quad k = 0, 1, \ldots, \qquad (\text{П.2.22a})$$

$$(\alpha I + V)\,\mathbf{x}^{k+1} = (\alpha I - H)\,\mathbf{x}^{k+\frac{1}{2}} + \mathbf{b}, \quad k = 0, 1, \ldots. \qquad (\text{П.2.22b})$$

П.2.18. Теорема о сходимости ADI. *Пусть A, H и V — симметричные положительно определенные матрицы, причем $A = H + V$, и пусть $\alpha > 0$. Тогда итерационный процесс* (П.2.22) *корректно определен и сходится к единственному решению системы $A\mathbf{x} = \mathbf{b}$.*

Доказательство. Матрица перехода для (П.2.22) имеет вид

$$S = (\alpha I + V)^{-1}(\alpha I - H)(\alpha I + H)^{-1}(\alpha I - V),$$

и достаточно показать, что $\rho(S) < 1$. Так как H и V положительно определены и $\alpha > 0$, матрицы $\alpha I + H$ и $\alpha I + V$ невырождены и итерации (П.2.22) корректно определены. Положим

$$F = (\alpha I - H)(\alpha I + H)^{-1}, \quad G = (\alpha I - V)(\alpha I + V)^{-1}.$$

Если $\lambda_n \geqslant \ldots \geqslant \lambda_1 > 0$ — собственные значения матрицы H, то $(\alpha - \lambda_i)/(\alpha + \lambda_i)$ являются собственными значениями матрицы F; очевидно, что они по модулю меньше единицы. Так как матрицы $\alpha I - H$ и $(\alpha I + H)^{-1}$ перестановочны, то матрица F симметрична и в норме l_2 справедлива оценка

$$\|F\|_2 = \max_i \frac{|\alpha - \lambda_i|}{\alpha + \lambda_i} < 1.$$

Аналогичным образом получаем, что $\|G\|_2 < 1$. Отсюда следует

$$\|FG\|_2 \leqslant \|F\|_2 \|G\|_2 < 1,$$

и поэтому спектральный радиус матрицы FG удовлетворяет неравенству $\rho(FG) < 1$. Но матрица S подобна FG в силу

$$S = (\alpha I + V)^{-1} FG\,(\alpha I + V)$$

и, таким образом, $\rho(S) < 1$. □

Скорость сходимости

Для итерационного процесса (П.2.1), удовлетворяющего условию $\rho(H) < 1$, *асимптотический множитель сходимости* определяется как

$$\alpha = \sup_{\mathbf{x}} \left(\limsup_{k \to \infty} \| \mathbf{x}^k - \hat{\mathbf{x}} \|^{1/k} \right), \qquad (\text{П.2.23})$$

где $\hat{\mathbf{x}}$ удовлетворяет уравнению $\hat{\mathbf{x}} = H\hat{\mathbf{x}} + \mathbf{d}$.

Чтобы лучше понять смысл определения (П.2.23), рассмотрим отдельно одну из последовательностей и положим

$$\beta = \limsup_{k \to \infty} \| \mathbf{x}^k - \hat{\mathbf{x}} \|^{1/k}.$$

Так как $\mathbf{x}^k \to \hat{\mathbf{x}}$ при $k \to \infty$, ясно, что $\beta \leqslant 1$ и что для любого $\varepsilon > 0$ найдется такой номер k_0, что

$$\| \mathbf{x}^k - \hat{\mathbf{x}} \| \leqslant (\beta + \varepsilon)^k, \quad k \geqslant k_0.$$

Таким образом, если $\beta < 1$, то мы можем выбрать такое ε, что $\beta + \varepsilon < 1$ и, следовательно, норма $\| \mathbf{x}^k - \hat{\mathbf{x}} \|$ стремится к нулю (асимптотически) по крайней мере так же быстро, как геометрическая прогрессия $(\beta + \varepsilon)^k$. Точная верхняя грань в (П.2.23) берется для того, чтобы учесть наихудшее возможное поведение любой отдельно взятой последовательности.

Основная теорема об асимптотическом множителе сходимости итераций (П.2.1) формулируется следующим образом.

П.2.19. Теорема. *Асимптотический множитель сходимости в любой норме равен* $\rho(H)$.

Строгое доказательство этого утверждения можно найти, например, в [Ortega, 1972]. Мы ограничимся неформальным обсуждением, носящим иллюстративный характер.

Заметим сначала, что из теоремы П.2.19 следует, что в любой норме для любой последовательности $\{\mathbf{x}^k\}$ и произвольного $\varepsilon > 0$, такого, что $\rho(H) + \varepsilon < 1$, найдется такой номер k_0, что

$$\| \mathbf{x}^k - \hat{\mathbf{x}} \| \leqslant (\rho(H) + \varepsilon)^k, \quad k \geqslant k_0. \qquad (\text{П.2.24})$$

Таким образом, асимптотически погрешности убывают почти так же быстро, как $(\rho(H))^k$. Важно отметить, что указанное убывание погрешности будет, вообще говоря, лишь асимптотическим. Рассмотрим, например, матрицу

$$H = \begin{bmatrix} 0.5 & a \\ 0 & 0 \end{bmatrix},$$

для которой $\rho(H) = 0.5$. Но

$$H^k = \begin{bmatrix} (0.5)^k & a(0.5)^{k-1} \\ 0 & 0 \end{bmatrix},$$

и поэтому при $\mathbf{e}^k = \mathbf{x}^k - \hat{\mathbf{x}}$ и $\mathbf{e}^0 = (0, 1)^T$ получаем, что

$$\mathbf{e}^k = H^k \mathbf{e}^0 = (a(0.5)^{k-1}, 0)^T.$$

Предположим, что $a = 2^p$. Тогда

$$\|\mathbf{e}^k\|_2 = 2^{p-k+1} \geqslant \|\mathbf{e}^0\|_2 \quad \text{для} \quad k \leqslant p+1,$$

так что первые p погрешностей превосходят в 2-норме начальную погрешность. Это показывает, что величина $\rho(H)$ может не являться мерой скорости сходимости по отношению к какой-либо общеупотребительной норме на начальной стадии итерационного процесса.

Проиллюстрируем теперь утверждение теоремы П.2.19 следующим образом. Предположим, что матрица H имеет n линейно независимых собственных векторов $\mathbf{v}_1, \ldots, \mathbf{v}_n$, отвечающих n собственным значениям, упорядоченным таким образом, что $|\lambda_1| \geqslant |\lambda_2| \geqslant \ldots \geqslant |\lambda_n|$. Тогда для любой последовательности $\mathbf{x}^k$, порождаемой итерационным процессом (П.2.1), погрешности $\mathbf{e}^k = \mathbf{x}^k - \hat{\mathbf{x}}$ допускают представление

$$\mathbf{e}^k = H^k \mathbf{e}^0 = \alpha_1 \lambda_1^k \mathbf{v}_1 + \ldots + \alpha_n \lambda_n^k \mathbf{v}_n = \lambda_1^k \left(\alpha_1 \mathbf{v}_1 + \sum_{i=2}^{n} \left(\frac{\lambda_i}{\lambda_1} \right)^k \alpha_i \mathbf{v}_i \right), \tag{П.2.25}$$

где $\mathbf{e}^0 = \alpha_1 \mathbf{v}_1 + \ldots + \alpha_n \mathbf{v}_n$. Предположим теперь, что $|\lambda_1| > |\lambda_2|$. Тогда

$$\frac{\mathbf{e}^k}{\lambda_1^k} \to \alpha_1 \mathbf{v}_1 \quad \text{при} \quad k \to \infty. \tag{П.2.26}$$

Можно дать следующую интерпретацию соотношения (П.2.26): векторы погрешности $\mathbf{e}^k$ сходятся по направлению к вектору $\mathbf{v}_1$, и указанные погрешности асимптотически ведут себя как $\lambda_1^k \alpha_1 \mathbf{v}_1$. Таким образом, погрешности (асимптотически) убывают, так как умножаются на каждой итерации на множитель $\rho(H) = |\lambda_1|$.

Заметим, что если $\alpha_1 = 0$, то скорость сходимости рассматриваемой последовательности определяется величиной $|\lambda_2| < \rho(H)$. Указанное обстоятельство представляется, однако, несущественным, так как весьма маловероятно, что начальное

приближение $\mathbf{x}^0$ окажется таким, что $\alpha_1 = 0$. Даже если указанная ситуация реализуется, ошибки округления в процессе вычислений вызовут эффект, эквивалентный появлению коэффициента $\alpha_1 \neq 0$ на последующих итерациях.

Допустим теперь, что $|\lambda_1| = |\lambda_2| > |\lambda_3|$. В этом случае

$$\mathbf{e}^k = \alpha_1 \lambda_1^k \mathbf{v}_1 + \alpha_2 \lambda_2^k \mathbf{v}_2 + \binom{\text{пренебрежимо}}{\text{малые}\quad\text{члены}},$$

Если $\lambda_1 = \lambda_2$, то $\mathbf{e}^k$ стремится по направлению к $\alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2$. В противном случае вектор $\mathbf{e}^k$ не обязан стремиться к какому-либо фиксированному направлению; здесь будет наблюдаться «стремление» к подпространству, натянутому на $\mathbf{v}_1$ и $\mathbf{v}_2$. Например, если $\lambda_1 = -\lambda_2$, то

$$\mathbf{e}^k = \lambda_1^k (\alpha_1 \mathbf{v}_1 + (-1)^k \alpha_2 \mathbf{v}_2) + \binom{\text{пренебрежимо}}{\text{малые}\quad\text{члены}}.$$

Аналогично, если $|\lambda_1| = \ldots = |\lambda_p| > |\lambda_{p+1}|$, то $\mathbf{e}^k$ стремится к подпространству, натянутому на $\mathbf{v}_1, \ldots, \mathbf{v}_p$. В любом случае асимптотическое убывание погрешности характеризуется множителем $\rho(H)$.

Предположим теперь, что жорданова форма матрицы H не является диагональной; допустим, в частности, что $\lambda_1 = \ldots = \lambda_p$ и что собственное значение λ_1 связано с жордановым блоком $p \times p$. Пусть $\mathbf{v}_1$ — собственный вектор, отвечающий λ_1, а $\mathbf{v}_2, \ldots, \mathbf{v}_p$ — корневые векторы, такие, что

$$H\mathbf{v}_i = \lambda_1 \mathbf{v}_i + \mathbf{v}_{i-1}, \quad i = 2, \ldots, p \qquad (\text{П.2.27})$$

(см., например, [Ortega, 1987a, § 3.2]). Из (П.2.27) следует, что

$$H^k \mathbf{v}_i = \lambda_1^k \mathbf{v}_i + k\lambda_1^{k-1} \mathbf{v}_{i-1} + \ldots + \binom{k}{i-1} \lambda_1^{k-i+1} \mathbf{v}_1, \qquad (\text{П.2.28})$$

где $\binom{k}{j} = \dfrac{k!}{(k-j)!\,j!}$ — биномиальные коэффициенты.

Рассмотрим для простоты случай $p = 2$, т. е. $|\lambda_1| = |\lambda_2| > |\lambda_3| \geqslant \ldots \geqslant |\lambda_n|$. Используя (П.2.28) при $i = 2$, получаем

$$\mathbf{e}^k = H^k \mathbf{e}^0 = \alpha_1 \lambda_1^k \mathbf{v}_1 + \alpha_2 (\lambda_1^k \mathbf{v}_2 + k\lambda_1^{k-1} \mathbf{v}_1) + \sum_{i=3}^{n} \alpha_i H^k \mathbf{v}_i =$$

$$= \lambda_1^k \left(\left(\alpha_1 + \alpha_2 \frac{k}{\lambda_1} \right) \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \frac{1}{\lambda_1^k} \sum_{i=3}^{n} \alpha_i H^k \mathbf{v}_i \right), \qquad (\text{П.2.29})$$

где, как и прежде, $\mathbf{e}^0 = \alpha_1 \mathbf{v}_1 + \ldots + \alpha_n \mathbf{v}_n$, а $\mathbf{v}_3, \ldots, \mathbf{v}_n$ — остальные собственные или корневые векторы. Так как $|\lambda_1| > |\lambda_i|$

для $i = 3, \ldots, n$, из того, что

$$\binom{k}{j}\left|\frac{\lambda_i}{\lambda_1}\right|^k \to 0 \quad \text{при} \quad k \to \infty, \quad i = 3, \ldots, n,$$

следует, что

$$\lambda_1{}^k H^k \mathbf{v}_i \to 0 \quad \text{при} \quad k \to \infty, \quad i = 3, \ldots, n.$$

Таким образом, асимптотическое поведение погрешности характеризуется соотношением

$$\mathbf{e}^k \sim \lambda_1^k \left(\left(\alpha_1 + \alpha_2 \frac{k}{\lambda_1}\right)\mathbf{v}_1 + \alpha_2 \mathbf{v}_2\right) \sim \alpha_2 k \lambda_1^{k-1} \mathbf{v}_1. \quad (\text{П.2.30})$$

Тем самым показано, что векторы погрешности сходятся по направлению к вектору $\mathbf{v}_1$, однако из-за наличия множителя k асимптотическое убывание погрешности уже не характеризуется одним лишь умножением на коэффициент $|\lambda_1| = \rho(H)$. Тем не менее величина $\rho(H)$ остается наилучшей скалярной характеристикой скорости сходимости, так как для любого $\delta > 0$

$$\frac{k\lambda_1^k}{(|\lambda_1| + \delta)^k} \to 0 \quad \text{при} \quad k \to \infty.$$

Таким образом, погрешность стремится к нулю асимптотически по меньшей мере так же быстро, как и μ^k, где величина μ сколь угодно близка к $\rho(H)$.

В общем случае результат выглядит аналогично. Если собственные значения $\lambda_1 = \ldots = \lambda_p$ связаны с жордановым блоком $p \times p$ и $|\lambda_p| > |\lambda_{p+1}|$, то можно показать, действуя так же, как и выше, что

$$\mathbf{e}^k \sim ck^{p-1}\lambda_1^k \mathbf{v}_1.$$

Поскольку

$$\frac{k^{p-1}\lambda_1^k}{(|\lambda_1| + \delta)^k} \to 0 \quad \text{при} \quad k \to \infty$$

для любого $\delta > 0$, мы опять видим, что погрешность стремится к нулю не медленнее, чем $(\rho(H) + \delta)^k$, для любого $\delta > 0$. Если имеются несколько жордановых блоков равного размера, связанных с собственным значением, которое является наибольшим по модулю, то справедлив тот же результат, хотя в этом случае погрешность стремится уже к подпространству, натянутому на соответствующие собственные векторы.

На основе утверждения теоремы П.2.19 можно заключить, что асимптотическая скорость сходимости итерационного метода (П.2.1) тем выше, чем меньше спектральный радиус

$\rho(H)$. Рассмотрим теперь задачу о выборе параметра ω, минимизирующего $\rho(H_\omega)$, где H_ω — матрица перехода метода SOR, определенная соотношением (П.2.10). Вообще эта задача очень трудна, однако если матрица коэффициентов A является матрицей, отвечающей дискретному уравнению Пуассона (3.1.7), или, в более общем случае, является согласованно упорядоченной (см. [Young, 1971]), то решение удается найти, применив изящную теорию, развитую Франкелом и Янгом в начале 50-х годов (см. [Varga, 1962] и [Young, 1971], где дается полное изложение). Основной результат формулируется следующим образом.

П.2.20. Теорема. *Пусть симметричная положительно определенная матрица A является согласованно упорядоченной, и пусть H_ω — матрица перехода метода SOR* (П.2.10). *Пусть также $\mu = \rho(J)$, где J — матрица перехода метода Якоби. Тогда*

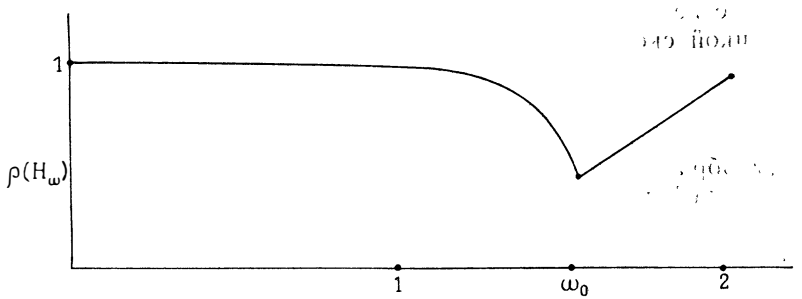


Рис. П.2.1. График зависимости спектрального радиуса SOR от ω

существует единственное значение $\omega \in (0, 2)$, минимизирующее $\rho(H_\omega)$, которое задается формулой

$$\omega_0 = \frac{2}{1 + (1 - \mu^2)^{1/2}}. \qquad (П.2.31)$$

Кроме того,

$$\rho(H_\omega) = \omega - 1, \quad \omega_0 \leqslant \omega \leqslant 2, \qquad (П.2.32a)$$

и

$$\rho(H_\omega) = \frac{1}{4}\left(\omega\mu + (\omega^2\mu^2 - 4(\omega - 1))^{1/2}\right)^2, \quad 0 \leqslant \omega \leqslant \omega_0. \qquad (П.2.32b)$$

Исходя из формул (П.2.32), мы можем изображать $\rho(H_\omega)$ как функцию от ω, как показано на рис. П.2.1. Наклон кривой справа от ω_0 равен единице, а в точке ω_0 слева — бесконечности. Поэтому лучше использовать приближение к ω_0 с избытком, чем с недостатком.

Непосредственно из (П.2.32b) вытекает, что

$$\rho(H_1) = \mu^2; \tag{П.2.33}$$

тем самым устанавливается, что в условиях сформулированной теоремы асимптотическая скорость сходимости метода Гаусса — Зейделя ровно вдвое превосходит асимптотическую скорость сходимости метода Якоби.

Для непосредственного применения формул (П.2.31) и (П.2.32) требуется точное значение μ, которое, вообще говоря, недоступно. Однако для матрицы $N^2 \times N^2$ (3.1.7), отвечающей дискретному уравнению Пуассона, можно показать (см. упражнение 3.1.21), что

$$\mu = \cos \frac{\pi}{N+1}. \tag{П.2.34}$$

Например, при $N = 44$ получаем

$$\mu = 0.99756, \quad \rho(H_1) = 0.99513, \quad \omega_0 = 1.87, \quad \rho(H_{\omega_0}) = 0.87.$$

Так как $\rho(H_{\omega_0}) \approx (\rho(H_1))^{30}$, асимптотическая скорость сходимости при оптимальном значении ω_0 примерно в 30 раз выше, чем для метода Гаусса — Зейделя. В общем случае для указанной задачи Пуассона из (П.2.32b) и (П.2.34) следует, что асимптотическая скорость сходимости метода SOR при $\omega = \omega_0$ приблизительно в $0.63N$ раз превосходит асимптотическую скорость сходимости метода Гаусса — Зейделя.

