Нижегородский государственный университет им. Н.И. Лобачевского
Факультет вычислительной математики и кибернетики

Образовательный комплекс «Методы параллельных вычислений»

Методическое пособие

Баркалов К.А.

При поддержке компании Intel

Нижний Новгород
2011

УДК 519.6

ББК В22.1

Баркалов К.А. Методы параллельных вычислений. Н. Новгород: Изд-во Нижегородского госуниверситета им. Н.И. Лобачевского, 2011.

В пособии рассматриваются типовые численные алгоритмы из различных разделов вычислительной математики: методы решения систем линейных алгебраических уравнений (с плотными и разреженными матрицами), решение дифференциальных уравнений в частных производных, методы Монте-Карло. Для каждого рассматриваемого алгоритма приводится его математическое описание, излагается последовательная версия, рассматриваются способы возможного распараллеливания в системах с общей памятью. Проводится сравнение разработанных алгоритмов с известными высокопроизводительными реализациями (на примере Intel MKL)

Пособие разработано на базе лаборатории «Информационные технологии» ITLab факультета ВМК ННГУ при поддержке компании Intel и Совета по грантам Президента Российской Федерации (грант № НШ-64729.2010.9).

Для студентов и магистрантов, обучающихся на факультетах физико-математического профиля.

ББК В22.1

Содержание

Введение

Традиционно *численные методы* рассматривают как раздел вычислительной математики, связанный с разработкой алгоритмов решения типовых задач, возникающих при исследованиях математических моделей объектов и процессов реального мира, и их реализации в виде программ на алгоритмическом языке для той или иной вычислительной системы. В качестве примера таких типовых задач можно назвать задачи алгебры: здесь большое значение имеют численные методы решения систем линейных алгебраических уравнений, обращение матриц, нахождение собственных значений матриц. Другие примеры – численные методы дифференцирования и интегрирования функций одного или нескольких переменных; численные методы решения обыкновенных дифференциальных уравнений и уравнений в частных производных.

В настоящее время основной тенденцией развития вычислительной техники является параллельность: любой современный компьютер содержит несколько вычислительных ядер на центральном процессоре, и несколько десятков ядер – на графическом. Развитие техники определяет и развитие численных алгоритмов: все большее внимание уделяется параллельным численным методам, любой алгоритм сейчас рассматривается сквозь призму его возможного распараллеливания.

Целью данного пособия является изложение некоторых известных численных алгоритмов и рассмотрение круга вопросов, связанных с их распараллеливанием.

Пособие построено следующим образом.

Первая часть посвящена рассмотрению прямых методов решения систем линейных алгебраических уравнений с матрицами как общего, так и специального вида: метод исключения Гаусса, разложение Холецкого, метод прогонки. Изложены классические варианты алгоритмов, показана их недостаточная эффективность при использовании современных вычислительных архитектур. Последовательно проводится идея блочной обработки данных.

Во второй части рассмотрены итерационные методы решения систем линейных уравнений: методы простой итерации и верхней релаксации, метод сопряженных градиентов. Рассмотрены способы их распараллеливания,

приведены теоретические и экспериментельные оценки ускорения, достигаемого за счет введения паралеллизма.

Третья часть посвящена задачам разреженной алгебры. Дан краткий обзор структур хранения разреженных матриц, рассмотрены типовые проблемы, возникающие при реализации основных операций над разреженными матрицами. В качестве примера вычислительного алгоритма рассмотрено разложение Холецкого, показана проблема роста коэффициента заполнения матрицы при разложении, изложены алгоритмы переупорядочивания матрицы, снижающие заполнение результирующей матрицы.

Четвертая часть затрагивает вопросы параллельного решения дифференциальных уравнений в частных производных. Рассмотрены типовые уравнения в частных производных, для них построены разностные схемы, приведены последовательные и параллельные алгоритмы их решения.

Заключительная часть посвящена численным методам Монте-Карло. Здесь читатели могут познакомиться с проблемами, возникающими при использовании случайных чисел при решении задач методами Монте-Карло. Рассмотрены различные способы генерации случайных чисел, как последовательные, так и параллельные; обсуждаются вопросы безызбыточного распараллеливания метода.

В каждой главе приводятся результаты вычислительных экспериментов, которые были получены с использованием следующей инфраструктуры.

Процессор	2x Intel Xeon E5520 (4 cores, 2.27 GHz)
Память	16 Gb
Операционная система	Microsoft Windows 7
Среда разработки	Microsoft Visual Studio 8.0
Компилятор, профилировщик, отладчик	Intel Parallel Studio SP1
Математическая библиотека	Intel MKL v. 10.2.5.035

Автор выражает благодарность творческому коллективу сотрудников и студентов факультета ВМК ННГУ им. Н.И. Лобачевского, принимавших участие в реализации алгоритмов и проведении вычислительных экспериментов. Результаты, приведенные в пунктах 1.2, 1.3, 2.2 и 4.3 пособия, получны Маловой А.Ю, в пунктам 1.1,2.3, 4.1 – Сафоновой Я.Ю, в пункте 3.3 – Скляровым О.А, в пункте 4.2 – Кустиковой В.Д., в главе 5 – Мееровым И.Б. и Сысоевым А.В.

Также автор выражает признательность Гергелю Виктору Павловичу за полезные комментарии и обсуждения в процессе разработки данного пособия.

1. Прямые методы решения СЛАУ

Методы решения систем линейных алгебраических уравнений (СЛАУ) относятся к численным методам алгебры. При формальном подходе решение подобных задач не встречает затруднений: решение системы можно найти, раскрыв определители в формуле Крамера. Однако при непосредственном раскрытии определителей решение системы с n неизвестными требует $O(n!)$ арифметических операций; уже при n порядка 20 такое число операций недоступно для современных компьютеров. При сколько-нибудь больших n применение методов с таким порядком числа операций будет невозможно и в обозримом будущем. Другой причиной, по которой этот классический способ неприменим даже при малых n, является сильное влияние на окончательный результат округлений при вычислениях.

Методы решения алгебраических задач можно разделить на точные и итерационные. Классы задач, для решения которых обычно применяют методы этих групп, можно условно назвать соответственно классами задач со средним и большим числом неизвестных. Изменение объема и структуры памяти вычислительных систем, увеличение их быстродействия и развитие численных методов приводят к смещению границ применения методов в сторону систем более высоких порядков.

Например, в 80-х годах прошлого века точные методы применялись для решения систем до порядка 10^4, итерационные – до порядка 10^7, в 90-х – до порядков 10^5 и 10^8 соответственно. Современные суперкомпьютеры способны использовать точные методы при решении еще больших систем.

Мы будем рассматривать систему из n линейных алгебраических уравнений вида

$$\begin{aligned} a_{11}x_1 &+ a_{12}x_2 &+ \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 &+ a_{21}x_2 &+ \ldots + a_{2n}x_n &= b_2 \\ &\ldots & & \\ a_{n1}x_1 &+ a_{n2}x_2 &+ \ldots + a_{nn}x_n &= b_n \end{aligned} \qquad (1.1)$$

В матричном виде система может быть представлена как

$$Ax=b, \qquad (1.2)$$

где $A=(a_{ij})$ есть вещественная матрица размера $n\times n$; b и x – вектора из n элементов.

Под задачей решения системы линейных уравнений для заданных матрицы A и вектора b мы будем считать нахождение значения вектора неизвестных x, при котором выполняются все уравнения системы.

1.1. Метод прогонки

Одним из частных (но, тем не менее, часто встречающихся) видов системы (1.2) является система

$$Ax=f,$$

с ленточной матрицей A. Матрица A называется *ленточной*, когда все ее ненулевые элементы находятся вблизи главной диагонали, т.е. $a_{ij}=0$, если $|i-j|>l$, где $l<n$. Число l называется *шириной ленты*. Примером является трехдиагональная матрица (при $l=1$) вида:

$$A=\begin{bmatrix} c_1 & b_1 & 0 & \dots & 0 \\ a_2 & c_2 & b_2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & a_{n-1} & c_{n-1} & b_{n-1} \\ 0 & \dots & 0 & a_n & c_n \end{bmatrix}$$

Матрицы такого вида возникают, например, при решении задачи сплайн-интерполяции [1].

Рассмотрим *метод прогонки*, применимый для решения систем с трехдиагональной матрицей. Предположим, что имеет место соотношение

$$x_i=\alpha_{i+1}x_{i+1}+\beta_{i+1} \tag{1.3}$$

с неопределенными коэффициентами α_{i+1} и β_{i+1}, и подставим выражение $x_{i-1}=\alpha_i x_i+\beta_i$ в i-е уравнение системы:

$$(\alpha_i a_i+c_i)x_i+b_i x_{i+1}=f_i-a_i\beta_i.$$

Сравнивая полученное выражение с (1.3), находим

$$\begin{aligned} \alpha_{i+1} &= \frac{-b_i}{a_i\alpha_i+c_i},\ i=2,\dots,n-1, \\ \beta_{i+1} &= \frac{f_i-a_i\beta_i}{a_i\alpha_i+c_i},\ i=2,\dots,n-1. \end{aligned} \tag{1.4}$$

Из первого уравнения системы

$$c_1x_1+b_1x_2=f_1$$

находим

$$\alpha_2=-b_1/c_1,\ \beta_2=f_1/c_1.$$

Зная α_2,β_2 и переходя от i к i+1 в формулах (1.4), определим α_i,β_i для всех i=3,…,n.

Определим x_n из последнего уравнения системы и условия (1.3) при i=n–1.

$$\begin{aligned} x_{n-1}&=\alpha_n x_n+\beta_n \\ a_n x_{n-1}+c_n x_n&=f_n \end{aligned} \qquad (1.5)$$

Решив систему из двух уравнений с двумя неизвестными, находим

$$x_n=\frac{f_n-a_n\beta_n}{a_n\alpha_n+c_n}.$$

После того, как значение x_n найдено, определяем все остальные значения x_i в обратном порядке, используя формулу (1.3). Соберем теперь все формулы прогонки и запишем их в порядке применения.

Прямой ход:

$$\begin{aligned} &\alpha_2=-b_1/c_1,\ \alpha_{i+1}=\frac{-b_i}{a_i\alpha_i+c_i},\ i=2,\ldots,n-1, \\ &\beta_2=f_1/c_1,\ \beta_{i+1}=\frac{f_i-a_i\beta_i}{a_i\alpha_i+c_i},\ i=2,\ldots,n-1. \end{aligned} \qquad (1.6)$$

Обратный ход:

$$x_n=\frac{f_n-a_n\beta_n}{a_n\alpha_n+c_n},\ x_i=\alpha_{i+1}x_{i+1}+\beta_{i+1},\ i=n-1,\ldots,1. \qquad (1.7)$$

Известно [4], что для вычислительной устойчивости метода прогонки необходимо выполнение условия *диагонального преобладания*

$$|c_1|\ge|b_1|,\ |c_n|\ge|a_n|,$$

$$|c_i|>|a_i|+|b_i|,\ i=2,\ldots,n-1.$$

Оценим трудоемкость метода прогонки. При выполнении прямого хода по формулам (1.6) потребуется 8(n–2)+2 операций. Для выполнения обратного хода по формулам (1.7) потребуется 2(n–1)+5 операций. Таким образом, общее число операций можно оценить величиной

$$10n+O(1), \tag{1.8}$$

а время решения системы методом прогонки при больших n будет определяться как

$$T_1 = 10n\tau,$$

где τ – время выполнения одной операции.

1.1.1. Метод встречной прогонки и его распараллеливание

Рассмотренный в предыдущем пункте метод прогонки, определяемый соотношениями (1.6) и (1.7), при котором определение x_i происходит последовательно справа налево, называют *правой прогонкой*. Аналогично выписываются формулы *левой прогонки*.

Прямой ход:

$$\begin{aligned} &\xi_n=-a_n/c_n,\ \xi_i = \frac{-a_i}{c_i + b_i\xi_{i+1}},\ i=n-1,\ldots,2;\\ &\eta_n=f_n/c_n,\ \eta_i = \frac{f_i - b_i\eta_{i+1}}{c_i + b_i\xi_{i+1}},\ i=n-1,\ldots,2. \end{aligned} \tag{1.9}$$

Обратный ход:

$$x_1 = \frac{f_1 - b_1\eta_2}{b_1\xi_2 + c_1},\ x_{i+1}=\xi_{i+1}x_i+\eta_{i+1},\ i=1,\ldots,n-1. \tag{1.10}$$

В самом деле, предполагая, что $x_{i+1}=\xi_{i+1}x_i+\eta_{i+1}$, исключим из i-го уравнения системы переменную x_{i+1}, получим

$$a_ix_{i-1}+c_ix_i+b_i(\xi_{i+1}x_i+\eta_{i+1})=f_i,$$

или

$$x_i = \frac{-a_i}{c_i + b_i\xi_{i+1}}x_{i-1} + \frac{f_i - b_i\eta_{i+1}}{c_i + b_i\xi_{i+1}}.$$

Сравнивая с формулой $x_i=\xi_ix_{i-1}+\eta_i$, получим расчетные формулы (1.9). Значение x_1 находим из первого уравнения и условия $x_2=\xi_2x_1+\eta_2$, затем, используя условие $x_i=\xi_ix_{i-1}+\eta_i$ и известные коэффициенты ξ_i, η_i, можно найти все остальные значения неизвестных.

Нетрудно видеть, что трудоемкость левой прогонки составляет также $10n+O(1)$.

Комбинация левой и правой прогонок дает метод *встречной прогонки*, который допускает распараллеливание на два потока. Разделим систему между двумя потоками – первый будет оперировать уравнениями с номерами $1 \le i \le p$, второй – уравнениями $p \le i \le n$, где $p = \lceil n/2 \rceil$.