Приложение 3

Метод сопряженных градиентов

В этом приложении мы рассмотрим различные теоретические результаты, касающиеся метода сопряженных градиентов. Здесь и далее мы предполагаем, что матрица коэффициентов A симметрична и положительно определена. Мы хотим установить, что векторы $\mathbf{p}^k$, порождаемые методом сопряженных градиентов, по существу являются сопряженными направлениями, а также получить оценку погрешности (3.4.1).

Сначала перепишем алгоритм в форме, несколько отличной от приведенной в (3.3.15). Разница заключается в определении β_k; ниже мы покажем, что эти две формы записи эквивалентны. Мы также используем определение α_k, задаваемое отношением (3.3.5), и покажем, что оно эквивалентно использованному в (3.3.15a). Для упрощения обозначений будем указывать номер итерации при помощи нижнего индекса. С перечисленными изменениями алгоритм запишется следующим образом:

Выбираем $\mathbf{x}_0$. Полагаем $\mathbf{p}_0 = \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$.

Для $k = 0, 1, \ldots$

$$\alpha_k = -\mathbf{r}_k^{\mathrm{T}}\mathbf{p}_k / \mathbf{p}_k^{\mathrm{T}} A \mathbf{p}_k. \tag{П.3.1a}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{p}_k, \tag{П.3.1b}$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \alpha_k A \mathbf{p}_k, \tag{П.3.1c}$$

$$\beta_k = -\mathbf{p}_k^{\mathrm{T}} A \mathbf{r}_{k+1} / p_k^{\mathrm{T}} A \mathbf{p}_k, \tag{П.3.1d}$$

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k. \tag{П.3.1e}$$

П.3.1. Теорема. *Пусть A — симметричная положительно определенная матрица $n \times n$, а $\hat{\mathbf{x}}$ — решение системы $A\mathbf{x} = \mathbf{b}$. Тогда векторы $\mathbf{p}_k$, порождаемые алгоритмом* (П.3.1), *удовлетворяют соотношениям*

$$\mathbf{p}_k^{\mathrm{T}} A \mathbf{p}_j = 0, \quad 0 \leqslant j < k, \quad k = 1, \ldots, n-1, \tag{П.3.2}$$

и $\mathbf{p}_k \neq 0$, за исключением случая $\mathbf{x}_k = \hat{\mathbf{x}}$. Таким образом, $\mathbf{x}_m = \hat{\mathbf{x}}$ для некоторого $m \leqslant n$.

Доказательство. Мы докажем не только формулы (П.3.2), но и сопутствующие соотношения для невязок $\mathbf{r}_j = \mathbf{b} - A\mathbf{x}_j$, имеющие вид

$$\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_j = 0, \quad 0 \leqslant j < k, \quad k = 1, \ldots, n-1. \tag{П.3.3}$$

Заметим сначала, что по определению (П.3.1a) и (П.3.1d) параметров α_j, β_j и в силу (П.3.1c) и (П.3.1e)

$$\mathbf{p}_j^{\mathrm{T}}\mathbf{r}_{j+1} = \mathbf{p}_j^{\mathrm{T}}\mathbf{r}_j + \alpha_j \mathbf{p}_j^{\mathrm{T}} A\mathbf{p}_j = 0, \quad j = 0, 1, \ldots, \tag{П.3.4a}$$

$$\mathbf{p}_j^{\mathrm{T}} A\mathbf{p}_{j+1} = \mathbf{p}_j^{\mathrm{T}} A\mathbf{r}_{j+1} + \beta_j \mathbf{p}_j^{\mathrm{T}} A\mathbf{p}_j = 0, \quad j = 0, 1, \ldots. \tag{П.3.4b}$$

Примем теперь в качестве индуктивного предположения, что соотношения (П.3.2) и (П.3.3) справедливы для некоторого $k < n-1$. Покажем, что тогда эти соотношения справедливы для $k+1$. Так как $\mathbf{p}_0 = \mathbf{r}_0$, (П.3.4) показывает, что они выполняются для $k = 1$. Для любого $j < k$ в силу (П.3.1c, e) имеем

$$\begin{aligned}\mathbf{r}_j^{\mathrm{T}}\mathbf{r}_{k+1} &= \mathbf{r}_j^{\mathrm{T}}(\mathbf{r}_k + \alpha_k A\mathbf{p}_k) = \mathbf{r}_j^{\mathrm{T}}\mathbf{r}_k + \alpha_k \mathbf{p}_k^{\mathrm{T}} A\mathbf{r}_j = \\ &= \mathbf{r}_j^{\mathrm{T}}\mathbf{r}_k + \alpha_k \mathbf{p}_k^{\mathrm{T}} A(\mathbf{p}_j - \beta_{j-1}\mathbf{p}_{j-1}) = 0,\end{aligned}$$

так как все три члена в правой части обращаются в нуль по индуктивному предположению. Кроме того, используя (П.3.1e), (П.3.4a), а затем (П.3.1c), получаем

$$\begin{aligned}\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_{k+1} &= (\mathbf{p}_k - \beta_{k-1}\mathbf{p}_{k-1})^{\mathrm{T}}\mathbf{r}_{k+1} = -\beta_{k-1}\mathbf{p}_{k-1}^{\mathrm{T}}\mathbf{r}_{k+1} = \\ &= -\beta_{k-1}\mathbf{p}_{k-1}^{\mathrm{T}}(\mathbf{r}_k + \alpha_k A\mathbf{p}_k) = 0,\end{aligned}$$

так как последние два члена равны нулю в соответствии с (П.3.4). Таким образом, мы показали, что (П.3.3) выполняется для $k+1$.

Далее, для любого $j < k$ получаем, используя (П.3.1e), индуктивное предположение, (П.3.1c), а затем (П.3.2), что

$$\mathbf{p}_j^{\mathrm{T}} A\mathbf{p}_{k+1} = \mathbf{p}_j^{\mathrm{T}} A(\mathbf{r}_{k+1} + \beta_k \mathbf{p}_k) = \mathbf{p}_j^{\mathrm{T}} A\mathbf{r}_{k+1} = \alpha_j^{-1}(\mathbf{r}_{j+1} - \mathbf{r}_j)^{\mathrm{T}}\mathbf{r}_{k+1} = 0.$$

Здесь предполагается, что $\alpha_j \neq 0$; к этому условию мы скоро вернемся. В силу (П.3.4b) получаем также $\mathbf{p}_k^{\mathrm{T}} A\mathbf{p}_{k+1} = 0$. Таким образом, (П.3.2) выполняется для $k+1$, и индуктивный переход завершен.

Покажем теперь, что векторы $\mathbf{p}_k$ остаются ненулевыми до тех пор, пока не найдено решение. Допустим, что $\mathbf{p}_m = 0$ для некоторого $m < n$. Тогда, используя (П.3.1e), получаем

$$\begin{aligned}0 = \mathbf{p}_m^{\mathrm{T}}\mathbf{p}_m &= (\mathbf{r}_m + \beta_{m-1}\mathbf{p}_{m-1})^{\mathrm{T}}(\mathbf{r}_m + \beta_{m-1}\mathbf{p}_{m-1}) = \\ &= \mathbf{r}_m^{\mathrm{T}}\mathbf{r}_m + 2\beta_{m-1}\mathbf{r}_m^{\mathrm{T}}\mathbf{p}_{m-1} + \beta_{m-1}^2 \mathbf{p}_{m-1}^{\mathrm{T}}\mathbf{p}_{m-1} \geqslant \mathbf{r}_m^{\mathrm{T}}\mathbf{r}_m,\end{aligned}$$

так как $\mathbf{r}_m^{\mathrm{T}}\mathbf{p}_{m-1} = 0$ в силу (П.3.4a). Таким образом, $\mathbf{r}_m = \mathbf{b} - A\mathbf{x}_m = 0$, так что $\mathbf{x}_m = \hat{\mathbf{x}}$. С другой стороны, если все векторы $\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}$ ненулевые, то $\mathbf{x}_n = \hat{\mathbf{x}}$ в силу теоремы 3.3.1 о сопряженных направлениях.

Наконец, вернемся к предположению о том, что $\alpha_j \neq 0$. В силу (П.3.1e) и (П.3.4b) получаем

$$\mathbf{r}_j^{\mathrm{T}}\mathbf{p}_j = \mathbf{r}_j^{\mathrm{T}}(\mathbf{r}_j + \beta_{j-1}\mathbf{p}_{j-1}) = \mathbf{r}_j^{\mathrm{T}}\mathbf{r}_j. \tag{П.3.5}$$

Отсюда находим, что определение α_j (П.3.1a) эквивалентно

$$\alpha_j = -\mathbf{r}_j^{\mathrm{T}}\mathbf{r}_j / \mathbf{p}_j^{\mathrm{T}} A\mathbf{p}_j. \tag{П.3.6}$$

Следовательно, если $\alpha_j = 0$, то $\mathbf{r}_j = 0$ и, как было показано выше, $\mathbf{x}_j = \hat{\mathbf{x}}$, так что итерационный процесс останавливается на вычислении $\mathbf{x}_j$. □

Покажем теперь что определение параметра β_k, использованное в (П.3.1d), эквивалентно данному в § 3.3. Действительно, из (П.3.1c), (П.3.4b), (П.3.1c) и (П.3.3) следует, что

$$\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{p}_{k+1} = (\mathbf{r}_k + \alpha_k A\mathbf{p}_k)^{\mathrm{T}}\mathbf{p}_{k+1} = \mathbf{r}_k^{\mathrm{T}}\mathbf{p}_{k+1} = \mathbf{r}_k^{\mathrm{T}}(\mathbf{r}_{k+1} + \beta_k\mathbf{p}_k) = \beta_k\mathbf{r}_k^{\mathrm{T}}\mathbf{p}_k.$$

Отсюда, используя (П.3.5), получаем

$$\beta_k = \mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{p}_{k+1} / \mathbf{r}_k^{\mathrm{T}}\mathbf{p}_k = \mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1} / \mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k, \tag{П.3.7}$$

что совпадает с формулой для β_k в (3.3.15e). Выражение для α_k и β_k вида (П.3.6) и (П.3.7) предпочтительнее с вычислительной точки зрения, так как на каждом шаге требуется вычислять лишь скалярное произведение $\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k$. Выражения (П.3.1a, d) иногда оказываются полезными для теоретических исследований.

Следующим шагом будет вывод некоторых основных соотношений, которые требуются для доказательства оценки погрешности (3.4.1) и позволяют более глубоко осмыслить метод сопряженных градиентов. Напомним, что через $\operatorname{span}(\mathbf{y}_1, \ldots, \mathbf{y}_m)$ обозначается подпространство, порожденное всевозможными линейными комбинациями векторов $\mathbf{y}_1, \ldots, \mathbf{y}_m$.

П.3.2. Теорема. *Пусть* $\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}$ — *векторы направлений, порожденные методом сопряженных градиентов, а* $\mathbf{r}_0, \ldots, \mathbf{r}_{n-1}$ — *векторы невязок. Тогда*

$$A\mathbf{p}_i \in \operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_{i+1}), \quad i = 0, \ldots, n-2, \tag{П.3.8}$$

$$\mathbf{r}_i \in \operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_i), \quad i = 0, \ldots, n-1, \tag{П.3.9}$$

$$\operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_i) = \operatorname{span}(\mathbf{p}_0, A\mathbf{p}_0, \ldots, A^i\mathbf{p}_0) = \operatorname{span}(\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^i\mathbf{r}_0), \quad i = 0, \ldots, n-1. \tag{П.3.10}$$

Доказательство. Соотношения (П.3.8) и (П.3.9) докажем совместно по методу математической индукции. В силу (П.3.1c) и (П.3.1e)

$$\mathbf{p}_1 = \mathbf{r}_1 + \beta_0 \mathbf{p}_0 = \mathbf{r}_0 + \alpha_0 A\mathbf{p}_0 + \beta_0 p_0.$$

Используя $\mathbf{r}_0 = \mathbf{p}_0$, получаем

$$A\mathbf{p}_0 = \alpha_0^{-1}(\mathbf{p}_1 - \mathbf{p}_0 - \beta_0 \mathbf{p}_0),$$

т. е. (П.3.8) справедливо для $i = 0$. Так как $\mathbf{r}_0 = \mathbf{p}_0$, для $i = 0$ выполняется также и (П.3.9).

Предположим теперь, что (П.3.8) и (П.3.9) выполняются для $i = 0, \ldots, k < n - 2$. В силу (П.3.1c) и индуктивного предположения имеем

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \alpha_k A\mathbf{p}_k = \sum_{j=0}^{k} \nu_j \mathbf{p}_j + \alpha_k \sum_{j=0}^{k+1} \eta_j \mathbf{p}_j.$$

Таким образом, $\mathbf{r}_{k+1} \in \operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_{k+1})$. Тогда из (П.3.1e) и (П.3.1c) получаем

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+2} + \beta_{k+1}\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \alpha_{k+1} A\mathbf{p}_{k+1} + \beta_{k+1}\mathbf{p}_{k+1}.$$

Поскольку $\mathbf{r}_{k+1} \in \operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_{k+1})$, то $A\mathbf{p}_{k+1} \in \operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_{k+2})$. Таким образом, индуктивное доказательство завершено, и остается только показать, что (П.3.9) выполняется также для $\mathbf{r}_{n-1}$; это делается так же, как и выше.

Чтобы доказать (П.3.10), снова используем индукцию. Для $i = 0$ утверждение тривиально в силу $\mathbf{p}_0 = \mathbf{r}_0$. Предположим, что оно справедливо при $k \leq n - 1$. Тогда в силу индуктивного предположения и (П.3.8)

$$A^{k+1}\mathbf{p}_0 = A(A^k \mathbf{p}_0) = A\left(\sum_{j=0}^{k} \nu_j \mathbf{p}_j\right) = \sum_{j=0}^{k} \nu_j A\mathbf{p}_j \in \operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_{k+1}).$$

Теперь заметим, что для любых η_j справедливо

$$\sum_{j=0}^{k+1} \eta_j A^j \mathbf{p}_0 = \eta_{k+1} A^{k+1}\mathbf{p}_0 + \sum_{j=0}^{k} \eta_j A^j \mathbf{p}_0.$$

Первое слагаемое в правой части принадлежит $\operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_{k+1})$ в силу предыдущего соотношения, а второе — по индуктивному предположению. Таким образом, мы показали, что

$$\operatorname{span}(\mathbf{p}_0, A\mathbf{p}_0, \ldots, A^{k+1}\mathbf{p}_0) \subset \operatorname{span}(\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{k+1}).$$

Чтобы доказать обратное включение, сначала запишем

$$\sum_{j=0}^{k+1} \eta_j \mathbf{p}_j = \eta_{k+1}\mathbf{p}_{k+1} + \sum_{j=0}^{k} \eta_j \mathbf{p}_j.$$

Второе слагаемое принадлежит $\operatorname{span}(\mathbf{p}_0, \ldots, A^k\mathbf{p}_0)$ по индуктивному предположению. Первое слагаемое в силу (П.3.1c) и (П.3.1e) можно представить в виде

$$\eta_{k+1}\mathbf{p}_{k+1} = \eta_{k+1}(\mathbf{r}_{k+1} + \beta_k\mathbf{p}_k) = \eta_{k+1}(\mathbf{r}_k + \alpha_k A\mathbf{p}_k + \beta_k\mathbf{p}_k).$$

Первый и третий члены в правой части этого уравнения принадлежат $\operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_k)$ в силу (П.3.9) и поэтому принадлежат $\operatorname{span}(\mathbf{p}_0, \ldots, A^k\mathbf{p}_0)$ по индуктивному предположению. Во втором члене вектор $A\mathbf{p}_k$ с использованием (П.3.8) и индуктивного предположения можно представить в виде

$$A\mathbf{p}_k = A\sum_{j=0}^{k} \nu_j A^j\mathbf{p}_0 = \sum_{j=0}^{k} \nu_j A^{j+1}\mathbf{p}_0,$$

т. е. он лежит в $\operatorname{span}(\mathbf{p}_0, \ldots, A^{k+1}\mathbf{p}_0)$. Таким образом, мы показали, что $\operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_{k+1}) \subset \operatorname{span}(\mathbf{p}_0, \ldots, A^{k+1}\mathbf{p}_0)$, и первое из соотношений (П.3.10) доказано. Второе соотношение (П.3.10) тривиально в силу $\mathbf{p}_0 = \mathbf{r}_0$. □

Подпространство вида $\operatorname{span}(\mathbf{p}_0, \ldots, A^l\mathbf{p}_0)$ принято называть *подпространством Крылова*. Покажем теперь, что итерации метода сопряженных градиентов минимизируют A-норму погрешности на сдвинутом (иначе говоря, аффинном) подпространстве Крылова. Напомним, что A-норма определяется соотношением $\|\mathbf{y}\|_A^2 = \mathbf{y}^T A\mathbf{y}$.