При параллельном решении системы в первом потоке по формулам (1.4) вычисляются прогоночные коэффициенты α_i, β_i, при $1 \le i \le p$, а во втором потоке по формулам (1.9) находятся ξ_i, η_i, при $p \le i \le n$. При $i=p$ проводится сопряжение решений в форме (1.7) и (1.10): находим значение x_p из системы

$$\begin{cases} x_p = \alpha_{p+1} x_{p+1} + \beta_{p+1} \\ x_{p+1} = \xi_{p+1} x_p + \eta_{p+1} \end{cases}.$$

Найдя указанное значение, в первом потоке можно по формуле (1.7) найти все x_i, при $1 \le i < p$, а во втором – по формуле (1.10) – все x_i, при $p < i \le n$.

Трудоемкость метода параллельной встречной прогонки можно оценить как

$$T_2 = 5n\tau + \delta ,$$

где δ – время, необходимое на организацию и закрытие параллельной секции. Следует отметить, что расчеты и при прямом, и при обратном ходе производятся независимо, теоретическое ускорение здесь должно быть равно двум.

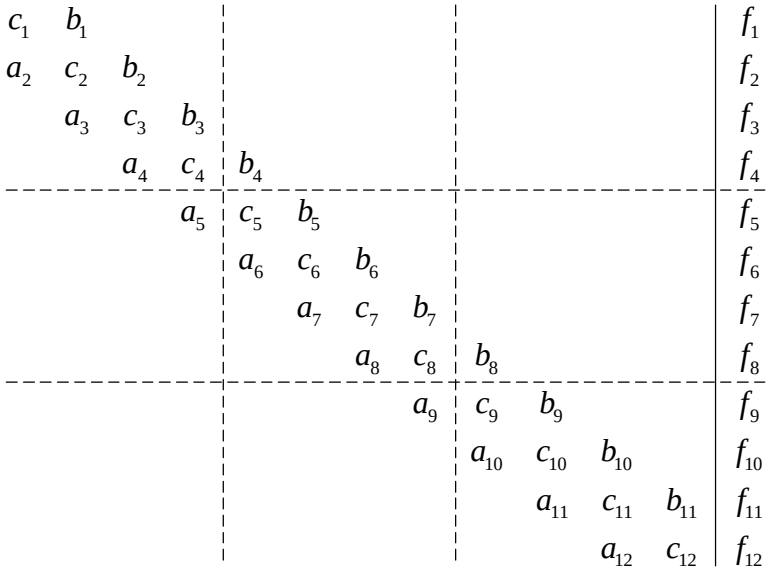
1.1.2. Параллельный вариант метода прогонки

Рассмотрим теперь схему распараллеливания метода прогонки при использовании p потоков. Пусть нужно решить трехдиагональную систему линейных уравнений

$$a_i x_{i-1} + c_i x_i + b_i x_{i+1} = f_i ,\ i=1,\ldots,n,\ x_0 = x_{n+1} = 0, \qquad (1.11)$$

с использованием p параллельных потоков.

Применим блочный подход к разделению данных: пусть каждый поток обрабатывает $m = \lfloor n / p \rfloor$ строк матрицы A, т.е. k-й поток обрабатывает строки с номерами $1+(k-1)m \le i \le km$. Для простоты изложения мы предполагаем, что число уравнений в системе кратно числу потоков, в общем случае изменится только число уравнений в последнем потоке. Ниже представлено разделение данных для трех потоков в случае системы из 12 уравнений.



В пределах полосы матрицы, обрабатываемой k-м потоком, можно организовать исключение поддиагональных элементов матрицы (прямой ход метода). Для этого осуществляется вычитание строки i, умноженной на константу a_{i+1}/c_i, из строки i+1 с тем, чтобы результирующий коэффициент при неизвестной x_i в (i+1)-й строке оказался нулевым.

Если исключение первым потоком поддиагональных переменных не добавит в матрицу новых коэффициентов, то исключение поддиагональных элементов в остальных потоках приведет к возникновению столбца отличных от нуля коэффициентов: во всех блоках (кроме первого) число ненулевых элементов в строке не изменится, но изменится структура уравнений. Модификации также подвергнутся элементы вектора правой части. Матрица (1.12) иллюстрирует данный процесс, чертой сверху отмечены элементы, которые будут модифицированы.

$$
\left(\begin{array}{cccccccccccc|c}
c_1 & b_1 & & & & & & & & & & & f_1 \\
 & \bar{c}_2 & b_2 & & & & & & & & & & \bar{f}_2 \\
 & & \bar{c}_3 & b_3 & & & & & & & & & \bar{f}_3 \\
 & & & \bar{c}_4 & b_4 & & & & & & & & \bar{f}_4 \\
\hline
 & & & a_5 & c_5 & b_5 & & & & & & & f_5 \\
 & & & d_6 & & \bar{c}_6 & b_6 & & & & & & \bar{f}_6 \\
 & & & d_7 & & & \bar{c}_7 & b_7 & & & & & \bar{f}_7 \\
 & & & d_8 & & & & \bar{c}_8 & b_8 & & & & \bar{f}_8 \\
\hline
 & & & & & & & a_9 & c_9 & b_9 & & & f_9 \\
 & & & & & & & d_{10} & & \bar{c}_{10} & b_{10} & & \bar{f}_{10} \\
 & & & & & & & d_{11} & & & \bar{c}_{11} & b_{11} & \bar{f}_{11} \\
 & & & & & & & d_{12} & & & & \bar{c}_{12} & \bar{f}_{12}
\end{array}\right) \qquad (1.12)
$$

Затем выполняется обратный ход алгоритма – каждый поток исключает наддиагональные элементы, начиная с последнего.

$$
\left(\begin{array}{cccccccccccc|c}
c_1 & & & g_1 & & & & & & & & & \bar{f}_1 \\
 & \bar{c}_2 & & g_2 & & & & & & & & & \bar{f}_2 \\
 & & \bar{c}_3 & g_3 & & & & & & & & & \bar{f}_3 \\
 & & & \bar{c}_4 & b_4 & & & & & & & & \bar{f}_4 \\
\hline
 & & & \bar{a}_5 & c_5 & & & & g_5 & & & & \bar{f}_5 \\
 & & & \bar{d}_6 & & \bar{c}_6 & & & g_6 & & & & \bar{f}_6 \\
 & & & \bar{d}_7 & & & \bar{c}_7 & & g_7 & & & & \bar{f}_7 \\
 & & & d_8 & & & & \bar{c}_8 & b_8 & & & & \bar{f}_8 \\
\hline
 & & & & & & & \bar{a}_9 & c_9 & & & & \bar{f}_9 \\
 & & & & & & & \bar{d}_{10} & & \bar{c}_{10} & & & \bar{f}_{10} \\
 & & & & & & & \bar{d}_{11} & & & \bar{c}_{11} & & \bar{f}_{11} \\
 & & & & & & & d_{12} & & & & \bar{c}_{12} & \bar{f}_{12}
\end{array}\right)
$$

После выполнения обратного хода матрица стала блочной. Исключим из нее внутренние строки каждой полосы, в результате получим систему уравнений относительно части исходный неизвестных, частный вид которой представлен ниже.

$$\left(\begin{array}{ccccccc|c}
c_1 & g_1 & & & & & & \bar{f}_1 \\
 & \bar{c}_4 & b_4 & & & & & \bar{f}_4 \\
 & \bar{a}_5 & c_5 & g_5 & & & & \bar{f}_5 \\
 & & d_8 & \bar{c}_8 & b_8 & & & \bar{f}_8 \\
 & & & \bar{a}_9 & c_9 & & & \bar{f}_9 \\
 & & & & d_{12} & \bar{c}_{12} & & \bar{f}_{12}
\end{array}\right)$$

Данная система будет содержать $2p$ уравнений, и будет трехдиагональной. Ее можно решить последовательным методом прогонки. После того, как эта система будет решена, станут известны значения неизвестных на границах полос разделения данных. Далее можно за один проход найти значения внутренних переменных.