П.3.3. Минимизирующее свойство метода сопряженных градиентов. *Для каждого* $k = 0, 1, \ldots, n-1$, *приближение* $\mathbf{x}_{k+1}$, *полученное методом сопряженных градиентов, минимизирует* $\|\hat{\mathbf{x}} - \mathbf{x}\|_A$ *на аффинном подпространстве*

$$S_k = \mathbf{x}_0 + \operatorname{span}(\mathbf{p}_0, \ldots, \mathbf{p}_k) = \mathbf{x}_0 + \operatorname{span}(\mathbf{p}_0, A\mathbf{p}_0, \ldots, A^k\mathbf{p}_0).$$

Доказательство. Пусть для заданного k вектор $\mathbf{x} = \mathbf{x}_0 + \sum_{j=0}^{k} \nu_j\mathbf{p}_j$ принадлежит S_k. Так как

$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 = \mathbf{b} - A\mathbf{x}_0 - (\mathbf{b} - A\hat{\mathbf{x}}) = A(\hat{\mathbf{x}} - \mathbf{x}_0),$$

получаем

$$\hat{\mathbf{x}} - \mathbf{x} = \hat{\mathbf{x}} - \mathbf{x}_0 - \sum_{j=0}^{k} \nu_j\mathbf{p}_j = A^{-1}\mathbf{r}_0 - \sum_{j=0}^{k} \nu_j\mathbf{p}_j.$$

Таким образом, используя сопряженность направлений $\mathbf{p}_i$, приходим к оценке.

$$\|\hat{\mathbf{x}} - \mathbf{x}\|_A^2 = (\hat{\mathbf{x}} - \mathbf{x})^T A(\hat{\mathbf{x}} - \mathbf{x}) =$$
$$= \mathbf{r}_0^T A^{-1}\mathbf{r}_0 - 2\mathbf{r}_0^T\sum_{j=0}^{k} \nu_j\mathbf{p}_j + \sum_{j=0}^{k} \nu_j^2\mathbf{p}_j^T A\mathbf{p}_j.$$

Правая часть — это диагональная квадратичная [1]) форма относительно переменных ν_j, принимающая минимальное значение при

$$\nu_j = \mathbf{r}_0^{\mathrm{T}}\mathbf{p}_j / \mathbf{p}_j^{\mathrm{T}} A \mathbf{p}_j, \quad j = 0, \ldots, k.$$

Применяя повторно соотношение (П.3.1c), получаем

$$\mathbf{r}_j = \mathbf{r}_{j-1} + \alpha_{j-1} A \mathbf{p}_{j-1} = \ldots = \mathbf{r}_0 + \sum_{i=0}^{j-1} \alpha_i A \mathbf{p}_i,$$

и в силу сопряженности $\mathbf{p}_i$ имеем $\mathbf{r}_j^{\mathrm{T}}\mathbf{p}_j = \mathbf{r}_0^{\mathrm{T}}\mathbf{p}_j$. Поэтому с учетом (П.3.1a) справедливо $\nu_j = -\alpha_j$. Отсюда следует, что итерационное приближение $\mathbf{x}_{k+1}$ метода сопряженных градиентов имеет вид

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{p}_k = \ldots = \mathbf{x}_0 - \sum_{i=0}^{k} \alpha_i \mathbf{p}_i = \mathbf{x}_0 + \sum_{i=0}^{k} \nu_i \mathbf{p}_i,$$

и, тем самым, является минимизатором функционала $\|\tilde{\mathbf{x}} - \mathbf{x}\|_1^2$ на аффинном подпространстве S_k. □

Сформулируем результат теоремы П.3.3 в эквивалентной форме, которая будет полезна в дальнейшем. В силу (П.3.10)

$$S_{k-1} = \mathbf{x}_0 + \operatorname{span}(\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0),$$

так что элементы аффинного подпространства S_{k-1} можно записать в виде $\mathbf{x}_0 + \sum_{j=0}^{k-1} q_j A^j \mathbf{r}_0$. По теореме П.3.3 итерационные приближения x_k метода сопряженных градиентов удовлетворяют соотношению

$$\|\hat{\mathbf{x}} - \mathbf{x}_k\|_1^2 = \min_{q_0, \ldots, q_{k-1}} Q(q_0, \ldots, q_{k-1}),$$

где

$$Q = \left(\hat{\mathbf{x}} - \mathbf{x}_0 - \sum_{j=0}^{k-1} q_j A^j \mathbf{r}_0\right)^{\mathrm{T}} A \left(\hat{\mathbf{x}} - \mathbf{x}_0 - \sum_{j=0}^{k-1} q_j A^j \mathbf{r}_0\right).$$

Поскольку, как и выше, $A(\hat{\mathbf{x}} - \mathbf{x}_0) = \mathbf{r}_0$, имеем

$$Q = \left(\mathbf{r}_0 - \sum_{j=0}^{k-1} q_j A^{j+1}\mathbf{r}_0\right)^{\mathrm{T}} A^{-1} \left(\mathbf{r}_0 - \sum_{j=0}^{k-1} d_j A^{j+1}\mathbf{r}_0\right) =$$

$$= (R_k(A)\mathbf{r}_0)^{\mathrm{T}} A^{-1} (R_k(A)\mathbf{r}_0),$$

[1]) Подразумевается, что правая часть представляет собой квадратичную функцию с членами 2-го порядка, образующими сумму квадратов. — *Прим. перев.*

где $R_k(A) = I - \sum_{j=0}^{k-1} q_j A^{j+1}$ — матричный полином, соответствующий скалярному полиному $1 - \sum_{j=0}^{k-1} q_j \lambda^{j+1}$. Таким образом, мы можем сформулировать результат П.3.3 в эквивалентной форме:

$$\| \hat{\mathbf{x}} - \mathbf{x}_k \|_1^2 = \min_{R_k \in \mathcal{R}_k} \mathbf{r}_0^{\mathrm{T}} R_k(A)^{\mathrm{T}} A^{-1} R_k(A)\, \mathbf{r}_0, \qquad (\text{П.3.11})$$

где $\mathcal{R}_k$ — множество многочленов степени k, удовлетворяющих условию $R_k(0) = 1$.

Представление (П.3.11) позволит нам получить оценки погрешности на k-м шаге. Сначала установим некоторые свойства полиномов R_k.

Пусть $\lambda_1, \ldots, \lambda_n$ и $\mathbf{v}_1, \ldots, \mathbf{v}_n$ — собственные значения и соответствующие им ортонормированные собственные векторы матрицы A, и пусть $\mathbf{r}_0 = \sum_{j=1}^{n} \eta_j \mathbf{v}_j$. Тогда

$$R_k(A)\, \mathbf{r}_0 = \sum_{j=1}^{n} \eta_j R_k(A) \mathbf{v}_j = \sum_{j=1}^{n} \eta_j R_k(\lambda_j) \mathbf{v}_j.$$

Таким образом,

$$\mathbf{r}_0^{\mathrm{T}} R_k(A)^{\mathrm{T}} A^{-1} R_k(A)\, \mathbf{r}_0 = \left(\sum_{j=1}^{n} \eta_j R_k(\lambda_j) \mathbf{v}_j \right)^{\mathrm{T}} \times$$

$$\times \left(\sum_{j=1}^{n} \eta_j R_k(\lambda_j) \lambda_j^{-1} \mathbf{v}_j \right) = \sum_{j=1}^{n} \eta_j^2 (R_k(\lambda_j))^2 \lambda_j^{-1}$$

в силу ортонормированности векторов $\mathbf{v}_j$. Подставляя полученное представление в (П.3.11), где минимум берется по всем полиномам из $\mathcal{R}_k$, получаем

$$\| \hat{\mathbf{x}} - \mathbf{x}_k \|_A^2 = \min \sum_{j=1}^{n} \eta_j^2 (R_k(\lambda_j))^2 \lambda_j^{-1} \leqslant \min \max_{1 \leqslant j \leqslant n} | R_k(\lambda_j) |^2 \sum_{j=1}^{n} \eta_j^2 \lambda_j^{-1}. \qquad (\text{П.3.12})$$

Далее,

$$\sum_{j=1}^{n} \eta_j^2 \lambda_j^{-1} = \left(\sum_{j=1}^{n} \eta_j \mathbf{v}_j \right)^{\mathrm{T}} A^{-1} \left(\sum_{j=1}^{n} \eta_j \mathbf{v}_j \right) =$$

$$= \mathbf{r}_0^{\mathrm{T}} A^{-1} \mathbf{r}_0 = (\hat{\mathbf{x}} - \mathbf{x}_0)^{\mathrm{T}} A (\hat{\mathbf{x}} - \mathbf{x}_0). \qquad (\text{П.3.13})$$

Сопоставляя (П.3.12) и (П.3.13), приходим к оценке

$$\| \hat{\mathbf{x}} - \mathbf{x}_k \|_A^2 \leqslant \gamma_k^2 \| \hat{\mathbf{x}} - \mathbf{x}_0 \|_A^2, \qquad (\text{П.3.14})$$

где

$$\gamma_k = \min_{R_k \in \mathcal{R}_k} \max_{1 \leqslant j \leqslant n} | R_k(\lambda_j) |. \qquad (\text{П.3.15})$$

Таким образом, мы оценили погрешность на k-м шаге через начальную погрешность. Теперь задача заключается в том, чтобы получить оценку для γ_k.

Ясно, что если λ_1 и λ_n — наименьшее и наибольшее собственные значения матрицы A, то

$$\gamma_k \leqslant \min_{R_k \in \mathcal{R}_k} \max_{\lambda_1 \leqslant \lambda \leqslant \lambda_n} |R_k(\lambda)|. \qquad (\text{П.3.16})$$

Таким образом, возникает классическая задача чебышёвской минимизации, решение которой имеет вид

$$\widehat{R}_k(\lambda) = T_k\left(\frac{\lambda_n + \lambda_1 - 2\lambda}{\lambda_n - \lambda_1}\right) \Big/ T_k\left(\frac{\lambda_n + \lambda_1}{\lambda_n - \lambda_1}\right),$$

где $T_k(z)$ — полином Чебышёва степени k (см. (3.2.54)). Используя теорию многочленов Чебышёва, можно показать, что

$$\min_{R_k \in \mathcal{R}_k} \max_{\lambda_1 \leqslant \lambda \leqslant \lambda_n} |R_k(\lambda)| \leqslant 2\left(1 - \sqrt{\nu}\right)^k / \left(1 + \sqrt{\nu}\right)^k, \quad \nu = \lambda_1/\lambda_n. \qquad (\text{П.3.17})$$

Поскольку A — симметричная положительно определенная матрица, ее число обусловленности в норме l_2 равно $\varkappa = 1/\nu$. Комбинируя (П.3.14), (П.3.16) и (П.3.17), получаем оценку погрешности [1])

$$\|\widehat{\mathbf{x}} - \mathbf{x}_k\|_A \leqslant 2\alpha^k \|\widehat{\mathbf{x}} - \mathbf{x}_0\|_A, \quad \alpha = \left(\sqrt{\varkappa} - 1\right)/\left(\sqrt{\varkappa} + 1\right). \qquad (\text{П.3.18})$$

Это стандартная оценка погрешности метода сопряженных градиентов в A-норме.

Мы можем также получить оценки типа (П.3.18) в 2-норме при помощи неравенств

$$\|\mathbf{x}\|_2^2 / \|A^{-1}\|_2 = \lambda_1 \mathbf{x}^{\mathrm{T}}\mathbf{x} \leqslant \mathbf{x}^{\mathrm{T}} A \mathbf{x} \leqslant \lambda_n \mathbf{x}^{\mathrm{T}}\mathbf{x} = \|A\|_2 \|\mathbf{x}\|_2^2.$$

Применяя эти неравенства к (П.3.18), получаем

$$\|\widehat{\mathbf{x}} - \mathbf{x}_k\|_2^2 \leqslant \|A^{-1}\|_2 \|\widehat{\mathbf{x}} - \mathbf{x}_k\|_A^2 \leqslant$$
$$\leqslant (2\alpha^k)^2 \|A^{-1}\|_2 \|\widehat{\mathbf{x}} - \mathbf{x}_0\|_A^2 \leqslant (2\alpha^k)^2 \|A^{-1}\|_2 \|A\|_2 \|\widehat{\mathbf{x}} - \mathbf{x}_0\|_2^2,$$

или

$$\|\widehat{\mathbf{x}} - \mathbf{x}_k\|_2 \leqslant 2\sqrt{\varkappa}\,\alpha^k \|\widehat{\mathbf{x}} - \mathbf{x}_0\|_2. \qquad (\text{П.3.19})$$

---

[1]) Точная оценка имеет вид $\|\widehat{\mathbf{x}} - \mathbf{x}_k\|_A \leqslant \dfrac{2\alpha^k}{1 + \alpha^{2k}} \|\widehat{\mathbf{x}} - \mathbf{x}_0\|_A$ и является в определенном смысле неулучшаемой (см., например, [Greenbaum A. Comparison of splittings used with the conjugate gradient algorithm. — Numer. Math., 1979, v. 33, N 2, p. 181—193]). — *Прим. перев.*

Суммируем приведенные оценки погрешности в следующем утверждении.

П.3.4. Теорема. *Приближение* $\mathbf{x}_k$, *полученное методом сопряженных градиентов, удовлетворяет оценкам*

$$\|\hat{\mathbf{x}} - \mathbf{x}_k\|_1 \leqslant 2\alpha^k \|\hat{\mathbf{x}} - \mathbf{x}_0\|_1,$$
$$\|\hat{\mathbf{x}} - \mathbf{x}_k\|_2 \leqslant 2\sqrt{\varkappa}\,\alpha^k \|\hat{\mathbf{x}} - \mathbf{x}_0\|_2,$$

где $\varkappa = \operatorname{cond}(A)$ *и* $\alpha = (\sqrt{\varkappa} - 1)/(\sqrt{\varkappa} + 1)$.

Из оценки погрешности (П.3.19) еще не следует, что погрешность действительно убывает на каждом шаге. Это важное свойство устанавливает следующая теорема.

П.3.5. Теорема. *Приближения, полученные методом сопряженных градиентов, удовлетворяют оценке*

$$\|\hat{\mathbf{x}} - \mathbf{x}_k\|_2 < \|\hat{\mathbf{x}} - \mathbf{x}_{k-1}\|_2, \tag{П.3.20}$$

если только не выполнено $\mathbf{x}_{k-1} = \hat{\mathbf{x}}$.

Доказательство. Заметим сначала, что

$$\|\hat{\mathbf{x}} - \mathbf{x}_{k-1}\|_2^2 = (\hat{\mathbf{x}} - \mathbf{x}_k + \mathbf{x}_k - \mathbf{x}_{k-1})^{\mathrm{T}} (\hat{\mathbf{x}} - \mathbf{x}_k + \mathbf{x}_k - \mathbf{x}_{k-1}) =$$
$$= (\hat{\mathbf{x}} - \mathbf{x}_k,\ \hat{\mathbf{x}} - \mathbf{x}_k) + 2(\hat{\mathbf{x}} - \mathbf{x}_k,\ \mathbf{x}_k - \mathbf{x}_{k-1}) + (\mathbf{x}_k - \mathbf{x}_{k-1},\ \mathbf{x}_k - \mathbf{x}_{k-1}),$$

или

$$\|\hat{\mathbf{x}} - \mathbf{x}_{k-1}\|_2^2 = \|\hat{\mathbf{x}} - \mathbf{x}_k\|_2^2 + 2(\hat{\mathbf{x}} - \mathbf{x}_k,\ \mathbf{x}_k - \mathbf{x}_{k-1}) + \|\mathbf{x}_k - \mathbf{x}_{k-1}\|_2^2. \tag{П.3.21}$$

Последняя величина в правой части положительна, за исключением случая, когда $\mathbf{x}_k = \mathbf{x}_{k-1}$. Как уже упоминалось выше, из $\mathbf{x}_k = \mathbf{x}_{k-1}$ следует, что $\mathbf{x}_{k-1} = \hat{\mathbf{x}}$. Таким образом, если $\mathbf{x}_k \neq \mathbf{x}_{k-1}$, достаточно показать, что второе слагаемое в правой части (П.3.21) неотрицательно.

Пусть $\mathbf{x}_m = \hat{\mathbf{x}}$. Тогда в силу

$$\mathbf{x}_m - \mathbf{x}_k = \mathbf{x}_m - \mathbf{x}_{m-1} + \mathbf{x}_{m-1} - \ldots - \mathbf{x}_{k+1} + \mathbf{x}_{k+1} - \mathbf{x}_k$$

с использованием (П.3.1b) получаем

$$(\hat{\mathbf{x}} - \mathbf{x}_k)^{\mathrm{T}} (\mathbf{x}_k - \mathbf{x}_{k-1}) = (\alpha_{m-1}\mathbf{p}_m^{\mathrm{T}}\mathbf{p}_{k-1} + \ldots + \alpha_k \mathbf{p}_k^{\mathrm{T}}\mathbf{p}_{k-1})\,\alpha_{k-1}. \tag{П.3.22}$$

В силу (П.3.6) все величины α_i неположительны; таким образом, достаточно показать, что $p_j^{\mathrm{T}} p_{k-1} \geqslant 0$ при $j \geqslant k$. Применяя

повторно соотношение (П.3.1), получаем

$$\mathbf{p}_j = \mathbf{r}_j + \beta_{j-1}\mathbf{r}_{j-1} + \ldots + (\beta_{j-1}\ldots\beta_k)\,\mathbf{r}_k + (\beta_{j-1}\ldots\beta_{k-1})\,\mathbf{p}_{k-1}. \tag{П.3.23}$$

В частности, для $k = 1$ соотношение (П.3.23) показывает, что $\mathbf{p}_j \in \operatorname{span}(\mathbf{r}_0, \ldots, \mathbf{r}_j)$, и в силу ортогональности векторов $\mathbf{r}_i$ (см. (П.3.3)) получаем, что $\mathbf{r}_j^{\mathrm{T}}\mathbf{p}_{k-1} = 0$, $j \geqslant k$. Таким образом, из (П.3.23) и (П.3.7) следует

$$\mathbf{p}_j^{\mathrm{T}}\mathbf{p}_{k-1} = \beta_{j-1}\ldots\beta_{k-1}\mathbf{p}_{k-1}^{\mathrm{T}}\mathbf{p}_{k-1} = \mathbf{r}_j^{\mathrm{T}}\mathbf{r}_j\mathbf{p}_{k-1}^{\mathrm{T}}\mathbf{p}_{k-1}/\mathbf{r}_{k-1}^{\mathrm{T}}\mathbf{r}_{k-1} \geqslant 0. \quad \square$$

Приведенные результаты получены без учета каких-либо специальных свойств матрицы A. В частности, представляет интерес следующий результат.

П.3.6. Теорема. *Если матрица A имеет только m различных собственных значений, то метод сопряженных градиентов сходится не более чем за m итераций.*

Доказательство. Это утверждение является почти прямым следствием основной оценки погрешности (П.3.11). Пусть $\lambda_1, \ldots, \lambda_m$ — различные собственные значения матрицы A (положительные в силу положительной определенности A); определим полином степени m формулой

$$R_m(\lambda) = \prod_{i=1}^{m}(\lambda_i - \lambda)/\prod_{i=1}^{m}\lambda_i.$$

Ясно, что $R_m(0) = 1$. Кроме того, $R_m(A) = 0$, поскольку $A = PDP^{\mathrm{T}}$, где D — диагональная матрица, а P — ортогональная; но тогда $R_m(A) = PR_m(D)P^{\mathrm{T}}$, и матрица $R_m(D)$ является произведением диагональных матриц, таких, что для любого i по крайней мере одна из этих матриц имеет нулевой элемент в i-й диагональной позиции. Тогда соотношение (П.3.11) показывает, что $\hat{\mathbf{x}} - \mathbf{x}_m = \mathbf{0}$.

Трехчленное рекуррентное соотношение

В заключение покажем, что приближения метода сопряженных градиентов удовлетворяют трехчленному рекуррентному соотношению вида

$$\mathbf{x}_{k+1} = \rho_{k+1}(\delta_{k+1}\mathbf{r}_k + \mathbf{x}_k) + (1 - \rho_{k+1})\,\mathbf{x}_{k-1}, \tag{П.3.24}$$

где при α_k и β_{k-1}, определенных в (П.3.1),

$$\rho_{k+1} = 1 - \alpha_k\beta_{k-1}/\alpha_{k-1}, \quad \delta_{k+1} = -\alpha_k/\rho_{k+1}. \tag{П.3.25}$$

Чтобы получить (П.3.24), заметим сначала, что из (П.3.1b) следует

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{p}_k = \mathbf{x}_k - \alpha_k (\mathbf{r}_k + \beta_{k-1} \mathbf{p}_{k-1}). \qquad (\text{П.3.26})$$

Подставим теперь соотношение $\mathbf{p}_{k-1} = \alpha_{k-1}^{-1} (\mathbf{x}_k - \mathbf{x}_{k-1})$, полученное из (П.3.1b), в (П.3.26).

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{r}_k - \alpha_k \beta_{k-1} (\mathbf{x}_k - \mathbf{x}_{k-1}) / \alpha_{k-1} =$$
$$= \left(1 - \frac{\alpha_k \beta_{k-1}}{\alpha_{k-1}}\right) \mathbf{x}_k - \alpha_k \mathbf{r}_k + \frac{\alpha_k \beta_{k-1}}{\alpha_{k-1}} \mathbf{x}_{k-1}, \qquad (\text{П.3.27})$$

что совпадает с (П.3.24).

Теперь заменим $\mathbf{r}_k$ в (П.3.24) на $\mathbf{b} - A\mathbf{x}_k$:

$$\mathbf{x}_{k+1} = \rho_{k+1} (\delta_{k+1} (\mathbf{b} - A\mathbf{x}_k) + \mathbf{x}_k) + (1 - \rho_{k+1}) \mathbf{x}_{k-1} =$$
$$= \rho_{k+1} (\delta_{k+1} ((I - A) \mathbf{x}_k + \mathbf{b}) + (1 + \delta_{k+1}) \mathbf{x}_k) + (1 - \rho_{k+1}) \mathbf{x}_{k-1}. \qquad (\text{П.3.28})$$

Если матрица A масштабирована так, что все ее диагональные элементы равны единице, то итерации метода Якоби имеют вид

$$\mathbf{y}_{k+1} = (I - A) \mathbf{y}_k + \mathbf{b}.$$

Таким образом, (П.3.28) представляет собой ускорение метода Якоби, аналогичное полуитерационному методу (3.2.61), — за исключением того, что параметр δ_{k+1} теперь может изменяться на каждой итерации. Можно показать, что если использовать ускорение по методу сопряженных градиентов для метода Якоби, то получится одношаговый предобусловленный метод Якоби — сопряженных градиентов. Аналогично, если мы применим такое ускорение к итерациям симметричного метода последовательной верхней релаксации (SSOR), то получим одношаговый предобусловленный метод SSOR — сопряженных градиентов.

Приложение 4

Основные сведения из линейной алгебры

В этом приложении мы приведем (без доказательств) некоторые фундаментальные результаты из линейной алгебры, используемые в тексте. Полное изложение можно найти, например, в книге [Ortega, 1987a]. Мы рассматриваем только случай вещественных векторов и матриц. Предполагается, что читатель знаком с элементарными разделами линейной алгебры.

Если A — матрица размерности $n \times n$, то следующие утверждения эквивалентны:

$\det A \neq 0$;

существует такая матрица A^{-1} размерности $n \times n$, что $AA^{-1} = A^{-1}A = I$;

$A\mathbf{x} = \mathbf{b}$ имеет единственное решение для любого $\mathbf{b}$;

$A\mathbf{x} = 0$ имеет только решение $\mathbf{x} = 0$;

строки или столбцы A линейно независимы.

Если A удовлетворяет любому из этих условий (и, таким образом, всем условиям), она является невырожденной и A^{-1} — ее обратная матрица.

Ранг матрицы равен количеству линейно независимых строк (или, что то же самое, столбцов). Следовательно, последнее из перечисленных утверждений можно сформулировать как «A имеет ранг n». Важный класс матриц образуют матрицы ранга единица; каждую такую матрицу можно записать в виде $\mathbf{uv}^T$, где $\mathbf{u}$, $\mathbf{v}$ — векторы-столбцы.

Матрица P называется *ортогональной*, если $PP^T = I$. Столбцы $\mathbf{p}_i$ ортогональной матрицы образуют ортогональное множество векторов ($\mathbf{p}_i^T\mathbf{p}_j = 0$, $i \neq j$), являющееся также ортонормированным ($\mathbf{p}_i^T\mathbf{p}_i = 1$). Матрицы, обратные к ортогональным матрицам, как и их произведения, также будут ортогональными. Важный тип ортогональных матриц — это *матрицы перестановки*, каждая строка и столбец которых содержит в точности один элемент, равный единице, а остальные элементы равны нулю. В результате умножения вида PA, где P — матрица переста-

новки, происходит переупорядочение строк матрицы A. Умножение вида AP приводит к переупорядочению столбцов.

Собственное значение матрицы A — это скаляр, для которого уравнение

$$A\mathbf{x} = \lambda\mathbf{x}$$

имеет ненулевое решение $\mathbf{x}$, называемое *собственным вектором*. Матрица размера $n \times n$ имеет в точности n собственных значений (с учетом их кратностей). Они являются корнями *характеристического уравнения* $\det(A - \lambda I) = 0$, представляющего собой полиномиальное уравнение степени n. Собственное значение может быть комплексным, даже если матрица A вещественная, и соответствующий собственный вектор тогда обязательно будет комплексным. Множество собственных значений матрицы называют *спектром* A, а максимальный из модулей собственных значений матрицы A называют *спектральным радиусом* A и обозначают $\rho(A)$.

Если $p(\gamma) = a_0 + a_1\gamma + \ldots + a_m\gamma^m$ — полином степени m, то матрица $p(A) = a_0 I + a_1 A + \ldots + a_m A^m$ является соответствующим *матричным полиномом*. Если λ и $\mathbf{x}$ — собственное значение и собственный вектор матрицы A, то $p(\lambda)$ и $\mathbf{x}$ — собственное значение и собственный вектор матрицы $p(A)$. В частности, если A имеет собственные значения $\lambda_1, \ldots, \lambda_n$, то $\alpha I + A$ имеет собственные значения $\alpha + \lambda_1, \ldots, \alpha + \lambda_n$ и те же собственные векторы, что и A. Кроме того, матрица A невырожденна в том и только в том случае, если $\lambda_i \neq 0$, $i = 1, \ldots, n$, и в этом случае A^{-1} имеет собственные значения λ_i^{-1} и те же собственные векторы, что и A.

Матрицу A называют *симметричной*, если она совпадает со своей транспонированной матрицей: $A = A^{\mathrm{T}}$. Симметричная матрица имеет только вещественные собственные значения, а соответствующие собственные векторы $\mathbf{x}_i$ попарно ортогональны. Симметричная матрица может быть записана в виде

$$A = PDP^{\mathrm{T}}, \qquad (\text{П.4.1})$$

где $D = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ — диагональная матрица, диагональные элементы которой равны собственным значениям A, а P — ортогональная матрица, столбцы которой являются собственными векторами A.

Главной подматрицей A называют подматрицу, полученную удалением строк и столбцов матрицы A с одними и теми же номерами. *Ведущая главная подматрица* $m \times m$ получается при удалении всех строк и столбцов с номерами $m + 1, \ldots, n$.

Симметричная матрица является *положительно определенной*, если выполняется одно из следующих эквивалентных условий:

$\mathbf{x}^T A\mathbf{x} > 0$ для всех векторов $\mathbf{x} \neq 0$;

все собственные значения A положительны;

все ведущие главные подматрицы A имеют положительный определитель;

существует факторизация Холесского $A = LL^T$.

Симметричная положительно определенная матрица A всегда невырожденна, и матрица A^{-1} также является симметричной и положительно определенной. Для любой симметричной положительно определенной матрицы существует квадратный корень, который можно ввести (с использованием (П.4.1)) посредством соотношения $A^{1/2} = PD^{1/2}P^T$, где $D^{1/2}$ — диагональная матрица, диагональные элементы которой равны положительным квадратным корням из собственных значений матрицы A.

Симметричную матрицу называют *положительно полуопределенной* (или неотрицательно определенной), если все ее собственные значения неотрицательны или, что эквивалентно, $\mathbf{x}^T A\mathbf{x} \geqslant 0$ для всех x. В общем случае, если A — симметричная матрица с собственными значениями $\lambda_1 \leqslant \lambda_2 \leqslant \ldots \leqslant \lambda_n$, то

$$\lambda_1 \mathbf{x}^T\mathbf{x} \leqslant \mathbf{x}^T A\mathbf{x} \leqslant \lambda_n \mathbf{x}^T\mathbf{x}$$

для всех векторов $\mathbf{x}$.

Если P — невырожденная матрица, то PAP^{-1} называют *преобразованием подобия*, а PAP^T — *преобразованием конгруэнтности*. Преобразование подобия сохраняет собственные значения, т. е. матрицы A и PAP^{-1} имеют одни и те же собственные значения. Преобразование конгруэнтности в общем случае изменяет собственные значения, однако сохраняет их знаки: если A — симметричная матрица, то матрицы A и PAP^T имеют одинаковые количества как положительных, так и отрицательных собственных значений. В частности, если A положительно определена, то матрица PAP^T также положительно определена.

Соотношение (П.4.1) показывает, что симметричная матрица подобна диагональной матрице. В общем случае матрица A подобна диагональной тогда и только тогда, когда она имеет n линейно независимых собственных векторов. В противном случае лучшее, чего мы можем добиться, — это подобие матрицы A матрице с диагональными элементами, равными собственным значениям матрицы A, некоторыми наддиагональными элементами, равными единице, и остальными элементами, равными нулю. Указанное представление называют *жордановой канонической формой* матрицы A.

Норма $\|\cdot\|$ на пространстве R^n векторов длины n удовлетворяет аксиомам

$$\begin{aligned} &\|\mathbf{x}\| \geqslant 0 \text{ и } \|\mathbf{x}\| = 0 \text{ только для } \mathbf{x} = 0; \\ &\|\alpha\mathbf{x}\| = |\alpha| \|\mathbf{x}\| \text{ для всех скаляров } \alpha; \\ &\|\mathbf{x} + \mathbf{y}\| \leqslant \|\mathbf{x}\| + \|\mathbf{y}\|. \end{aligned} \tag{П.4.2}$$

Часто используются норма l_∞ (или *норма «максимум модуля»*)

$$\|\mathbf{x}\|_\infty = \max_{1 \leqslant i \leqslant n} |x_i|$$

и нормы l_p, $0 < p < \infty$,

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p},$$

особенно для $p = 1$ и $p = 2$. Норма l_2 представляет собой обычное евклидово расстояние [1]). В этой норме для ортогональных матриц справедливо важное свойство $\|P\mathbf{x}\|_2 = \|\mathbf{x}\|_2$. Другой класс норм определяется соотношением

$$\|\mathbf{x}\|_B = (\mathbf{x}^T B \mathbf{x})^{1/2},$$

где B — симметричная положительно определенная матрица. Это *нормы типа скалярного произведения*, так как выражение $(\mathbf{x}, \mathbf{y})_B \equiv \mathbf{x}^T B \mathbf{y}$ определяет скалярное произведение [2]).

Нормы матриц можно определить при помощи векторных норм:

$$\|A\| = \max_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\|\mathbf{x}\| = 1} \|A\mathbf{x}\|.$$

Такие матричные нормы удовлетворяют всем свойствам векторных норм, а также свойству

$$\|AB\| \leqslant \|A\| \|B\|.$$

Если $\|A\| < 1$ для некоторой нормы $\|\cdot\|$ (или, что эквивалентно, $\rho(A) < 1$), то справедливо следующее выражение для суммы геометрической прогрессии, называемое также *разложением Неймана:*

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k.$$

---

[1]) Точнее, расстояние от 0 до $\mathbf{x}$. — *Прим. перев.*

[2]) С точки зрения обобщения на комплексный случай корректнее записывать $(\mathbf{x}, \mathbf{y})_B = \mathbf{y}^H B \mathbf{x}$, так как именно выражение $\mathbf{y}^H B \mathbf{x}$ определяет при $B = B^H$ скалярное произведение на C^n; здесь H — символ транспонирования с сопряжением. — *Прим. перев.*

Литература

Обширная библиография по численным методам для параллельных и векторных компьютеров составляет содержание отчета [Ortega, Voigt, 1987]. Мы намерены поддерживать и обновлять эту библиографию, а также сделать ее доступной абонентам компьютерных сетей. Информацию по этому поводу можно получить по адресу ICASE, MS132C, NASA-Langley Research Center, Hampton, Virginia 23665, 804-865-2513.

В приводимый ниже список включены антологии и сборники трудов некоторых конференций, опубликованные в книжной форме, при этом сборники нужно искать по фамилии их редактора(ов). Отдельные статьи таких сборников снабжаются указателями на сборник в целом, так, ссылка Brandt A [1981] «Multigrid Solvers on Parallel Computers» in Schultz [1981], p. 39–83 относится к статье Брандта, находящейся в сборнике, который нужно искать в рубрике Schultz [1981].

Abu-Shomays I. [1985]. «Comparison of Methods and Algorithms for Vectorization of Diffusion Computation», in Numrich [1985], p. 29—56.

Abu-Sufah W., Malony A. [1986] «Vector Processing on the Alliant FX/8 Multiprocessor», Proc. 1986 Int. Conf. Parallel Processing, p. 559—566.

Adams L. [1982] «Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers», Ph. D. dissertation, Applied Mathematics, University of Virginia; also published as NASA CR-166027, NASA Langley Research Center.

Adams L. [1983]. «An M-Step Preconditioned Conjugate Gradient Method for Parallel Computation», Proc. 1983 Int. Conf. Parallel Processing, p. 36–43.

Adams L. [1985]. «An M-Step Preconditioned Conjugate Gradient Methods», SIAM J. Sci. Stat. Comput. 6, 452–463.

Adams L. [1986]. «Reordering Computations for Parallel Execution», Commun. Appl. Numer. Math. 2, 263–271.

Adams L., Jordan H. [1985]. «Is SOR Color-Blind?» SIAM J. Sci. Stat. Comput. 7, 490—506.

Adams L., LeVeque R., Young D. [1987] «Analysis of the SOR Iteration for the 9-Point Laplacian», SIAM J. Numer. Anal. To appear.

Adams L., Ong E. [1987] «Additive Polynomial Preconditioners for Parallel Computers», Parallel Comput., to appear.

Adams L., Ortega J. [1982] «A Multi-Color SOR Method for Parallel Computation», Proc. 1982 Int. Conf. Parallel Processing, p. 53–56.

Adams L., Voigt R. [1984]. «A Methodology for Exploiting Parallelism In the Finite Element Process», in Kowalik [1984], p. 373–392.

Alaghband G., Jordan H. [1985] «Multiprocessor Sparse L/U Decomposition with Controlled Fill-In», ICASE Report No. 85–48, NASA Langley Research Center.

Amdahl G. [1967]. «The Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities», AFIPS Conf. Proc. 30, p. 483—485.

Andrews G., Schneider F [1983] «Concepts and Notations for Concurrent Programming», Comput Surveys 15, 3--43
Ashcraft C [1985a] «Parallel Reduction Methods for the Solution of Banded Systems of Equations», General Motors Research Lab Report No GMR-5094
Ashcraft S. [1985b] «A Moving Computation Front Approach for Vectorizing ICCG Calculations», General Motors Research Lab Report No GMR-5174.
Ashcraft C. [1987a] «A Vector Implementation of the Multifrontal Method for Large Sparse, Symmetric Positive Definite Linear Systems», Applied Mathematics Technical Report ETA-TR-51, Boeing Computer Services
Ashcraft C [1987b] «Domain Decoupled Incomplete Factorizations»
Applied Mathematics Technical Report ETA-TR-47, Boeing Computer Services
Ashcraft C, Grimes R [1987] «On Vectorizing Incomplete Factorization and SSOR Preconditioners», Applied Mathematics Technical Report ETA-TR-41, Boeing Computer Services, SIAM J Sci Stat Comput. To appear
Axelrod T [1986] «Effects of Synchronization Barriers on Multiprocessor Performance», Parallel Comput 3, 129—140
Axelsson O [1876]. «A Class of Iterative Methods for Finite Element Equations», Comput Methods Appl Mech Eng 9, 123—137
Axelsson O [1985] «A Survey of Vectorizable Preconditioning Methods for Large Scale Finite Element Matrix Problems», BIT 25, 166—187
Axelsson O [1986] «A General Incomplete Block-Matrix Factorization Method», Lin. Alg Appl 74, 179—190
Axelsson O, Lindskog G [1986] «On the Rate of Convergence of the Preconditioned Conjugate Gradient Method», Numer Math 48, 499—523
Axelsson O, Polman B [1986] «On Approximate Factorization Methods for Block Matrices Suitable for Vector and Parallel Processors», Lin Alg. Appl. 77, 3--26

Bank R, Douglas C [1985] «An Efficient Implementation for SSOR and Incomplete Factorization Preconditionings», Appl Numer Math 1, 489—492
Barkai D, Brandt A. [1983] «Vectorized Multigrid Poisson Solver for the CDC Cyber 205», Appl Math Comp 13, 215 --228
Barlow J, Ipsen I [1984] «Parallel Scaled Givens Rotations for the Solution of Linear Least Squares Problems», Department of Computer Science Report RR-310, Yale University
Barlow R, Evans D. [1982] «Synchronous and Asynchronous Iterative Parallel Algorithms for Linear Systems», Comput. J 25, 56—60
Baudet G [1978] «Asynchronous Iterative Methods for Multiprocessors», J ACM 25, 226—244
Berger M., Bokhari S [1985] «A Partitioning Strategy for PDE's Across Multiprocessors», Proc 1985 Int. Conf Parallel Processing, p 166—170
Berger M, Bokhari S [1987]. «A Partitioning Strategy for Non-Uniform Problems on Multiprocessors», IEEE Trans Comput TC36, 570--580.
Berger P., Brouaye P, Syre J [1982] «A Mesh Coloring Method for Efficient MIMD Processing in Finite Element Problems», Proc. 1982 Int Conf Parallel Processing, p 41 - 46
Bini D [1984] «Parallel Solution of Certain Toeplitz Linear Systems», SIAM J. Comput 13, 368—476.
Birkhoff G, Schoenstadt A (eds) [1984] Elliptic Problem Solvers, Academic Press, New York
Bischof C, Van Loan C [1987] «The WY Representation for Products of Householder Matrices», SIAM J Sci Stat. Comput 8, s2—s13.
Bjorstad P [1987] «A Large Scale, Sparse, Secondary Storage, Direct Linear Equation Solver for Structural Analysis and its Implementation on Vector and Parallel Architectures», Parallel Comput. 5, 3—12.