Рассмотренный способ распараллеливания уже дает хорошие результаты, но можно использовать лучшую стратегию исключения неизвестных. Прямой ход нового алгоритма будет таким же, а во время обратного хода каждый поток исключает наддиагональные элементы, начиная со своего предпоследнего, и заканчивая последним для предыдущего потока. Матрица (1.13) иллюстрирует данный процесс.

$$\left(\begin{array}{ccccccccccc|c}
c_1 & & & g_1 & & & & & & & & \bar{f}_1 \\
 & \bar{c}_2 & & g_2 & & & & & & & & \bar{f}_2 \\
 & & \bar{c}_3 & b_3 & & & & & & & & \bar{f}_3 \\
 & & & \bar{c}_4 & & & g_4 & & & & & \bar{f}_4 \\
\hdashline
 & & & \bar{a}_5 & c_5 & & g_5 & & & & & \bar{f}_5 \\
 & & & \bar{d}_6 & & \bar{c}_6 & g_6 & & & & & \bar{f}_6 \\
 & & & d_7 & & & \bar{c}_7 & b_7 & & & & \bar{f}_7 \\
 & & & d_8 & & & & \bar{c}_8 & & & g_8 & \bar{f}_8 \\
\hdashline
 & & & & & & & \bar{a}_9 & c_9 & & g_9 & \bar{f}_9 \\
 & & & & & & & \bar{d}_{10} & & \bar{c}_{10} & g_{10} & \bar{f}_{10} \\
 & & & & & & & d_{11} & & & b_{11} & \bar{f}_{11} \\
 & & & & & & & d_{12} & & & \bar{c}_{12} & \bar{f}_{12}
\end{array}\right) \qquad (1.13)$$

Изменение порядка исключения переменных в обратном ходе алгоритма приводит к тому, что можно сформировать вспомогательную задачу меньшего размера. Исключим из матрицы все строки каждой полосы, кроме

последней, в результате получим систему уравнения относительно части исходный неизвестных, частный вид которой представлен ниже.

$$\left|\begin{array}{ccc|c} \bar{c}_4 & g_4 & & \bar{f}_4 \\ d_8 & \bar{c}_8 & g_8 & \bar{f}_8 \\ & d_{12} & \bar{c}_{12} & \bar{f}_{12} \end{array}\right|$$

Даная система будет содержать всего p уравнений, и также будет трехдиагональной. Ее можно решить последовательным методом прогонки (так как для систем с общей памятью число потоков p будет не слишком велико, применять для решения вспомогательной системы даже параллельный метод встречной прогонки нецелесообразно). После того, как эта система будет решена, станут известны значения неизвестных на нижних границах полос разделения данных. Далее можно за один проход найти значения внутренних переменных в каждом потоке.

Оценим трудоемкость рассмотренного параллельного варианта метода прогонки. В соответствии с введенными ранее обозначениями n есть порядок решаемой системы линейных уравнений, а p, $p<n$, обозначает число потоков. Тем самым, матрица коэффициентов A имеет размер $n \times n$ и, соответственно, $m=n/p$ есть размер полосы матрицы A на каждом процессоре.

При выполнении прямого хода алгоритма на каждой итерации каждый процессор должен осуществить исключение в пределах своей полосы поддиагональных элементов (что требует $8(m-1)$ операций) и наддиагональных элементов (что требует $7m$ операций).

Затем следует произвести сборку вспомогательной трехдиагональной системы уравнений в одном потоке, и осуществить ее решение методом прогонки. В соответствии с оценкой (1.8) затраты на выполнение этого чисто последовательного этапа составят порядка $10p$ операций.

На следующем этапе алгоритма каждый процессор выполняет обратный ход алгоритма, который потребует $5(m-1)$ операций. Таким образом, общую трудоемкость параллельного метода прогонки можно оценить как

$$T_p = 20m + 10p\,. \tag{1.14}$$

Как результат выполненного анализа, показатели ускорения и эффективности параллельного варианта метода прогонки могут быть определены при помощи соотношений следующего вида:

$$S_p = \frac{T_1}{T_p} = \frac{10n}{20\dfrac{n}{p} + 10p} = p\frac{10n}{20n + p^2}; \tag{1.15}$$

$$E_p = \frac{S_p}{p} = \frac{10n}{20n + p^2}.$$

Из приведенных соотношений видно, что в случае решения системы уравнений с большим числом неизвестных, при котором $p << n$, показатели ускорения и эффективности будут определяться как

$$S_p \approx \frac{p}{2},\ E_p \approx 0.5. \tag{1.16}$$

1.1.3. Результаты вычислительных экспериментов

Вычислительные эксперименты для оценки эффективности параллельного варианта метода прогонки для решения трехдиагональных систем линейных уравнений проводились на аппаратуре, технические характеристики которой указаны во введении. Также следует отметить, что для простоты написания параллельных программ нами рассматривались задачи размера, кратного 8, чтобы размер блоков был одинаков для всех потоков. С целью формирования матрицы с диагональным преобладанием элементы на побочных диагоналях матрицы генерировались в диапазоне от 0 до 100, а элемент на главной диагонали был равен удвоенной сумме элементов в строке.

Сначала приведем результаты сравнения реализованной нами правой и левой прогонки со специальной функцией библиотеки Intel MKL, решающей трехдиагональные системы с диагональным преобладанием. Результаты, отражающие зависимость времени решения задачи *T* от ее размера *N*, приведены на рис. 1.1.