Bianczyk A, Brent R., Kung H [1984] «Numerically Stable Solution of Dense Systems of Linear Equations using Mesh-Connected Processors», SIAM Sci Somput 5, 95—104
Bokhari S. [1981] «On the Mapping Problem», IEEE Trans Comput C-30, 207—214
Bokhari S [1985] «Partitioning Problems in Parallel, Pipelined and Distributed Computing», ICASE Report No 85-54, NASA Langley Research Center. To appear in IEEE Trans Comput
Book D [1981] Finite Difference Techniques for Vectorized Fluid Dynamics Calculation, Springer-Verlag, New York
Bowgen G., Modi, J. [1985]. «Implementation of QR Factorization on the DAP Using Householder Transformations», Comput. Phys Commun 37, 167—170.
Brandt A. [1977]. «Multigrid Adaptive Solutions to Boundary Value Problems», Math Comp. 31, 333—390
Brandt A [1981]. «Multigrid Solvers on Parallel Computers», in Schultz [1981], p. 39—83
Bucher I. [1983] «The Computational Speed of Supercomputers», Proc. ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems, p. 151—165.
Buzbee B. [1973] «A Fast Poisson Solver Amenable to Parallel Computation», IEEE Trans Comput C-22, 793—796
Buzbee B [1983]. «Remarks for the IFIP Congress '83 Panel on How to Obtain High Performance for High-Speed Processors», Los Alamos National Laboratory Report No LA-UR-83-1392
Buzbee B [1985]. «Two Parallel Formulations of Particle-in-Cell Models», in Snyder et al [1985], p 223—232

Calahan D [1985]. «Task Granularity Studies on a Many-Processor CRAY X-MP», Parallel Comput. 2, 109—118
Calahan D. [1986] «Block-Oriented, Local-Memory-Based Linear Equation Solution on the CRAY-2: Uniprocessor Algorithms», Proc 1986 Int. Conf. Parallel Processing, p. 375—378
Chamberlain R. [1987]. «An Alternative View of LU Factorization with Partial Pivoting on a Hypercube Multiprocessor», in Heath [1987], p. 569—575
Chan T. [1985] «Analysis of Preconditioners for Domain Decomposition», Department of Computer Science Report RR-408, Yale University
Chan T [1987] «On the Implementation of Kernel Numerical Algorithms for Computational Fluid Dynamics of Hypercubes», in Heath [1987], p 747—755.
Chan T., Resasco D. [1987a]. «A Domain-Decomposed Fast Poisson Solver on a Rectangle», SIAM J. Sci. Stat Comput. 8, s14—s26
Chan T., Resasco D. [1987b], «Hypercube Implementation of Domain Decomposed Fast Poisson Solvers», in Heath [1987], p. 738—746
Chan T., Saad Y., Schultz M [1987]. «Solving Elliptic Partial Differential Equations on the Hypercube Multiprocessor, Appl. Numer Methods 3, 81—88.
Chan T., Tuminaro R [1987] «Implementation of Multigrid Algorithms on Hypercubes», in Heath [1987], p 730—737
Chazan D., Miranker W [1969]. «Chaotic Relaxation», Lin. Alg Appl. 2, 199—222
Chen S [1982]. «Polynomial Scaling in the Conjugate Gradient Method and Related Topics in Matrix Scaling», Ph D dissertation, Departmen of Computer Science, Pennsylvania State University.
Chen S. [1984]. «Large-Scale and High-Speed Multiprocessor System for Scientific Applications: CRAY X-MP-2 Series»,in Kowalik [1984], p. 59—67.

Cheng K., Sahni S [1987] «VLSI Systems for Band Matrix Multiplication», Parallel Comput 4, 239—258
Chu E, George A. [1987]. «Gaussian Elimination with Partial Pivoting and Load Balancing on a Multiprocessor», Parallel Comput 5, 65—74
Cleary A., Harrar D, Ortega J [1986]. «Gaussian Elimination and Choleski Factorization on the FLEX/32», Applied Mathematics Report RM-86-13, University of Virginia.
Clementi E, Detrich J., Chin S., Corongiu G, Folsom D., Logan D., Caltabiano R, Carnevali A, Helin J., Russo M., Gnudi A, Palamidese P. [1987]. «Large-Scale Computations on a Scalar, Vector and Parallel 'Sypercomputer'», Parallel Comput. 5, 13—44.
Concus P, Golub G., Meurant G. [1985] «Block Preconditioning for the Conjugate Gradient Method», SIAM J. Sci Stat. Comput 6, 220—252.
Concus P, Golub G., O'Leary D [1976] «A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations», in Sparse Matrix Computations, J Bunch and D Rose (eds), Academic Press, New York, p 309—322
Conrad V, Wallach Y. [1977]. «A Faster SSOR Algorithm», Numer Math. 27, 371—372
Conrad V., Wallach Y. (1979]. «Alternating Methods for Sets of Linear Equations», Numer Math 32, 105—108
Cosnard M, Robert Y. [1986]. «Complexity of Parallel QR Factorization», J. ACM 33, 712—723
Cowell W, Thompson C. [1986]. «Transforming Fortran DO Loops to Improve Performance on Vector Architectures», ACM Trans Math. Software 12, 324—353.

Dave A, Duff I. [1987] «Sparse Matrix Calculations on the CRAY-2», Parallel Comput. 5, 55—64
Davis G. [1986]. «Column LU Factorization with Pivoting on a Hypercube Multiprocessor», SIAM J. Algebraic Discrete Methods 7, 538—550
Delosme J.-M, Ipsen I [1987] «Efficient Systolic Arrays for the solution of Toeplitz Systems: An Illustration of a Methodology for the Construction of Systolic Architectures in VLSI», in Systolic Arrays, Adam Hilger Ltd., Bristol, p 37—46
Deminet J. [1982]. «Experience with Multiprocessor Algorithms», IEEE Trans. Comput. C-31, 278—288
Dennis J., Schnabel, R. [1983]. Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, Englewood Cliffs, New Jersey. [Имеется перевод: Дэннис Дж., Шнабель Р. Численные методы безусловной оптимизации и решение нелинейных уравнений — М: Мир, 1988]
Dongarra J., DuCroz J., Hammarling S., Hanson R. [1986]. «An Update Notice on the Extended BLAS», ACM Signum Newslett 21(4), 2—4
Dongarra J, Eisenstat S [1984]. «Squeezing the Most out of an Algorithm in CRAY-FORTRAN», ACM Trans. Math Softw. 10, 221—230.
Dongarra J., Gustavson F, Karp A [1984]. «Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine», SIAM Rev. 26, 91—112.
Dongarra J., Hewitt T [1086] «Implementing Dense Linear Algebra Algorithms Using Multitasking on the CRAYX-MP-4 (or approaching the gigaflop)», SIAM J Sci Stat. Comput 7, 347—350.
Dongarra J, Hinds A. [1979]. «Unrolling Loops in FORTRAN», Software Pract. Exper 9, 219—229.
Dongarra J., Hinds A. [1985]. «Comparison of the CRAY X-MR-4, Fujitsu VP-200 and Hitachi S-810/20. An Argonne Perspective», Argonne National Laboratory Report No. ANL-8579.

Dongarra J, Johnsson L [1987] «Solving Banded Systems on a Parallel Processor», Parallel Comput 5, 219–246
Dongarra J, Kaufman K, Hammarling S [1986] «Squeezing the Most Out of Eigenvalue Solvers on High Performance Computers», Linear Alg Appl. 77, 113–136
Dongarra J, Sameh A [1984] «On Some Parallel Banded System Solvers», Argonne National Laboratory Report No ANL/MCS-TM-27
Dongarra J, Sameh A, Sorensen D [1986] «Implementation of Some Concurrent Algorithms for Matrix Factorization», Parallel Comput 3, 25—34
Dongarra J, Sorensen D [1987]. «A Portable Environment for Developing Parallel FORTRAN Programs», Parallel Comput 5, 175—186
Dubois M, Briggs F [1982]. «Performance of Synchronized Iterative Processes in Multiprocessor Systems», IEEE Trans Software Eng SE-8, 419–431
Dubois P, Greenbaum A, Rodrigue G [1979]. «Approximating the Inverse of a Matrix for Use in Iterative Algorithms on Vector Processors», Computing 22, 257–268
Duff I [1984]. «The Solution of Sparse Linear Equations on the CRAY-1», in Kowalik [1984], p 293–309
Duff I [1986] «Parallel Implementation of Multifrontal Schemes», Parallel Comput 3, 193—209
Dunigan T [1987]. «Hypercube Performance», in Heath [1987], p. 178–192.

Eisenstat S [1981]. «Efficient Implementation of a Class of Conjugate Gradient Methods», SIAM J Sci Stat Comput 2, 1–4
Eisenstat S, Elman H, Schultz M [1984] «Block-Preconditioned Conjugate Gradient-Like Methods for Numerical Reservoir Simulation», Department of Computer Science Report RR-346, Yale University
Eisenstat S, Heath M, Henkel C, Romine C [1987]. «Modified Cyclic Algorithms for Solving Triangular Systems on Distributed Memory Multiprocessors», SIAM J Sci Stat Comput, to appear
Engeli H, Ginsburg T, Rutischauser H, Stiefel E [1959]. «Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-adjoint Boundary Value Problems» Mitteilungen aus dem Institut fur Angewandte Mathematik, 8, Birkhauser Verlag, Basel
Ericksen J [1972]. «Iterative and Direct Methods for Solving Poisson's Equation and Their Adaptability to ILLIAC IV», Center for Advanced Computation Document No 60, University of Illinois

Fergusson W [1986]. «The Rate of Convergence of a Class of Block Jacobi Schemes», SIAM J Numer Anal 23, 297—303
Fernbach S (ed.) [1986]. Supercomputers, North-Holland, Amsterdam.
Flynn M. [1966]. «Very High Speed Computing Systems», Pros IEEE 54, 1901–1909.
Fong K, Jordan T [1977] «Some Linear Algebraic Algorithms and Their Performance on the CRAY-1», Los Alamos National Laboratory Report No. LA-6774
Forsythe G, Wasow W [1960]. Finite Difference Methods for Partial Differential Equations, Wiley, New York [Имеется перевод: Вазов В, Форсайт Дж. Разностные методы решения дифференциальных уравнений в частных производных — М: ИЛ, 1963]
Fox G, Furmanski W [1987]. «Communication Algorithms for Regular Convolutions and Matrix Problems on the Hupercube», in Heath [1987], p 223–238
Fox G, Kowala A, Williams, R [1987]. «The Implementation of a Dynamic Load Balancer», in Heath [1987], p 114- 121

Fox G., Otto S., Hey A. [1987] «Matrix Algorithms on a Hypercube I: Matrix Multiplication», Parallel Comput. 4, 17—32

Funderlic R., Geist A. [1986] «Torus Data Flow for Parallel Computation of Missized Matrix Problems», Lin. Alg. Appl. 77, 149—163

Gallivan K., Jalby W., Meire U., Sameh A. [1987]. «The Impact of Hierarchical Memory Systems on Linear Algebra Algorithm Design», Center for Supercomputing Research and Development Report No. 625, Univ. of Illinois

Gannon D. [1980] «A Note on Pipelining a Mesh Connected Multiprocessor for Finite Element Problems by Nested Dissection», Proc. 1980 Int. Conf. Parallel Processing, p. 197—204

Gannon D. [1981] «On Mapping Non-Uniform PDE Structures and Algorithms onto Uniform Array Architectures», Proc. 1981 Int. Conf. Parallel Processing, p. 100—105

Gannon D. [1986] «Restructuring Nested Loops on the Alliant Cedar Cluster: A Case Study of Gaussian Elimination of Banded Matrices», Center for Supercomputing Research and Development Report No. 543, University of Illinois

Gannon D., Van Rosendale J. [1984] «On the Impact of Communication Complexity in the Design of Parallel Numerical Algorithms», IEEE Trans. Comput. C-33, 1180—1194

Gannon D., Van Rosendale J. [1986] «On the Structure of Parallelism in a Highly Concurrent PDE Solver», J. Par. Dist. Comput. 3, 106—135

Gao G. [1986] «A Pipelined Solution Method of Tridiagonal Linear Equation Systems», Proc. 1986 Int. Conf. Parallel Processing, pp. 84—91

Gao G. [1987] «A Stability Classification Method and Its Application to Pipelined Solution of Linear Recurrences», Parallel Comput. 4, 305—321

Geist A., Heath M. [1986] «Matrix Factorization on a Hypercube Multiprocessor», in Heath [1986], p. 161—180

Geist G., Romine C. [1987] «LU Factorization Algorithms on Distributed-Memory Multiprocessor Architectures», Oak Ridge National Laboratory Report No. ORNL/TM-10383

Gentleman W. [1975] «Error Analysis of the QR Decomposition by Givens Transformations», Lin. Alg. Appl. 10, 189—197

Gentleman W. [1978]. «Some Complexity Results for Matrix Computations on Parallel Processors», J. ACM 25, 112—115

George A. [1977] «Numerical Experiments Using Dissection Methods to Solve n by n Grid problems», SIAM J. Numer. Anal. 14, 161—179

George A., Chu E. [1987]. «Gaussian Eliminatoin with Partial Pivoting and Load Balancing on a Multiprocessor», Oak Ridge National Laboratory Report No. ORNL/TM-10323

George A., Heath M., Liu J. [1986] «Parallel Cholesky Factorization on a Shared Memory Multiprocessor», Lin. Alg. Appl. 77, 165—187.

George A., Heath M., Ng E. [1986] «Sparse Cholesky Factorization on a Local Memory Multiprocessor», Oak Ridge National Laboratory Report No. ORNL/TM-9962

George A., Heath M., Liu J., Ng E. [1987a] «Solution of Sparse Positive Definite Systems on a Shared-Memory Multiprocessor», Oak Ridge National Laboratory Report No. ORNL/TM-10260

George A., Heath M., Ng E., Liu J. [1987b] «Symbolic Cholesky Factorization on a Local-Memory Multiprocessor», Parallel Comput. 5, 85—96

George A., Liu J. [1981] Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, New Jersey [Имеется перевод: Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений.— М.: Мир, 1983.]

George A., Liu J., Ng E. [1987]. «Communication Reduction in Parallel Sparse Cholesky Factorization on a Hypercube», in Heath [1987], p. 576--586

George A, Poole W, Voight R [1978] «Analysis of Dissection Algorithms for Vector Computers», Comput. Math Appl. 4, 287—304.

Gohberg I, Kailath T, Koltracht I., Lancaster P [1987] «Linear Complexity Parallel Algorithms for Linear Systems of Equations with Recursive Structure», Lin Alg Appl 88, 271—316

Golub G., O'Leary D [1987] «Some History of the Conjugate Gradient and Lanczos Algorithms 1948—1976», Department of Computer Science Report TR-87 20, University of Maryland

Golub G, Plemmons R, Sameh A [1986] «Parallel Block Schemes for Large Scale Least Squares Computations», Center for Supercomputing Research and Development Report No 574, University of Illinois

Golub G, van Loan C [1983] Matrix Computations Johns Hopkins University Press, Baltimore

Gonzalez R, Wheeler, M [1987] «Domain Decomposition for Elliptic Partial Differential Equations with Neumann Boundary Conditions», Parallel Comput 5, 257 - 263

Gottlieb A, Grishman R, Kruskal C, McAuliffe K, Rudolph L, Snir M. [1983] «The NYU Ultracomputer-Designing an MIMD Shared Memory Parallel Computer», IEEE Trans Comput C-32, 175 -189

Grear J, Sameh A [1981] «On Certain Parallel Toeplitz Linear System Solvers», SIAM J Sci Stat Comput 2, 238—256

Greenbaum A [1986a] «Synchronization Costs on Multiprocessors», New York University Ultracomputer Note No 98

Greenbaum A [1986b] «Solving Sparse Triangular Linear Systems Using Fortran with Parallel Extensions on the NYU Ultra-computer Prototype», New York University Ultracomputer Note No 99

Gropp W [1986] «Dynamic Grid Manipulation for PDE's on Hypercube Parallel Processors», Department of Computer Science Report RR-458, Yale University

Gustafsson I. [1978]. «A Class of First Order Factorization Methods», BIT 18, 142 156

Hack J [1986] «Peak vs Sustained Performance in Highly Concurrent Vector Machines», Computer 19(9), 11—19

Hay R, Gladwell I [1985] «Solving Almost Block Diagonal Linear Equations on the CDC Cyber 205», University of Manchester Numerical Analysis Report No 98

Hayes L, Devloo P [1986] «A Vectorized Version of a Sparse Matrix-Vector Multiply», Int J Num Met Eng 23, 1043—1056

Heath M. [1985]. «Parallel Cholesky Factorization in Message-Passing Multiprocessor Environments», Oak Ridge National Laboratory Report No. ORNL-6150

Heath M (ed) [1986] Hypercube Multiprocessors, 1986, Society for Industrial and Applied Mathematics, Philadelphia.

Heath M. (ed) [1987]. Hypercube Multiprocessors, 1987, Society for Industrial and Applied Mathematics, Philadelphia.

Heath M., Romine C. [1987]. «Parallel Solution of Triangular Systems on Distributed-Memory Multiprocessors», Oak Ridge National Laboratory Report No ORNL/TM-10384.

Heath M, Sorensen D [1986] «A Pipelined Givens Method for Computing the QR Factorization of a Sparse Matrix», Lin Alg. Appl. 77, 189—203

Heller D [1976]. «Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems», SIAM J Numer. Anal. 13, 184—496.

Heller D [1978] «A Survey of Parallel Algorithms in Numerical Linear Algebra», SIAM Rev 20, 740—777.
Heller D., Stevenson D., Traub J [1976] «Accelerated Iterative Methods for the Solution of Tridiagonal Linear Systems on Parallel Computers», J ACM 23, 636—654
Hestenes M [1956] «The Conjugate Gradient Methods for Solving Linear Systems», Proc Sixth Symp Appl. Math. McGraw-Hill, New York, 83—102.
Hestenes M., Stiefel E. [1952] «Methods of Conjugate Gradients for Solving Linear Systems» J Res Natl. Bur Stand Sect B 49, 409—436.
Hockney R. [1965] «A Fast Direct Solution of Poisson's Equation Using Fourier Analysis», J ACM 12, 95—113
Hockney R [1970] «The Potential Calculation and Some Applications», Meth. Comput. Phys 9, 135—211
Hockney R [1987] «Parametrization of Computer Performance», Parallel Comput. 5, 97—104
Hockney R., Jesshope C [1981]. «Parallel Computers Architecture, Programming and Algorithms», Adam Hilger, Ltd., Bristol. [Имеется перевод: Хокни Р., Джессхоуп К. Параллельные ЭВМ. М.: Радио и связь, 1986]
Houstis E., Rice J., Vavalis E [1987] «Parallelization of a New Class of Cubic Spline Collocation Methods», in Vichnevetsky and Stepleman [1987].
Hwang K., Briggs, F. (1984). «Computer Architecture and Parallel Processing», McGraw-Hill, New York

Ipsen I. [1984] «A Parallel QR Method Using Fast Givens' Rotation», Department of Computer Science Report RR-299, Yale University
Ipsen I [1987] «Systolic Algorithms for the Parallel Solution of Dense Symmetric Positive-Definite Toeplitz Systems», Department of Computer Science Report RR-539, Yale University.
Ipsen I., Saad Y., Schultz M [1986]. «Complexity of Dense Linear System Solution on Multiprocessor Ring», Lin Alg. Appl. 77, 205—239.

Jalby W., Meier U (1986]. «Optimizing Matrix Operations on a Parallel Multiprocessor with a Memory Hierarchy», Center for Supercomputing Research and Development Report No 555, University of Illinois.
Jesshope C., Hockney R (eds.). [1979] Infotech State of the Art Report: Supercomputers, Vols 1 & 2, Infotech Int Ltd., Maidenhead.
Johnson O., Micchelli C., Paul G. [1983] «Polynomial Preconditioners for Conjugate Gradient Calculations», SIAM J Numer Anal 20, 362—376.
Johnsson L [1985a]. «Data Permutations and Basic Linear Algebra Computations on Ensemble Architectures», Department of Computer Science Report RR-367, Yale University
Johnsson L [1985b]. «Solving Narrow Banded Systems on Ensemble Architectures», ACM Trans Math Software 11, 271—288.
Johnsson L. [1987a]. «Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures», J Par Dist Comp 4, 133—172.
Johnsson L. [1987b] «Solving Tridiagonal Systems on Ensemble Architectures», SIAM J Sci Stat Comput. 8, 354—392
Jordan H [1986] «Structuring Parallel Algorithms in an MIMD, Shared Memory Environment», Parallel Comput 3, 93—110.
Jordan T [1982] «A Guide to Parallel Computation and some CRAY-1 Experiences», in Rodrigue [1982], p 1—50

Kamowitz D [1987] «SOR and MGR[v] Experiments on the Crystal Multicomputers», Parallel Comput. 4, 117—142
Kapur R., Browne J (1984] «Techniques for Solving Block Tridiagonal Systems on Reconfigurable Array Computers», SIAM J. Sci. Stat. Comput. 5, 701—719.

Kascic M [1979] «Vector Processing on the CYBER 200», in Jesshope and Hockney [1979], p 237 - 270
Kershaw D [1978] «The Incomplete Choleski-Conjugate Gradient Method for the Iterative Solution of Systems of Linear Equations», J Comp Phys. 26, 43--65
Kershaw D [1982] «Solution of Single Tridiagonal Linear Systems and Vectorization of the ICCG Algorithm on the CRAY-1», in Rodrigue [1982], p 85—89
Keyes D, Gropp W [1987] «A Comparison of Domain Decomposition Techniques for Elliptic Partial Differential Equations and Their Parallel Implementation», SIAM J Sci Stat Comput 8, s166—s202
Kightley J, Jones I [1985] «A Comparison of Conjugate Gradient Preconditionings for Three-Dimensional Problems in a CRAY-1», Comput Phys. Commun 37, 205- 214
Kightley J, Thompson C [1987] «On the Performance of Some Rapid Elliptic Solvers on a Vector Processor», SIAM J Sci Stat Comput 8, 701—715
Kincaid D., Oppe T, Young D [1986a] «Vector Computations for Sparse Linear Systems», SIAM J Algebraic Discrete Methods 7, 99—112
Kincaid D, Oppe T, Young D [1986b]. «Vectorized Iterative Methods for Partial Differential Equations», Commun Appl Numer Math 2, 789—796
Kogge P [1981] «The Architecture of Pipelined Computers», McGraw-Hill, New York
Kowalik J (ed) [1984] Proceedings of the NATO Workshop on High Speed Computations, West Germany, NATO ASI Series, vol F-7, Springer-Verlag, Berlin
Kowalik J, Kumar S [1982] «An Efficient Parallel Block Conjugate Gradient Method for Linear Equations», Proc 1982 Int Conf Parallel Processing, p 47--52 [Имеется перевод Высокоскоростные вычисления Архитектура, производительность, прикладные алгоритмы и программы суперЭВМ/Под ред Я Ковалика М · Радио и связь, 1988]
Kuck D [1976] Parallel Processing of Ordinary Programs, Advances in Computers 15, Academic Press, New York, p 119—179
Kuck D [1977] «A Survey of Parallel Machine Organization and Programming», ACM Comput Surv 9, 29 59
Kuck D [1978] «The Structure of Computers and Computation», Wiley, New York
Kuck D, Lawrie D, Sameh A (eds) [1977] «High Speed Computer and Algorithm Organization», Academic Press, New York
Kumar S, Kowalik J [1984] «Parallel Factorization of a Positive Definite Matrix on an MIMD Computer», Proc 1984 Int Conf Parallel Processing, p 410—416.
Kumar S, Kowalik J [1986] «Triangularization of a Positive Definite Matrix on a Parallel Computer», J Par Dist Comp 3, 450—460
Kung H [1976] «Synchronized and Asynchronous Parallel Algorithms for Multi-processors», Algorithms and Complexity, J Traub (ed), Academic Press, New York, 153- -200
Kung H [1980]. «The Structure of Parallel Algorithms», Advances in Computers 19, M Yovitts (ed), Academic Press, New York, p. 65—112
Kung H [1982]. «Why Systolic Architectures?» Computer 15 (1), 37—46
Kung H [1984] «Systolic Algorithms», in Parter 1984, p 127—140.
Kung S. [1984] «On Supercomputers with Systolic/Wavefront Array Processors», Proc IEEE 72, 862—884.

Kuo J, Levy B, Muskus B [1987]. «A Local Relaxation Method for Solving Elliptic PDEs on Mesh Connected Arrays», SIAM J. Sci Stat. Comput. 8, 550—573.

Lakshmivarahan S, Dhall S. [1987]. «A lower Bound on the Communication Complexity in Solving Linear Tridiagonal Systems on Cube Architectures», in Heath 1987, p. 560—568

Lambiotte J [1975]. «The Solution of Linear Systems of Equations on a Vector Computer», Ph D. dissertation, University of Virginia

Lambiotte J, Voigt R [1975] «The Solution of Tridiagonal Linear Systems on the CDC STAR-100 Computer», ACM Trans Math Software 1, 308—329.

Larson J [1984] «Multitasking on the CRAY X-MR-2 Multiprocessor», Computer 17(7), 62—69.

Lawrie D, Sameh A [1984] «The Computation and Communication Complexity of a Parallel Banded System Solver», ACM Trans Math. Software 10, 185—195.

LeBlanc T. [1986] «Shared Memory versus Message Passing in a Tightly Coupled Multiprocessor A Case Study», Computer Science Department Report, University of Rochester

Lewis J., Simon H [1986] «The Impact of Hardware Gather/Scatter on Sparse Gaussian Elimination», Proc 1986 Int Conf. Parallel Processing, p 366—368

Li G., Coleman T [1987a] «A Parallel Triangular Solver for a Hypercube Multiprocessor», in Heath 1987, p. 539—551.

Li G, Coleman T [1987b] «A New Method for Solving Triangular Systems on Distributed Memory Message-Passing Multiprocessors», Computer Science Report TR 87-812, Cornell University

Lichnewsky A [1984] «Some Vector and Parallel Implementations for Preconditioned Conjugate Gradient Algorithms», in Kowalik 1984, p. 343—359.

Lim D, Thanakij R [1987] «A Survey of ADI Implementations on Hypercubes», in Heath 1987, pp 674—679

Lincoln N [1982] «Technology and Design Tradeoffs in the Creation of a Modern Supercomputer», IEEE Trans Comput C-31, 349—362

Liu J [1986] «Computational Models and Task Scheduling for Parallel Sparse Cholesky Factorization», Parallel Comput 3, 327—342

Liu J [1987] «Reordering Sparse Matrices for Parallel Elimination», Department of Computer Science Report No. CS-87-01, York University, Ontario, Canada

Lord R, Kowalik J, Kumar S 1980 «Solving Linear Algebraic Equations on a MIMD Computer», Proc 1980 Int Conf Parallel Processing, p. 205—210

Louter-Nool M [1987] «Basic Linear Algebra Subprograms (BLAS) on the CDC CYBER 205», Parallel Comput 4, 143—166.

Luk F [1986] A Rotation Method for Computing the QR-Decomposition», SIAM J Sci Stat Comput 7, 452—459

Madsen N, Rodrigue G, Karush J [1976]. «Matrix Multiplication by Diagonals on a Vector/Parallel Processor», Inf Proc Lett 5, 41—45

Manteuffel T [1980]. «An Incomplete Factorization Technique for Positive Definite Linear Systems», Math Comput 34, 473—497.

Mattingly B., Meyer C, Ortega J [1987]. «Orthogonal Reduction on Vector Computers», SIAM J Sci. Stat Comput. To appear

McBryan O [1987] «Numerical Computation on Massively Parallel Hypercubes», in Heath 1987, p 706—719

McBryan O, van de Velde E [1985]. «Parallel Algorithms for Elliptic Equations», Commun Pure Appl. Math. 38, 769—795.

McBryan O., van de Velde E. [1986a] «Elliptic Equation Algorithms on Parallel Computers», Commun Appl Numer Methods 2, 311–316
McBryan O., van de Velde E. [1986b] «Hypercube Programs for computational Fluid Dynamics», in Heath 1986, p 221—243
McBryan O., van de Velde E [1987a]. «Hypercube Algorithms and Implementations», SIAM J Sci. Stat Comput 8, s227—s287
McBryan O., van de Velde E [1987b]. «Matrix and Vector Operations on Hypercube Parallel Processors», Parallel Comput 5,117—126
Meier U. [1985[«A Parallel Partition Method for Solving Banded Systems of Linear Equations», Parallel Comput 2, 33—43
Meijerink J., van der Vorst H [1977] «An Iterative Solution for Linear Systems of which the Coefficient Matrix is a Cymmetric M-Matrix», Math. Comput 31, 148—162
Meijerink J., van der Vorst H. [1981] «Guidelines for the Usage of Incomplete Decompositions in Solving Sets on Linear Equations as They Occur in Practical Problems», J. Comput Phys 44, 134—155
Melhem R [1987a]. «Determination of Stripe Structures for Finite Element Matrices», SIAM J Numer Anal 24, 1419—1433
Melhem R [1987b]. «Parallel Solution of Linear Systems with Striped Sparse Matrices», Parallel Comput, to appear
Meurant G. [1984] «The Block Preconditioned Conjugate Gradient Method on Vector Computers», BIT 24, 623—633
Miranker W [1971] «A Survey of Parallelism in Numerical Analysis», SIAM Rev 13, 524–547
Modi J., Clarke M [1984]. «An Alternative Givens Ordering», Numer. Math. 43, 83—90
Moler C [1972] «Matrix Computations with Fortran and Paging». Commun ACM 15, 268—270
Moler C [1986] «Matrix Computation on Distributed Memory Multiprocessors», in Heath 1986, p 181—195
Morison R., Otto S [1987]. «The Scattered Decomposition for Finite Elements», J Sci Comput 2, 59—76

Naik V., Ta'asan S [1987]. «Performance Studies of the Multigrid Algorithms Implemented on Hypercube Multiprocessor Systems», in Heath 1987, p 720—729
Neta B., Tai H.-M. [1985] «LU Factorization on Parallel Computers», Comput. Math Appl. 11, 573–580
Neumann M., Plemmons R [1987]. «Convergence of Parallel Multisplitting Iterative Methods for M-Matrices», Lin Alg. Appl 88, 559—575.
Nodera T 1984 «PCG Method for the Four Color Ordered Finite Difference Schemes», in Vichnevetsky and Stepleman [1984], p 222–228
Noor A., Kamel H., Fulton R. [1978] «Substructuring Techniques — Status and Projections», Comput Structures 8, 621—632.
Nour-Omid B., Park K [1987]. «Solving Structural Mechanics Problems on the Caltech Hypercube», Comput Methods Appl Mech. Eng 61, 161—176.
Nour-Omid B., Parlett B [1987] «Element Preconditioning Using Splitting Techniques», SIAM J Sci. Stat. Comput 6, 761—770.

O'Leary D [1984]. «Ordering Schemes for Parallel Processing of Certain Mesh Problems», SIAM J Sci Stat Comput 5, 620—632
O'Leary D. [1987]. «Parallel Implementation of the Block Conjugate Gradient Algorithm», Parallel Comput. 5, 127—140.
O'Leary D., Stewart G. [1985]. «Data-Flow Algorithms for Parallel Matrix Computations», Commun. ACM 28, 840—853.

O'Leary D, White R [1985]. «Multi-splittings of Matrices and Parallel Solution of Linear Systems», SIAM J Alg. Discrete Methods 6, 630—640.
Onaga K, Takechi T [986]. «A Wavefront Algorithm for LU Decomposition of a Partitioned Matrix on VLSI Processor Arrays», J Par Dist. Comput. 3, 137—157
Oppe T., Kincaid D [1987] «The Performance of ITPACK on Vector Computers for Solving Large Sparse Linear Systems Arising in Sample Oil Reservoir Simulation Problems», Commun Appl Numer Math 3, 23—30
Opsahl T, Parkinson D [1986]. «An Algorithm for Solving Sparse Sets of Linear Equations with an Almost Tridiagonal Structure on SIMD Computers», Proc 1986 Int Conf Parallel Processing, pp 369—374
Ortega J [1972]. «Numerical Analysis A Second Course», Academic Press, New York
Ortega J [1987a] «Matrix Theory A Second Course», Plenum Press, New York
Ortega J [1987b] «The ijk Forms of Factorization Methods I. Vector Computers», Parallel Comp, to appear
Ortega J [1987c] «Efficient Implementations of Certain Iterative Methods», Applied Mathematics Report No RM-87-02, University of Virginia.
Ortega J, Romine C [1987] «The ijk Forms of Factorization Methods II. Parallel Computers», Applied Mathematics Report No RM-87-01, University of Virginia
Ortera J, Voigt R [1985] «Solution of Partial Differential Equations on Vector and Parallel Computers», SIAM Rev 27, 149—240
Ortega J, Voigt R [1987] «A Bibliography on Parallel and Vector Numerical Algorithms», ICASE Report I-3, NASA-Langley Research Center

Paker Y. [1983] «Multi-Microprocessor Systems», Academic Press, New York.
Parkinson D. [1987] «Organizational Aspects of Using Parallel Computers», Parallel Comput 5, 75—84
Parter S. (Ed) [1984] «Large Scale Scientific Computation», Academic Press, Orlando, Florida
Parter S., Steuerwalt S [1980]. «On k-line and $k \times k$ Block Iterative Schemes for a Problem Arising in 3-D Elliptic Difference Equations», SIAM J. Numer Anal 17, 823—839
Parter S, Steuerwalt M [1982] «Block Iterative Methods for Elliptic and Parabolic Difference Equations», SIAM J. Numer Anal 19, 1173—1195
Patel N, Jordan H [1984] «A Parallelized Point Rowwise Successive Over-Relaxation Method on a Mulitprocessor», Parallel Comput 1, 207—222
Patrick M, Reed D, Voigt R [1987]. «The Impact of Domain Partitioning on the Performance of a Shared Memory Multiprocessor», Parallel Comput 5, 211—218
Peters F [1984]. «Parallel Pivoting Algorithms for Sparse Symmetric Matrices», Parallel Comput 1, 99—110
Pfister G, Brantley W, George D, Harvey S, Kleinfelder W, McAuliffe K., Melton E, Norton V, Weiss J [1985]. «The IBM Research Parallel Processor Prototype (RP3). Introduction and Architecture», Proc 1985 Int. Conf. Parallel Processing, p. 764—771
Plemmons R. [1986]. «A Parallel Block Iterative Scheme Applied to Computations in Structural Analysis», SIAM J Algebraic Discrete Methods 7, 337—347
Poole E [1986] «Multicolor Incomplete Cholesky Conjugate Gradient Methods on Vector Computers», Ph. D dissertation, Applied Mathematics, University of Virginia
Poole E., Ortega J [1987]. «Multicolor ICCG Methods for Vector Computers», SIAM J. Numer. Anal 24, 1394—1418.

Pothen A., Jha S., Vemulapati U. [1987]. «Orthogonal Factorization on a Distributed Memory Multiprocessor», in Heath 1987, p. 587—596.

Reed D., Adams L., Patrick M. [1987]. «Stencils and Problem Partitionings: Their Influence on the Performance of Multiple Processor Systems», IEEE Trans. Comput. TC 36, 845—858.
Reed D., Patrick M. [1984]. «A Model of Asynchronous Iterative Algorithms for Solving Large Sparse Linear Systems», Proc. 1984 Int. Conf. Parallel Processing, p. 402—409.
Reed D., Patrick M. [1985a]. «Parallel Iterative Solution of Sparse Linear Systems: Models and Architectures», Parallel Comput 2, 45—67.
Reed D., Patrick M. [1985b]. «Iterative Solution of Large Sparse Linear Systems on a Static Data Flow Architecture. Performance Studies», IEEE Trans. Comput. C-34, 874—881
Reid J. [1971]. «On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations», Proc. Conf. Large Sparse Sets of Linear Equations, Academic Press, New York.
Reiter E., Rodrigue G. [1984]. «An Incomplete Choleski Factorization by a Matrix Partition Algorithm», in Birkhoff and Schoenstadt 1984, p. 161—173.
Robert I [1982]. «Regular Imcomplete Factorizations of Real Positive Definite Matrices», Lin. Alg. Appl. 48, 105—117.
Rodrigue G. (ed.) [1982]. «Parallel Computations», Academic Press, New York. [Имеется перевод: Параллельные вычисления/Под ред. Г. Родрига. — М.: Наука, 1986.]
Rodrigue G. [1986]. «Some Ideas for Decomposing the Domain of Elliptic Partial Differential Equations in the Schwarz Process», Commun. Appl. Numer. Meth. 2, 245—249.
Rodrigue G., Wolitzer D. [1984] «Preconditioning by Incomplete Block Cyclic Reduction», Math. Comput. 42, 549—565.
Romine C. [1986]. «Factorization Methods for the Parallel Solution of Linear Systems», Ph D. dissertation, Applied Mathematics, University of Virginia
Romine C., Ortega J. [1988]. «Parallel Solution of Triangular Systems of Equations», Parallel Comput. 6, 109—114.
Ronsch W. [1984]. «Stability Aspects in Using Parallel Algorithms», Parallel Comput. 1, 75—98.

Saad Y [1985]. «Practical Use of Polynomial Preconditionings for the Conjugate Gradient Method», SIAM J. Sci. Stat. Comput. 6, 865—882.
Saad Y. [1986a]. «Communication Complexity of the Gaussian Elimination Algorithm on Multiprocessors», Lin Alg. Appl. 77, 315—340.
Saad Y. [1986b]. «Gaussian Elimination on Hypercubes», Department of Computer Science Report RR-462, Yale University.
Saad Y., Sameh A., Saylor P. [1985]. «Solving Elliptic Difference Equations on a Linear Array of Processors», SIAM J. Sci. Stat. Comput. 6, 1049—1063.
Saad Y., Schultz M. [1985]. «Topological Properties of Hypercubes», Department of Computer Science Report RR-389, Yale University.
Saad Y., Schultz M. [1986]. «Data Communications in Parallel Architectures», Department of Computer Science Report RR/461, Yale University.
Saad Y., Schultz M. [1987]. «Parallel Direct Methods for Solving Banded Linear Systems», Lin Alg. Appl. 88, 623—650.
Saied F, Ho C.-T., Johnsson L., Schultz M. [1987] «Solving Schrodinger's Equation on the Intel iPSC by the Alternating Direction Method, in Heath 1987, pp. 680—691.