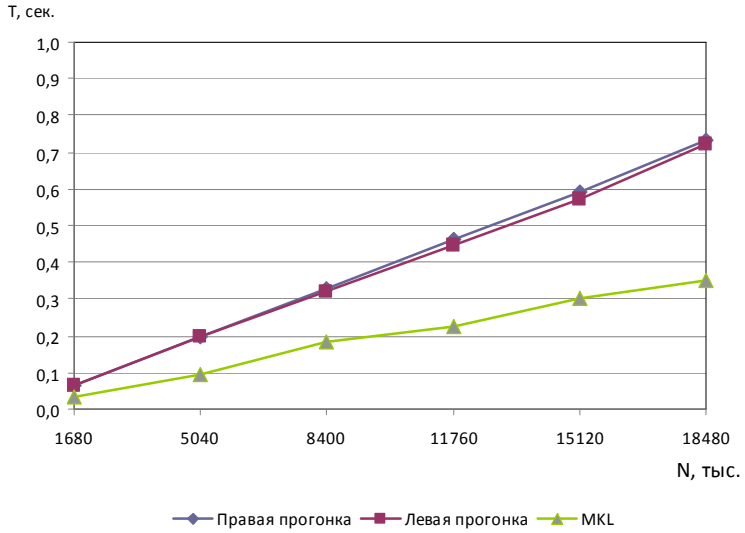


Рис. 1.1. Сравнение с библиотекой MKL

Результаты экспериментов демонстрируют двукратное отставание от библиотеки Intel MKL по времени, что является неплохим показателем.

Далее рассмотрим эффект, который дает использование встречной прогонки в двух потоках: проведем сравнение параллельной встречной прогонки с правой и левой прогонками. Результаты, отражающие зависимость ускорения по отношению к правой и левой прогонкам от размера задачи N приведены на рис. 1.2.

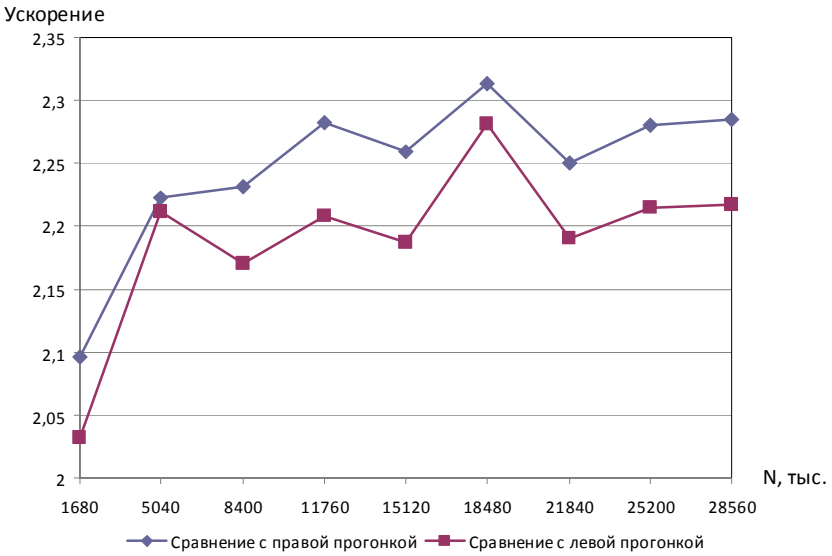


Рис. 1.2. Зависимость ускорения встречной прогонки от размера задачи

Перед тем, как переходить к экспериментам с большим числом потоков, выясним, какой из алгоритмов является наиболее быстрым в двухпоточной программе. Для сравнения будем использовать метод встречной прогонки, и два способа распараллеливания, описанных в п. 1.1.2.

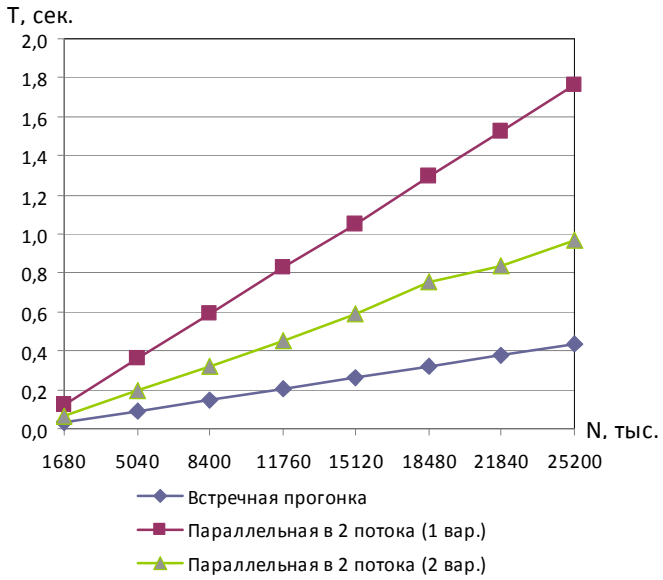


Рис. 1.3. Время работы различных вариантов метода прогонки

Как и следовало ожидать, наиболее быстродействующим оказался метод встречной прогонки. Однако встречную прогонку можно использовать только в двух потоках, для распараллеливания на большее число потоков нужно использовать вторую модификацию алгоритма, описанную в п. 1.1.2, которая обладает большей трудоемкостью, но и большей масштабируемостью. Ниже приведены результаты вычислительных экспериментов, полученные при использовании данной модификации параллельного метода прогонки.

Таблица 1.1. Результаты экспериментов (параллельная прогонка)

n, тыс.	1 поток	Параллельный алгоритм							
		2 потока		4 потока		6 потоков		8 потоков	
		T	*S*	*T*	*S*	*T*	*S*	*T*	*S*
1680	0,06	0,07	0,94	0,03	2,03	0,03	2,10	0,03	2,03

5040	0,20	0,20	1,00	0,11	1,83	0,09	2,29	0,09	2,14
8400	0,32	0,32	0,99	0,17	1,85	0,15	2,18	0,14	2,26
11760	0,45	0,45	0,98	0,23	1,91	0,20	2,25	0,20	2,20
15120	0,57	0,59	0,97	0,31	1,84	0,26	2,18	0,27	2,16
18480	0,72	0,76	0,95	0,41	1,78	0,32	2,25	0,31	2,31
21840	0,83	0,84	0,99	0,45	1,84	0,38	2,20	0,37	2,22
25200	0,96	0,97	0,99	0,53	1,80	0,43	2,25	0,44	2,19

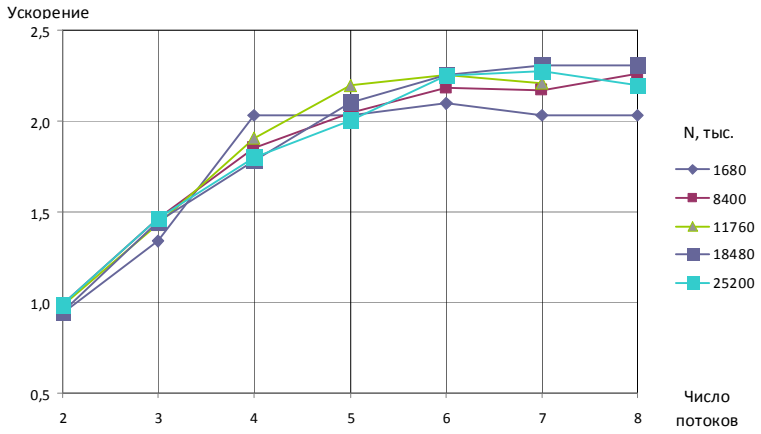


Рис. 1.4. Зависимость ускорения от числа потоков

На рис. 1.4 приведен график зависимости ускорения от числа потоков при использовании модификации алгоритма из п. 1.2.2. Видно, что при $p<6$ ускорение в целом соответствует оценке (1.16), полученной с помощью подсчета числа операций, необходимых для работы метода.

1.2. Метод исключения Гаусса

Рассмотрим теперь алгоритмы, предназначенные для решения системы

$$Ax=b, \tag{1.17}$$

с произвольной квадратной матрицей A. Основой для всех них служит широко известный *метод последовательного исключения неизвестных*, или же *метод Гаусса*.

Метод Гаусса основывается на возможности выполнения преобразований линейных уравнений, которые не меняют при этом решение рассматривае-

мой системы (такие преобразования носят наименование эквивалентных). К числу таких преобразований относятся:

- умножение любого из уравнений на ненулевую константу,
- перестановка уравнений,
- прибавление к уравнению любого другого уравнения системы.

Метод Гаусса включает последовательное выполнение двух этапов. На первом этапе, который называется *прямой ход*, исходная система линейных уравнений при помощи последовательного исключения неизвестных приводится к верхнему треугольному виду. При выполнении *обратного хода* (второй этап алгоритма) осуществляется определение значений неизвестных.

1.2.1. Последовательный алгоритм

Прямой ход состоит в последовательном исключении неизвестных в уравнениях решаемой системы линейных уравнений.

На итерации i, $1 \le i < n$, метода производится исключение неизвестной i для всех уравнений с номерами k, больших i (т.е. $i < k \le n$). Для этого из этих уравнений осуществляется вычитание строки i, умноженной на константу a_{ki}/a_{ii} с тем, чтобы результирующий коэффициент при неизвестной x_i в строках оказался нулевым – все необходимые вычисления могут быть определены при помощи соотношений:

$$\begin{aligned} a'_{kj} &= a_{kj} - \mu_{ki} a_{ij}, \\ b'_{k} &= b_{k} - \mu_{ki} b_{i}, \end{aligned} \quad i \le j \le n, i < k \le n, 1 \le i < n \ , \tag{1.18}$$

где $\mu_{ki} = a_{ki} / a_{ii}$ *– множители Гаусса*.

В итоге приходим к системе $Ux=c$ с верхней треугольной матрицей

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & \dots & u_{1,n} \\ 0 & u_{2,2} & \dots & u_{2,n} \\ & & \dots & \\ 0 & 0 & \dots & u_{n,n} \end{pmatrix},$$

При выполнении прямого хода метода Гаусса строка, которая используется для исключения неизвестных, носит наименование *ведущей*, а диагональный элемент ведущей строки называется *ведущим элементом*. Как можно заметить, выполнение вычислений является возможным только, ес-

ли ведущий элемент имеет ненулевое значение. Более того, если ведущий элемент a_{ii} имеет малое значение, то деление и умножение строк на этот элемент может приводить к накоплению вычислительной погрешности и вычислительной неустойчивости алгоритма.

Избежать подобной проблемы можно, если при выполнении каждой очередной итерации прямого хода метода Гаусса определить коэффициент с максимальным значением по абсолютной величине в столбце, соответствующем исключаемой неизвестной, т.е.

$$y = \max_{i \le k \le n} |a_{ki}| ,$$

и выбрать в качестве ведущей строку, в которой этот коэффициент располагается (данная схема выбора ведущего значения носит наименование *метода главных элементов*).

Обратный ход алгоритма состоит в следующем. После приведения матрицы коэффициентов к верхнему треугольному виду становится возможным определение значений неизвестных. Из последнего уравнения преобразованной системы может быть вычислено значение переменной x_n, после этого из предпоследнего уравнения становится возможным определение переменной x_{n-1} и т.д. В общем виде выполняемые вычисления при обратном ходе метода Гаусса могут быть представлены при помощи соотношений:

$$x_i = \left(c_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) \Big/ u_{ii} , \; i=n,\ldots,1.$$

Оценим трудоемкость метода Гаусса. При выполнении прямого хода число операций составит

$$\sum_{i=1}^{n-1} 2(n-i)^2 = \frac{n(n-1)(2n-1)}{3} = \frac{2}{3} n^3 + O(n^2) .$$

Для выполнения обратного хода потребуется

$$\sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2} = O(n^2) .$$

Таким образом, общее время выполнения метода Гаусса при больших n можно оценить как

$$T_1 = \frac{2}{3} n^3 \tau ,$$

где τ – время выполнения одной операции.

1.2.2. Параллельный алгоритм

При внимательном рассмотрении метода Гаусса можно заметить, что все вычисления сводятся к однотипным вычислительным операциям над строками матрицы коэффициентов системы линейных уравнений. Как результат, в основу параллельной реализации алгоритма Гаусса может быть положен принцип распараллеливания по данным. В качестве *базовой подзадачи* можно принять тогда все вычисления, связанные с обработкой одной строки матрицы A и соответствующего элемента вектора b. Рассмотрим общую схему параллельных вычислений и возникающие при этом информационные зависимости между базовыми подзадачами.

Для выполнения прямого хода метода Гаусса необходимо осуществить $(n-1)$ итерацию по исключению неизвестных для преобразования матрицы коэффициентов A к верхнему треугольному виду. Выполнение итерации i, $1 \le i \le n$, прямого хода метода Гаусса включает ряд последовательных действий. Прежде всего, в самом начале итерации необходимо выбрать ведущую строку, которая при использовании метода главных элементов определяется поиском строки с наибольшим по абсолютной величине значением среди элементов столбца i, соответствующего исключаемой переменной x_i. Зная ведущую строку, подзадачи выполняют вычитание строк, обеспечивая тем самым исключение соответствующей неизвестной x_i.

При выполнении обратного хода метода Гаусса подзадачи выполняют необходимые вычисления для нахождения значения неизвестных. Как только какая-либо подзадача i, $1 \le i \le n$, определяет значение своей переменной x_i, это значение должно быть использовано всеми подзадачам с номерами k, $k<i$: подзадачи подставляют полученное значение новой неизвестной и выполняют корректировку значений для элементов вектора b.

Выделенные базовые подзадачи характеризуются одинаковой вычислительной трудоемкостью. Однако размер матрицы, описывающей систему линейных уравнений, является существенно большим, чем число потоков в программе (т.е., $p << n$), и базовые подзадачи можно укрупнить, объединив в рамках одной подзадачи несколько строк матрицы. При этом применение последовательной схемы разделения данных для параллельного решения систем линейных уравнений приведет к неравномерной вычислительной нагрузке между потоками: по мере исключения (на прямом ходе) или определения (на обратном ходе) неизвестных в методе Гаусса для большей части потоков все необходимые вычисления будут завершены и они окажутся простаивающими. Возможное решение проблемы балансировки вычислений может состоять в использовании ленточной циклической схемы для распределения данных между укрупненными подзадачами.