Saltz J, Naik V., Nicol D. [1987]. «Reduction of the Effects of the Communication Delays in Scientific Algorithms on Message Passing MIMD Architectures», SIAM J. Sci. Stat Comput. 8, s118—s138.
Sameh A [1985]. «On Some Parallel Algorithms on a Ring of Processors», Comput. Phys. Commun. 37, 159—166
Sameh A, Kuck D. [1978]. «On Stable Parallel Linear System Solvers», J. ACM 25, 81—91
Schnendel U. [1984]. Introduction to Numerical Methods for Parallel Computers (translator, B W. Conolly), Halsted Press.
Schonauer W. [1983]. «Numerical Experiments with Instationary Jacobi-OR Methods for the Iterative Solution of Linear Equations», ZAMM 63, p. T380—T382.
Schreiber R [1986]. «On Systolic Array Methods for Band Matrix Factorizations», BIT 26, 303 -316
Schreiber R, Tang W. [1982]. «Vectorizing the Conjugate Gradient Method», in Control Data Corp 1982 in Proceedings Symposium CYBER 205 Applications, Fort Collins, Colorado.
Seager M [1986a]. «Overhead Considerations for Parallelizing Conjugate Gradient», Commun. Appl Numer. Math. 2, 273—279
Seager M. [1986b]. «Parallelizing Conjugate Gradient for the CRAY X-MP», Parallel Comput 3, 35—48
Seitz C [1985]. «The Cosmic Cube», Commun. ACM 28, 22—33.
Shanehchi J., Evans D [1982]. «Further Analysis of the QIF Method», Int. J. Comput Math. 11, 143– 154.
Snyder L, Jamieson L, Gannon D, Siegel H. (eds.) [1985]. Algorithmically Specialized Parallel Computers, Academic Press, Orlando, Florida.
Sorensen D. [1984]. «Buffering for Vector Performance on a Pipelined MIMD Machine», Parallel Comput 1, 143—164.
Sorensen D. [1985]. «Analysis of Pairwise Pivoting in Gaussian Elimination», IEEE Trans. Comput C-34, 274—278.
Stewart G W. [1973]. Intoroduction to Matrix Computations, Academic Press, New York
Stone H. [1973]. «An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations», J. ACM 20, 27—38.
Stone H [1975]. «Parallel Tridiagonal Equation Solvers», ACM Trans. Math. Software 1, 289—307.
Stone H. [1987]. High-Performance Computer Architecture, Addison-Wesley, Reading, MA
Storaasli O., Peebles S., Crockett T., Knott J., Adams L. [1982]. «The Finite Element Mashine: An Experiment in Parallel Processing», Proc of Conf. on Res. in Structures and Solid Mech., NASA Conf. Publ. 2245, NASA Langley Research Center, p. 201—217.
Swarztrauber P [1979]. «A Parallel Algorithm for Solving General Tridiagonal Equations», Math Comput. 33, 185—199.

Traub J. [1974]. «Iterative Solution of Tridiagonal Systems on Parallel or Vector Computers», in Traub, [1974], p 49 82
Traub J. (ed.) [1974]. «Complexity of Sequential and Parallel Numerical Algorithms», Academic Press, New York.

Uhr L. [1984]. «Algorithm Structured Computer Arrays and Networks», Academic Press, Orlando, Florida.

van der Sluis A., van der Vorst H. [1986]. «The Rate of Convergence of Conjugate Gradients», Numer Math 48, 543—560.
van der Vorst H. [1982]. «A Vectorizable Variant of Some ICCG Methods», SIAM J. Sci. Stat. Comput. 3, 350—356.

van der Vorst H. [1983]. «On the Vectorization of Some Simple ICCG Methods», First Int Conf Vector and Parallel Computation in Scientific Applications, Paris, 1983.
van der Vorst H [1986]. «Analysis of a Parallel Solution Method for Tridiagonal Systems», Department of Mathematics and Information Report No. 86-06, Delft University of Technology
van der Vorst H [1987] «Large Tridiagonal and Block Tridiagonal Linear Systems on Vector and Parallel Computer», Parallel Comput. 5, 45—54.
Varga R. [1960]. «Factorization and Normalized Iterative Methods», in Boundary Problems in Differential Equations (Rudolph E. Langer, ed), University of Wisconsin Press, Madison, p 121—142
Varga R. [1962]. «Matrix Iterative Analysis», Prentice Hall, Englewood Cliffs, New Jersey.
Vaughan C, Ortega J [1987]. «SSOR Preconditioned Conjugate Gradient on a Hypercube», in Heath 1987, pp 692—705
Veen A. [1986]. «Dataflow Machine Architecture», ACM Comput. Surveys 18, 365—396
Vichnevetsky R., Stepleman R (eds) [1987]. «Advances in computational Methods for Partial Differential Equations — VI», Proc of the Sixth IMACS International Symposium, IMACS, New Brunswick, Canada.
Voigt R [1977], «The Influence of Vector Computer Architecture on Numerical Algorithms», in Kuck et al. [1977], pp 229—244.

Wang H [1981]. «A Parallel Method for Tridiagonal Equations», ACM Trans. Math. Software 7, 170—183.
Wachspress E. [1984]. «Navier-Stokes Pressure Equation Iteration», in Birkhoff and Schoenstadt [1984], p 315—322
Ware W. [1973]. «The Ultimate Computer», IEEE Spect. 10 (3), 89—91.
White R. [1987] «Multisplittings of a Symmetric Positive Definite Matrix», Comput. Meth Appl Mech. Eng 64, 567—578.
Wing O., Huang J. [1977]. «A Parallel Triangulation Process of Sparse Matrices», Proc 1977 Int Conf Parallel Processing, p. 207—214
Wing O., Huang J. [1980]. «A Computational Model of Parallel Solutions of Linear Equations», IEEE Trans Comput C-29, 632—638.

Young D [1971]. Iterative Solution of Large Linear Systems, Academic Press, New York.
Young D., Oppe T., Kincaid D., Hayes L. [1985]. «On the Use of Vector Computers for Solving Large Sparse Linear Systems», Center for Numerical Analysis Report No. CNA-199, University of Texas at Austin.

Добавление

О предобусловливании и распараллеливании метода сопряженных градиентов

И. Е. Капорин

Судя по нарастающему количеству публикаций, посвященных параллельной реализации предобусловленного метода сопряженных градиентов, соответствующее направление исследований является довольно перспективным. Поэтому мы сочли целесообразным в порядке дополнения к материалу параграфов 3.3 и 3.4 провести рассмотрение вопроса под несколько иным углом зрения, приняв во внимание некоторые недавние результаты, не вошедшие в книгу.

1. Реальные задачи и их решение на мультипроцессорах. В гл. 3 изложение в основном строится на очень простых модельных задачах (что, впрочем, вполне естественно для вводного курса). Обратим внимание читателя на весьма поучительные затруднения, возникающие при переходе от пятиточечного шаблона к девятиточечному (см. обсуждение рис. 3.2.7, рис. 3.4.4 и упражнение 3.4.15), а также на довольно внушительный вид шаблона, отвечающего применению квадратичных лагранжевых элементов (см. рис. 3.2.9с). Принимая во внимание наблюдающуюся тенденцию к усложнению структур дискретных задач подобного типа (здесь могут действовать следующие факторы: переход от метода конечных разностей к методу конечных элементов и увеличение порядка конечных элементов [1]; применение метода граничных элементов [2]; усложнение сеток и разбиений области на элементы; переход от методов расщепления по физическим и пространственным переменным к «целостным» дискретизациям систем уравнений, описывающих трехмерные объекты), можно представить себе трудности, возникающие при попытках обобщения описанных «классических» методов на реальные ситуации. Матрицы соответствующих систем имеют довольно большой размер (до сотен тысяч элементов), достаточно сильно заполнены (до сотен и даже тысяч ненулевых элементов в каждой строке), не имеют диагонального преобладания, не являются M-матрицами и довольно плохо обусловлены. В общем случае можно рассчитывать лишь на симметрию и положительную определенность матрицы системы;

при этом соответствующая часть теоретических результатов, приведенных в книге (т. е. полученных для матриц с диагональным преобладанием и M-матриц), не сохраняется.

Понятно, что эффективное решение подобных алгебраических задач возможно лишь при использовании суперкомпьютеров с параллельной и/или векторной обработкой данных. Практическое внедрение параллельных компьютеров с гиперкубической структурой межпроцессорных связей, имеющих очень большое число процессоров, остро ставит вопрос о построении алгоритмов (в частности, реализующих предобусловленный метод сопряженных градиентов), обладающих, помимо высокой степени параллелизма, как можно более крупной зернистостью. Иначе говоря, желательно, чтобы процессоры не только были хорошо загружены, но и занимались в основном обработкой содержимого своей локальной памяти, лишь изредка обмениваясь пакетами данных. Причина такого требования — высокая цена инициализации обмена (т. е. величина s в формуле (1.1.5)) для подобного рода архитектур ЭВМ, оказывающаяся порой сравнимой со временем выполнения нескольких сотен и даже тысяч арифметических операций с плавающей запятой [3, 4], [Heat, 1987]. Построение таких алгоритмов является сложной и фактически нерешенной задачей, в контексте которой будут обсуждаться способы предобусловливания метода сопряженных градиентов.

2. Для чего вообще нужны итерационные методы? Действительно, существующие прямые методы, основанные на разреженном гауссовом исключении (см. литературу к § 2.3), реализованы в виде достаточно совершенных пакетов стандартных программ, и они позволяют получить точное (если не помешают ошибки округления) решение системы с разреженной симметричной положительно определенной матрицей за конечное число операций. Оказывается, однако, что область эффективной применимости этих методов ограничивается задачами с матрицами, структура разреженности которых описывается плоскими (и близкими к ним) графами [5, 6]; так, для простейших трехмерных сеточных задач типа описанной в упражнении 3.4.14 даже асимптотически неулучшаемые методы вложенных сечений оказываются неэффективными из-за возникающего интенсивного заполнения блоков вычисляемой факторизации. В то же время итерационные методы дают надежду на более быстрое отыскание решения; в самом деле, для той же задачи уже явный метод сопряженных градиентов достигает приемлемой точности за гораздо меньшее число операций. Помимо потенциального выигрыша в случае структуры разреженности

общего вида, итерационные методы имеют дополнительные преимущества. Главное из них, особенно с точки зрения параллельной (векторной, систолической, ...) реализации, заключается в том, что алгоритмы итерационных методов базируются на структурах данных, наследуемых от исходной задачи, и/или на структурах данных, родственных исходным или сконструированных априори (в сущности, по произволу разработчика метода). Понятно, что здесь в принципе открываются гораздо более широкие возможности для адаптации алгоритма к архитектуре того или иного суперкомпьютера по сравнению с прямыми методами разреженного исключения, где возникающие структуры данных отличаются существенно меньшей предсказуемостью и управляемостью.

3. Как строить предобусловливатель? Напомним, что при построении предобусловливателя, то есть матрицы $H = H^{\mathrm{T}} > 0$, фигурирующей в алгоритме метода сопряженных градиентов

$$
\begin{aligned}
&r_0 = b - Ax_0;\\
&p_0 = Hr_0;\\
&\text{Для } k = 0,\ 1,\ \ldots\\
&\alpha_k = (p_k^{\mathrm{T}} A p_k)^{-1} (r_k^{\mathrm{T}} H r_k),\\
&x_{k+1} = x_k + p_k \alpha_k,\\
&r_{k+1} = r_k - A p_k \alpha_k,\\
&\beta_k = (r_k^{\mathrm{T}} H r_k)^{-1} (r_{k+1}^{\mathrm{T}} H r_{k+1}),\\
&p_{k+1} = H r_{k+1} + p_k \beta_k,
\end{aligned}
$$

основным требованием является «близость» H к матрице A^{-1} в сочетании с возможностью «быстрого» вычисления произведения вида Hr. Первое условие призвано обеспечить сокращение числа итераций, а второе — ограничить возрастание затрат, связанных с введением предобусловливания. Эти условия (вообще говоря, взаимоисключающие) обычно вводятся в следующие рамки: во-первых, фиксируется некоторая структура предобусловливателя, однозначно определяющая число операций для вычисления Hr, а также потенциальные возможности приближения к A^{-1}, и, во-вторых, осуществляется конкретный выбор числовых значений свободных параметров, заложенных в конструкцию H (если таковые предусмотрены), обеспечивающих действительную близость H к A^{-1} в том или ином смысле.

Заметим, что одно из основных неудобств, возникающих при использовании итерационных методов для решения реальных задач,— это практически непредсказуемое число итераций, необходимое для достижения фиксированного критерия точности

(скажем, $r_k^{\mathrm{T}} H r_k \leqslant 10^{-12} r_0^{\mathrm{T}} r_0$). Подходящее предобусловливание как раз и служит средством удержания числа итераций в разумных пределах. Особенно полезным может оказаться сокращение числа итераций метода сопряженных градиентов при параллельной реализации на большом числе процессоров (даже если общее число операций несколько возрастает, как это бывает при полиномиальном предобусловливании [Johnson O. et al., 1983]). Действительно, если p — число процессоров и компоненты итерируемых векторов распределены по всем процессорам, то для вычисления скалярных произведений на каждой итерации потребуется не менее $O(\log p)$ последовательных этапов инициализации обменов, и эти затраты могут оказаться относительно большими.

3.1. Структуры предобусловливателей. Структура предобусловливателя, очевидно, определяет не только число арифметических операций для вычисления произведения Hr, но и возможность эффективного распараллеливания или векторизации этой операции. Поэтому развитие архитектур суперкомпьютеров дало мощный стимул к поиску нетрадиционных структур предобусловливателей. Обсудим некоторые подходы, известные к настоящему времени.

3.1.1. Неявные предобусловливатели. Здесь и далее будем считать, что предварительно проведено определенное переупорядочение и блочное (возможно, точечное) разбиение матрицы системы A. Рассмотрим, как и в § 3.4, предобусловливание типа неполного треугольного разложения без заполнения, где матрица H задается неявно: $H^{-1} = (D + L)E^{-1}(D + L^{\mathrm{T}})$; здесь D и E — диагональные матрицы, а L — строго нижнетреугольная матрица. Неразрешимая, по-видимому, дилемма заключается в следующем: при «естественном» упорядочении матриц сеточных задач часто удается получить хорошее качество предобусловливателя, но возможности распараллеливания оказываются совершенно недостаточными даже для простейших сеточных шаблонов и не очень большого числа процессоров [7, 8] (см. также [9, 10], где анализируются сходные алгоритмы); если же применить упорядочение типа «красно-черного», то параллелизм увеличивается, но качество предобусловливания падает, так как здесь получается, в сущности, разновидность явного предобусловливания (см. ниже). Другая дилемма — принять ли матрицу L равной нижней треугольной части матрицы A, что позволяет применить прием Айзенштата и тем самым удешевить каждую итерацию, или отказаться от этого ограничения в пользу потенциально более точного разложения типа IC(0). Для

матриц достаточно общего вида подобные методы еще требуют дальнейшей разработки; так, в [11] для решения системы, отвечающей дискретизации трехмерной системы уравнений теории упругости по методу конечных элементов высокого порядка, построен блочный симметричный метод Гаусса — Зейделя с использованием ускорения сопряженными градиентами, приема Айзенштата, локального красно-черного упорядочения, а также (в качестве предварительного этапа) понижения порядка исходной системы при помощи явного исключения части неизвестных.

3.1.2. Явные предобусловливатели. Способы явного (точнее, локального) предобусловливания предусматривают, как правило, использование разреженных матриц H. Простейшим из них является масштабирование, т. е. выбор диагональной матрицы H. Большего эффекта можно добиться, применив блочный вариант этой конструкции при достаточно большом размере блоков (см. обсуждение блочного метода Якоби в конце § 3.4). Дальнейшее улучшение можно получить, используя «перекрывающиеся» диагональные блоки, отвечающие декомпозиции области на перекрывающиеся подобласти [4]; при этом, конечно, возможно использование приближенного обращения диагональных блоков большого размера при помощи, например, неполного треугольного разложения. Как отмечено в [4], при этом иногда удается получить даже меньшее число итераций по сравнению с исходным стандартным методом неполного треугольного разложения, в то время как затраты на итерацию сравнимы, а параллелизм оказывается неизмеримо выше.

Заметим, что сходные структуры возникают и при применении «поэлементного» предобусловливания (см. п. 12 дополнений к § 3.4, а также работу [12]).

Частичное исключение неизвестных (см., например, [7, 11, 13], а также метод решения системы (3.3.18)) также можно трактовать как явное предобусловливание.

К этому же классу можно отнести полиномиальные предобусловливания, которые были описаны в § 3.4. Для их эффективной реализации можно использовать, с целью минимизации числа этапов инициализации обменов, технику дублирования данных, описанную, например, в [14].

Другая возможность, возникающая, когда в качестве матрицы H выбирается разреженная матрица, каждый ненулевой элемент которой является параметром предобусловливания, обсуждается в работах [15—18]. Правда, существующие способы выбора параметров не всегда позволяют обеспечить симметрию и положительную определенность одновременно со свойством

«оптимальности» матрицы H. Указанную трудность можно преодолеть, если строить предобусловливатель в факторизованном виде $H = G^{\mathrm{T}}G$, где G — нижнетреугольная матрица с предписанной структурой разреженности; теперь в качестве параметров предобусловливания фигурируют уже ненулевые элементы матрицы G. Такой подход описан в работе [19]; там же дается блочный вариант, позволяющий использовать приближенное обращение нужных подматриц исходной матрицы. Такие предобусловливатели допускают весьма эффективное распараллеливание, однако повышение их качества связано с увеличением заполнения G, что, в свою очередь, повышает затраты на вычисление предобусловливателя и на его применение в процессе итерации. Поэтому перспективными представляются заполненные матрицы-предобусловливатели, подход к построению которых также можно найти в [19], где рассматривается нижнетреугольная матрица G, каждая строка которой представляет собой линейную комбинацию заранее заданных векторов специального вида (не обязательно единичных, а, возможно, более заполненных).

3.2. Выбор параметров предобусловливания. Заключительный этап построения предобусловливателя, связанный с конкретной фиксацией значений элементов матрицы H, является все еще в значительной степени нестрогой процедурой, основанной на правдоподобных рассуждениях и частных результатах. Конечно, известно большое число теоретических результатов, связывающих свойства предобусловленной матрицы HA (характеризующих, так или иначе, ее «близость» к единичной матрице) со скоростью сходимости итераций метода сопряженных градиентов, однако практические вычисления достаточно часто преподносят разного рода неожиданности. Рассмотрим некоторые аспекты этой проблемы.