В этом случае матрица A делится на наборы (полосы) строк вида (см. рис. 1.5).

$$A = (A_1, A_2, ..., A_p)^T,$$

$$A_i = (a_{i_1}, a_{i_2}, ..., a_{i_k}), i_j = i + jp, 1 \le j \le k, k = n / p.$$

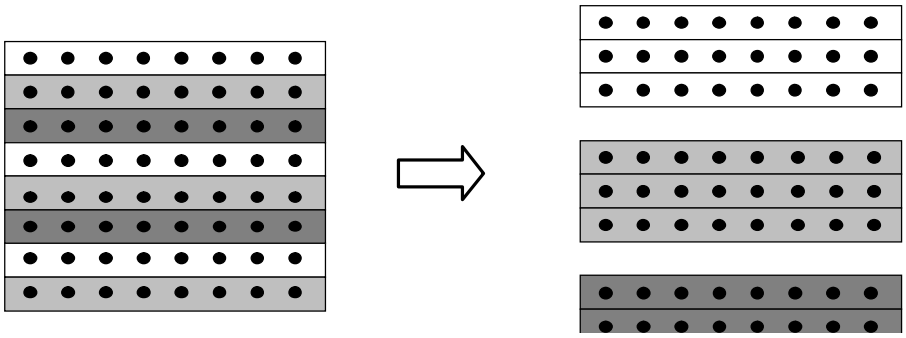


Рис. 1.5 Ленточная схема

Сопоставив схему разделения данных и порядок выполнения вычислений в методе Гаусса, можно отметить, что использование циклического способа формирования полос позволяет обеспечить лучшую балансировку вычислительной нагрузки между подзадачами.

Итак, проведя анализ последовательного варианта алгоритма Гаусса, можно заключить, что распараллеливание возможно для следующих вычислительных процедур:

- поиск ведущей строки,
- вычитание ведущей строки из всех строк, подлежащих обработке,
- выполнение обратного хода.

Оценим трудоемкость рассмотренного параллельного варианта метода Гаусса. Пусть n есть порядок решаемой системы линейных уравнений, а p, $p<n$, обозначает число потоков. При разработке параллельного алгоритма все вычислительные операции, выполняемые алгоритмом Гаусса, были распределены между потоками параллельной программы. Следовательно, время, необходимое для выполнения вычислений на этапе прямого хода, можно оценить как

$$T_p = T_1 / p.$$

Подставив выражение T_1 получим, что время выполнения вычислений для параллельного варианта метода Гаусса описывается выражением:

$$T_p = \frac{2n^3\tau}{3p}.$$

Теперь можно оценить величину накладных расходов, обусловленных организацией и закрытием параллельных секций. Пусть δ – время, необходимое на организацию и закрытие параллельной секции. Параллельная секция создается при каждом выборе ведущей строки, при выполнении вычитания ведущей строки из остальных строк линейной системы, подлежащих обработке, а также при выполнении каждой итерации обратного хода метода Гаусса. Таким образом, общее число параллельных секций составляет $3(n–1)$.

Сводя воедино все полученные оценки можно заключить, что время выполнения параллельного метода Гаусса описывается соотношением

$$T_p = \frac{2n^3\tau}{3p} + 3(n-1)\delta\ . \qquad (1.19)$$

1.2.3. Связь метода Гаусса и LU-разложения

LU-разложение – представление матрицы *A* в виде

$$A=LU, \qquad (1.20)$$

где *L* – нижняя треугольная матрица с диагональными элементами, равными единице, а *U* – верхняя треугольная матрица с ненулевыми диагональными элементами. *LU*-разложение также называют *LU*-факторизацией. Известно [4], что *LU*-разложение существует и единственно, если главные миноры матрицы *A* отличны от нуля.

Алгоритм *LU*-разложения тесно связан с методом исключения Гаусса. В самом деле, пусть мы решаем систему уравнений вида (1.2). Непосредственно проверяется, что преобразования *k*-го шага метода Гаусса равносильны домножению системы (1.2) слева на матрицу

$$M^{(k)} = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -\mu_{k+1,k} & 1 & & \\ & & -\mu_{k+2,k} & & 1 & \\ & & & & & \ddots \\ & & -\mu_{n,k} & & & 1 \end{bmatrix},$$

где μ_{ik} – множители Гаусса из (1.18). Как было рассмотрено в п. 1.2.1, прямой ход метода Гаусса преобразует исходную систему уравнений к виду

$$Ux=c,$$

с верхней треугольной матрицей U. Зная матрицы $M^{(i)}$, можно записать матрицу U и вектор c как

$$U = M^{(n-1)}M^{(n-2)}...M^{(1)}A,$$

$$c = M^{(n-1)}M^{(n-2)}...M^{(1)}b.$$

Обозначим $L^{-1} = M^{(n-1)}M^{(n-2)}...M^{(1)}$. Можно непосредственно проверить, что

$$L = \begin{bmatrix} 1 & & & & \\ \mu_{21} & 1 & & & \\ \vdots & \vdots & \ddots & & \\ \mu_{n-1,1} & \mu_{n-1,2} & \cdots & 1 & \\ \mu_{n,1} & \mu_{n,2} & \cdots & \mu_{n,n-1} & 1 \end{bmatrix}.$$

Отсюда получаем $A=LU$.

Таким образом, матрицу L можно получить как нижнюю треугольную матрицу коэффициентов Гаусса, а матрицу U – как верхнюю треугольную матрицу, получаемую в результате работы метода Гаусса. При этом очевидно, что трудоемкость получения LU-факторизации будет такой же – $2/3n^3 + O(n^2)$.

Рассмотренный нами алгоритм LU-факторизации реализован с помощью *исключения по столбцу*. Следует отметить, что можно сформулировать аналогичный алгоритм, основанный на *исключении по строке*. В самом деле, основная идея алгоритма с помощью исключения по столбцу заключа-

ется в том, что на *i*-й итерации ведущая строка с подходящими множителями вычитается из строк, лежащих ниже, чтобы занулить все элементы матрицы, расположенные в *i*-м столбце ниже диагонали. Между тем возможно и другое: на каждой *i*-й итерации можно вычитать из *i*-й строки все строки, расположенные выше, умноженные на подходящие коэффициенты, так, чтобы занулить все элементы *i*-й строки левее диагонали. При этом элементы матрицы *L* ниже главной диагонали и элементы матрицы *U* на главной диагонали и выше нее можно вычислять на месте матрицы *A*. Как и в случае исключения по столбцу, приведенная схема требует проведения $2/3n^3 + O(n^2)$ операций.