3.2.1. Эмпирические подходы. Не имея достаточной теоретической базы в виде строгих математических утверждений, вычислители-практики разработали некоторые разумные рецепты, позволяющие обеспечить «близость» матриц H^{-1} и A. Так, пусть в формуле из п. 3.1.1 для неполного треугольного разложения матрица E выбрана равной D, а матрица L — равной строго нижней треугольной части матрицы системы. Тогда, выдвигая условие совпадения диагональных частей матриц H^{-1} и A, получаем вариант неполного разложения Холесского, а заменяя его условием равенства векторов $H^{-1}u$ и Au при $u = (1 \ldots 1)^{\mathrm{T}}$, получаем вариант модифицированного разложения Холесского, в обоих случаях оказывается возможным примене-

ние приема Айзенштата. В случае, когда разрешение получающихся рекуррентных соотношений приводит к возникновению отрицательных элементов матрицы D, производится замена диагональных элементов матрицы A на бо́льшие значения (см., например, [Manteuffel, 1980]). В простых случаях эти методы дают неплохие результаты [Kershaw, 1978] и даже поддаются анализу на предмет асимптотической оценки числа итераций [20—22], однако для задач с симметричными положительно определенными матрицами общего вида, особенно при необходимости введения существенной диагональной модификации, эффективность снижается, и даже метод SSOR(ω), где $D = \frac{1}{\omega}\operatorname{diag} A$, может при удачном выборе ω оказаться лучше. Напомним, что распараллеливание таких предобусловливателей является затруднительным. Известны также многочисленные блочные варианты неполного треугольного разложения, анализ которых проведен для простых частных случаев (см., например, [9, 17, 22]).

3.2.2. Минимизация числа обусловленности предобусловленной матрицы. Общепринятым является критерий качества предобусловливания, описанный в § 3.4, когда в качестве меры «близости» H к A^{-1} принимается малость величины $\varkappa = \lambda_{\max}(HA)/\lambda_{\min}(HA)$, совпадающей в нашем случае со спектральным числом обусловленности матрицы HA. В основе этого подхода лежит хорошо известная оценка числа итераций k метода сопряженных градиентов, достаточного для уменьшения A^{-1}-нормы невязки r_k в ε^{-1} раз: $k < \frac{1}{2}\sqrt{\varkappa}\ln\frac{2}{\varepsilon}$ (эта оценка легко следует из теоремы П.3.4). Соответственно, практически единственным «легальным» методом исследования и обоснования способов предобусловливания считается оценка числа обусловленности $\varkappa$. Признавая несомненную силу такого подхода (так, с точки зрения получения рекордных асимптотик числа итераций для некоторых методов решения дискретного уравнения Пуассона на квадратной сетке величина $\varkappa$ остается незаменимым посредником), следует все же предостеречь от трактовки предобусловливания исключительно как средства минимизации $\varkappa$. Дело в том, что минимальность $\varkappa$ вовсе не является необходимым условием наиболее быстрой сходимости метода сопряженных градиентов (пожалуй, самым рельефным примером может служить оптимизация полиномиального предобусловливания [Johnson et al., 1983], и рекомендуемый в основном тексте выбор параметров α_i не случайно сделан с отступлением от принципа минимальности $\varkappa$). Заменой малости $\varkappa$ может служить разрежение спектра матрицы HA вблизи минимального

собственного значения (см., например, [Axelsson et al., 1986] и цитированную там литературу). Эффект разрежения спектра и предобусловленной матрицы в области наибольших собственных значений теоретически еще более выгоден, но практически может сойти на нет из-за влияния погрешностей округления, выражающегося в существенном увеличении числа итераций [21].

Таким образом, минимизация величины $\varkappa$, вообще говоря, не может являться главной и единственной целью предобусловливания. Необходимо учитывать более широкий спектр свойств матрицы HA; кроме того, построение удовлетворительных оценок обусловленности для реально решаемых задач и практически используемых способов предобусловливания, как правило, не представляется возможным. Сказанное выше хорошо согласуется, в частности, с замечаниями, сделанными в [23].

3.2.3. Альтернативные критерии качества предобусловливания. В работах [15, 16] в качестве меры близости матрицы HA к единичной предлагается выбирать величину $\gamma = \|I - HA\|_F^2$, то есть квадрат расстояния между этими матрицами в норме Фробениуса (которая равна сумме квадратов евклидовых норм всех строк матрицы $I - HA$). В работах [17, 18] рассмотрено обобщение этого подхода, где в качестве меры близости используется величина γ_W, равная сумме квадратов W-норм всех строк матрицы $I - HA$. Понятно, что если матрица H линейно зависит от параметров предобусловливания, то их значения, минимизирующие γ_W, нетрудно найти посредством решения вспомогательных систем линейных уравнений. Отметим, в частности, что построение полиномиального предобусловливания на основе минимизации γ, предложенное в [16], дает, как правило, хорошие результаты, близкие к лучшим из полученных в [Johnson et al., 1983], не требуя никакой информации о границах спектра матрицы A. Недостатки такого подхода заключаются, во-первых, в трудностях, возникающих при попытках построения оптимальных по γ_W симметричных положительно определенных предобусловливателей общего вида, и, во-вторых, в невозможности получения оценок числа итераций, имеющих какое-либо практическое значение, с использованием только величины γ. Можно, однако, получить (см. [24]) асимптотически неулучшаемую (при некоторых естественных предположениях) оценку

$$k = O\left(\frac{\gamma \ln \varkappa + \ln \varepsilon^{-1}}{\ln \ln \varkappa}\right),$$

дающую представление о сравнительном влиянии величин γ и $\varkappa$ на число итераций метода сопряженных градиентов.

Другая мера близости матрицы HA размера $n \times n$ к единичной, введенная в [19, 25], — величина $\beta = \frac{1}{n} \operatorname{tr} HA/(\det HA)^{1/n}$, где tr означает след матрицы, равный сумме всех ее диагональных элементов, а det означает определитель матрицы. При $H = G^{\mathrm{T}}G$, где G — треугольная матрица со строками, линейно зависящими от непересекающихся подмножеств параметров (см. п. 3.1.2 выше), можно легко решить задачу о минимизации β (см. [19, 25]). Справедлива также (см. [25]) асимптотически неулучшаемая оценка числа итераций метода сопряженных градиентов вида

$$k = O\left(n \ln \beta + \ln \frac{1}{\varepsilon}\right).$$

Численные эксперименты с такими предобусловливателями свидетельствуют об их высокой эффективности.

Следует признать, впрочем, что приведенные нестандартные оценки не позволяют получить столь же хорошие асимптотики числа итераций, как оценка через $\varkappa$, скажем, для дискретного уравнения Пуассона. С другой стороны, здесь нас интересует скорее согласованное изменение величины β (или γ) и действительного числа итераций, наблюдающееся на практике [19], а также сама возможность разработки вполне определенного подхода к предобусловливанию в случае произвольной симметричной положительно определенной матрицы.

4. Другие подходы к ускорению метода сопряженных градиентов. Обсудим теперь иные способы, принципиально отличные от введения предобусловливания, позволяющие сократить число операций и/или улучшить параллелизм метода сопряженных градиентов.

4.1. Перегруппировка операций скалярного произведения. Возможна модификация алгоритма, предусматривающая «совместное» параллельное вычисление на каждой итерации двух скалярных произведений вида $r_k^{\mathrm{T}}(Hr_k)$ и $(Hr_k)^{\mathrm{T}}(AHr_k)$, что позволяет ценой добавления одной векторной триады почти вдвое сократить соответствующие затраты на организацию межпроцессорных обменов. Такой алгоритм описан в [3], и там же указано дальнейшее развитие этого подхода, когда вычисляется только каждое s-приближение и, хотя число операций несколько увеличивается, затраты на инициализацию обменов при вычислении скалярных произведений уменьшаются в $2s$ раз по сравнению со стандартным алгоритмом. Вычислительная схема

метода имеет следующий вид:

$P_0 = 0;$
Для $k = 0, 1, \ldots$
$r_k = b - Ax_k,$
$Q_k = [Hr_k \; HAHr_k \; \cdots \; H(AH)^{s-1} r_k],$
$\mu_i = ((AH)^{\lceil i/2 \rceil} r_k)^{\mathrm{T}} H (AH)^{\lfloor i/2 \rfloor} r_k, \quad i = 0, 1, \ldots, 2s-1,$
находим за $O(s^3)$ операций α_k и β_k,
$P_{k+1} = Q_k + P_k \beta_k,$
$x_{k+1} = x_k + P_{k+1} \alpha_k.$

Здесь P_k, Q_k, α_k, β_k — матрицы размеров соответственно $n \times s$, $n \times s$, $s \times 1$ и $s \times s$. Заметим, что при использовании разреженного предобусловливателя H открываются дополнительные возможности при распараллеливании вычисления матриц Q_k (например, по аналогии с [14]).

4.2. Дефляционный метод сопряженных градиентов. Недавно был опубликован [26, 27] и опробован [28] алгоритм метода сопряженных градиентов, позволяющий получить улучшенные приближения за счет обеспечения ортогональности текущей невязки r_k не только ко всем предшествующим, как в обычном методе, но и к столбцам заранее заданной матрицы C размера $n \times n$. Алгоритм метода можно представить в следующем виде: если требуется решить систему $Ax = b$ с симметричной положительно определенной матрицей (возможно, симметрично предобусловленной), а $x_{\text{нач}}$ — заданное начальное приближение, то

$$x_0 = x_{\text{нач}} + C (C^{\mathrm{T}} AC)^{-1} C^{\mathrm{T}} (b - Ax_{\text{нач}}),$$
$$p_0 = r_0 = b - Ax_0,$$
Для $k = 0, 1, \ldots$
$$q_k = Ap_k,$$
$$t_k = (C^{\mathrm{T}} AC)^{-1} C^{\mathrm{T}} q_k,$$
$$\alpha_k = r_k^{\mathrm{T}} r_k / q_k^{\mathrm{T}} (p_k - Ct_k),$$
$$x_{k+1} = x_k + (p_k - Ct_k) \alpha_k,$$
$$r_{k+1} = r_k - (q_k - (AC) t_k) \alpha_k,$$
$$\beta_k = r_{k+1}^{\mathrm{T}} r_{k+1} / r_k^{\mathrm{T}} r_k,$$
$$p_{k+1} = r_{k+1} + p_k \beta_k.$$

Таким образом, если матрица AC вычислена заранее, то на каждой итерации по-прежнему достаточно умножать матрицу

A на вектор только один раз. Основной проблемой здесь является выбор матрицы C. В [26] для решения сеточных задач предлагается использовать декомпозицию области на m подобластей (без разделителей; при этом квадрат, например, разбивается на квадратные подобласти) и выбирать в качестве i-го столбца матрицы C вектор с компонентами, равными единице в узлах i-й подобласти и нулю в противном случае. Там же показано, что с увеличением числа подобластей m (то есть с уменьшением их размеров) скорость сходимости такого метода увеличивается; в этом случае, однако, возрастают и трудности, связанные с обращением матрицы $C^{\mathrm{T}}AC$, оказывающейся близкой по своим свойствам к исходной матрице A. Для параллельных вычислений, вероятно, целесообразным окажется использование в этой конструкции разбиения, включающего узлы-разделители, не принадлежащие ни одной из подобластей (см, обсуждение рис. 2.3.13). В этом случае матрица $C^{\mathrm{T}}AC$ будет иметь диагональный вид и вычисления резко упростятся. Здесь, конечно, существует некоторый оптимальный размер подобластей с точки зрения скорости сходимости итераций.

4.3. Блочный метод сопряженных градиентов. Алгоритм этого метода, упоминаемого в дополнении к § 3.3, имеет в точности тот же вид, что и обычный алгоритм (см. п. 3), но при b, x_k, r_k и p_k, заданных в виде матриц $n \times m$, и α_k и β_k, заданных в виде матриц $m \times m$. Соответственно возможна двоякая интерпретация этого алгоритма: можно либо одновременно решать m систем с одной и той же матрицей (то есть искать решение задачи $Ax = b$, где x имеет размер $n \times m$), либо, решая одну систему, выбрать специальным образом дополнительные $m - 1$ векторов, чтобы достроить матрицу r_0 (здесь просматривается некоторая аналогия с принципом, на котором построен дефляционный метод сопряженных градиентов). Для этого метода характерна более высокая скорость сходимости, чем для обычного метода сопряженных градиентов; он обладает также более высоким параллелизмом (см. [O'Leary, 1987] и цитированную там литературу, содержащую, в частности, описание модификации этого алгоритма, где используется QR-разложение матриц r_k для предотвращения обрывания итерационного процесса при потере ими полного ранга).

Литература

1. Зенкевич О, Морган К Конечные элементы и аппроксимация. — М : Мир, 1986.
2. Бенерджи П, Баттерфилд Р. Методы граничных элементов в прикладных науках. — М.: Мир, 1984.

3. Chronopoulos A A class of parallel iterative methods implemented on multiprocessors — Report No UIUCDCS-R-86-1267, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1986.
4 Radicati di Brozolo G, Robert Y Parallel conjugate gradient-like algorithms for solving sparse nonsymmetric linear systems on a vector multiprocessor. — Parallel Comput, 1989, v 11, p 223—239.
5. Джордж А, Лю Дж Численное решение больших разреженных систем уравнений — М · Мир, 1984
6 Lipton R ,J, Rose D J, Tarjan R E Generalized nested dissection. — SIAM J. Numer Anal, 1979, v 16, p 346—358
7. Baxter D, Saltz J, Schultz M, Eisenstat S, Crowley K An experimental study of methods for parallel preconditioned Krylov methods — Res. Report RR-629, Department of Computer Science, Yale University, 1988
8 Berryman H, Saltz J., Gropp W. Krylov methods preconditioned with incompletely factored matrices on the CM-2 — Tech. Report No. 685, Department of Computer Science, Yale University, 1989
9 Кучеров А. Б, Николаев Е С Параллельные и конвейерные алгоритмы попеременно-треугольного итерационного метода В кн.. Разностные методы математической физики/Под ред Е С Николаева — М · Изд-во Моск ун-та, 1984, с 66—83
10. Кучеров А Б, Николаев Е С Параллельные алгоритмы итерационных методов с факторизованным оператором для решения эллиптических краевых задач — Дифференц уравнения, 1984, т 20, № 7, с 1230—1237
11. Kolotilina L Yu., Yeremin A Yu Block SSOR preconditionings for high order 3D FE systems — BIT, 1989, v 29, N 4, p. 805—823.
12. Wille S. O A local preconditioner for symmetric and non symmetric equations for refined finite element meshes Numer Meth Laminar and Turbulent Flow Proc 5th Int Conf, Montreal, July 6—10 1987, v 5, pt. 1, Swansea, 1987, p 57—68.
13 Еремин А Ю, Капорин И Е Частичное исключение как способ переобусловливания — Препр ОВМ АН СССР № 216, М., 1988
14. Еремин А Ю, Капорин И Е Реализация явных чебышёвских методов при решении задач большой размерности В кн Многопроцессорные вычислительные структуры - Таганрог: ТРТИ, 1985, вып 7 (XVI), с 43—46.
15. Benson M, Krettmann J, Wright M Parallel algorithms for the solution of certain large sparse linear systems. — Int J Comput. Math., 1984, v. 16, p 245—260.
16. Еремин А. Ю., Капорин И Е Спектральная оптимизация явных итерационных методов I В кн · Численные методы и вопросы организации вычислений, вып 7. Зап научн семин ЛОМИ АН СССР, 1984, т. 139, с. 51—60
17. Еремин А Ю, Колотилина Л Ю Об одном семействе двухуровневых переобусловливаний типа неполной блочной факторизации — Препр. ОВМ АН СССР № 108, М, 1985
18. Колотилина Л Ю Об одном семействе явных переобусловливаний систем линейных алгебраических уравнений с разреженными матрицами. — Препр ЛОМИ АН СССР № Р-8-86, Л, 1986.
19. Капорин И. Е О предобусловливании метода сопряженных градиентов при решении дискретных аналогов дифференциальных задач — Дифференц. уравнения, 1990, т. 26, № 7, с 1225—1236.
20. Кучеров А. Б, Макаров М. М Метод приближенной факторизации для решения разностных смешанных эллиптических краевых задач. — В кн.: Разностные методы математической физики/Под ред. Е. С. Николаева. — М.: Изд-во Моск. ун-та, 1984, с. 54—65.

21. Van der Vorst H A. High performance preconditioning — SIAM J. Sci. Statist, Comput, 1989, v 10, p. 1174—1185
22 Axelsson O., Eijkhout V. Vectorizable preconditioners for elliptic difference equations in three space dimensions — J. Comput. Appl. Math, 1989, v 27, p 299—321.
23 Saad Y., Schultz M H Parallel implementations of preconditioned conjugate gradient methods. — Res Report RR-425, Department of Computer Science, Yale University, 1985.
24 Капорин И Е Оценки числа итераций методов типа сопряженных градиентов. – В кн Актуальные вопросы прикладной математики и математического обеспечения ЭВМ/Под ред Ю В Шестопалова -- М · Изд-во Моск ун-та, 1990, с 112 -113
25 Капорин И. Е. Альтернативный подход к оценке числа итераций метода сопряженных градиентов – В кн · Численные методы и программное обеспечение/Под ред. Ю. А Кузнецова — М. ОВМ АН СССР, 1990, с. 55—72
26. Nicolaides R. A Deflation of the conjugate gradients with applications to boundary value problems — SIAM J. Numer Anal., 1987, v 10, N 2 p. 355—365
27. Кучеров А Б. Корниенко М. Е. Проекционные методы неполного разложения – сопряженных градиентов для решения разностных эллиптических уравнений — В кн.: Разреженные матрицы Численные методы и алгоритмы/Под ред. Е. С. Николаева и А. Б. Кучерова. — М.: Изд-во Моск. ун-та, 1988, с. 80—98.
28. Mansfield L. On the use of deflation to improve the convergence of conjugate gradient iteration. — Commun. Appl. Numer. Methods, 1988, v. 4, N 2, p. 151—156

Предметный указатель

Оглавление

Уважаемый читатель!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И-110, ГСП, 1-й Рижский пер., д. 2, издательство «Мир».

Научное издание

Джеймс Ортега

ВВЕДЕНИЕ В ПАРАЛЛЕЛЬНЫЕ И ВЕКТОРНЫЕ МЕТОДЫ РЕШЕНИЯ ЛИНЕЙНЫХ СИСТЕМ

Заведующий редакцией академик В. И. Арнольд
Зам. зав. редакцией А. С. Попов
Научный редактор О. Р. Чуян
Мл. науч. редактор Т. Ю. Дехтярева
Художник В. С. Потапов
Художественный редактор В. И. Шаповалов
Технический редактор Е. С. Потапенкова

ИБ № 7543

Сдано в набор 26.04.90. Подписано к печати 23.06.91. Формат 60 × 88 1/16. Бумага книжно-журн. Печать офсетная. Гарнитура литературная. Объем 11,50 бум. л Усл. печ л 23,00. Усл. кр-отт. 23,00. Уч.-изд. л. 21,09. Изд. № 1/7662. Тираж 12 000 экз. Заказ № 581. Цена 5 р. 20 к

Издательство «Мир»

В/О «Совэкспорткнига» Государственного комитета СССР по делам издательств, полиграфии и книжной торговли. 129820, ГСП, Москва, 1-й Рижский пер., 2.

Набрано в Ленинградской типографии № 2 головном предприятии ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им Евгении Соколовой Государственного комитета СССР по печати, 198052, г. Ленинград, Л-52, Измайловский проспект, 29. Отпечатано в Ленинградской типографии № 4 Государственного комитета СССР по печати 191126, Ленинград, Социалистическая ул., 14.