Рассмотрим теперь еще один способ *LU*-факторизаци, называемый *компактной схемой*. Пусть матрица $A(n\times n)$ допускает *LU*-разложение (1.20), где

$$L = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ . & . & \dots & . \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix}, \; U = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ . & . & \dots & . \\ 0 & 0 & \dots & u_{nn} \end{bmatrix},$$

т.е. $l_{ii} = 1$, $l_{ij} = 0$ при $i<j$, а $u_{ij} = 0$ при $j<i$. Из соотношения (1.20) следует, что

$$a_{ij} = \sum_{k=1}^{n} l_{ik} u_{kj}, \; i, j = \overline{1,n}.$$

Преобразуем эту сумму двумя способами:

$$\sum_{k=1}^{n} l_{ik} u_{kj} = \sum_{k=1}^{i-1} l_{ik} u_{kj} + l_{ii} u_{ij} + \sum_{k=i+1}^{n} l_{ik} u_{kj} = \sum_{k=1}^{i-1} l_{ik} u_{kj} + l_{ii} u_{ij},$$

$$\sum_{k=1}^{n} l_{ik} u_{kj} = \sum_{k=1}^{j-1} l_{ik} u_{kj} + l_{ij} u_{jj} + \sum_{k=j+1}^{n} l_{ik} u_{kj} = \sum_{k=1}^{j-1} l_{ik} u_{kj} + l_{ij} u_{jj}.$$

Отсюда находим

$$l_{11} = 1, \; u_{11} = a_{11},$$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, \; i \le j, i = \overline{1, j-1}, \; j = \overline{2,n},$$

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) \Big/ u_{jj} \ , \ i>j, \ \ j = \overline{1, n-1} \, , \ i = \overline{2, n} \, .$$

Оценка числа операций данного алгоритма *LU*-факторизаци также составляет

$$\frac{2}{3} n^3 + O(n^2) \, .$$

Если разложение (1.20) получено, то решение системы (1.2) сводится к последовательному решению двух систем уравнений с треугольными матрицами (*обратный ход*)

$$Ly=b, \ Ux=y. \qquad (1.21)$$

Обратный ход требует $O(n^2)$ операций.

Как следует из приведенных оценок, вычислительная сложность метода исключения Гаусса и метода *LU*-разложения одинакова. Однако если необходимо решить несколько систем с одинаковыми матрицами коэффициентов, но различными векторами свободных членов (правая часть СЛАУ), то метод *LU*-разложения окажется предпочтительным, так как в этом случае нет необходимости производить разложение матрицы коэффициентов многократно. Достаточно лишь сохранить полученные треугольные матрицы в памяти и, подставляя различные вектора свободных членов, получать решения методами прямой и обратной подстановки. Это позволит значительно сократить объем вычислений по сравнению с методом Гаусса.

1.2.4. Результаты вычислительных экспериментов

Вычислительные эксперименты для оценки эффективности параллельного варианта метода Гаусса для решения систем линейных уравнений проводились при условиях, указанных во введении.

Результаты вычислительных экспериментов приведены в таблице 1.2 (время работы алгоритмов указано в секундах).

Табл. 1.2. Результаты экспериментов (параллельный метод Гаусса)

n	Посл.	Параллельный алгоритм							
		2 потока		4 потока		6 потоков		8 потоков	
		T	*S*	*T*	*S*	*T*	*S*	*T*	*S*
1000	0,44	0,48	0,90	0,28	1,56	0,23	1,93	0,20	2,15
2000	4,45	4,20	1,06	3,01	1,48	2,86	1,56	2,75	1,62
3000	16,29	14,27	1,14	10,98	1,48	10,89	1,50	10,87	1,50

4000	34,62	33,90	1,02	26,18	1,32	26,10	1,33	26,18	1,32
5000	72,20	68,19	1,06	51,45	1,40	51,11	1,41	51,87	1,39

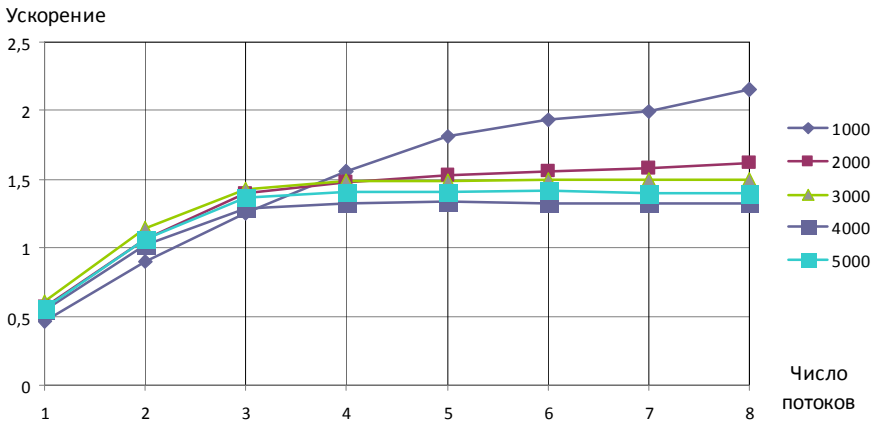


Рис. 1.6. Зависимость ускорения от числа потоков (метод Гаусса)

Как следует из приведенных результатов, при $p \geq 3$ получается ускорение около 1.5, которое не зависит ни от числа потоков, ни от размера матрицы, тогда как оценка (1.19) позволяла нам надеяться на лучшее (исключение составляет случай N=1000). Отсутствие значительного ускорения обусловлено следующим эффектом.

Во-первых, при N=1000 размер матрицы позволяет целиком загрузить ее в кэш процессора, и использовать для проведения операций быстродействующую память. А при N>1000 матрица не помещается в кэш-память целиком, и возрастает число *кэш-промахов*, что приводит к частым обращениям в относительно медленную оперативную память. Эффективно использовать кэш-память можно при блочном разделении данных, что будет показано в следующем пункте.

Во-вторых, компилятор с большей эффективностью оптимизирует чисто последовательную программу, чем более сложную параллельную. И в некоторых случаях однопоточная версия параллельной программы будет работать в 1.5-2 раза медленнее, чем ее чисто последовательный аналог, поэтому здесь и далее мы будем оценивать ускорение параллельной программы, запущенной в p потоков, по отношению к той же самой программме, запущенной в один поток.

Для иллюстрации сказанного приведем результаты сравнения параллельного алгоритма не с чисто последовательной программой, а с параллельной, запущенной в один поток. Результаты такого сравнения приведены в таблице 1.3 (время работы алгоритмов указано в секундах).

Табл. 1.3. Результаты экспериментов (параллельный метод Гаусса)

N	1 поток	Параллельный алгоритм							
		2 потока		4 потока		6 потоков		8 потоков	
		T	*S*	*T*	*S*	*T*	*S*	*T*	*S*
1000	0,94	0,48	1,94	0,28	3,34	0,23	4,14	0,20	4,61
2000	7,82	4,20	1,86	3,01	2,60	2,86	2,74	2,75	2,85
3000	26,72	14,27	1,87	10,98	2,43	10,89	2,45	10,87	2,46
4000	63,77	33,90	1,88	26,18	2,44	26,10	2,44	26,18	2,44
5000	130,09	68,19	1,91	51,45	2,53	51,11	2,55	51,87	2,51

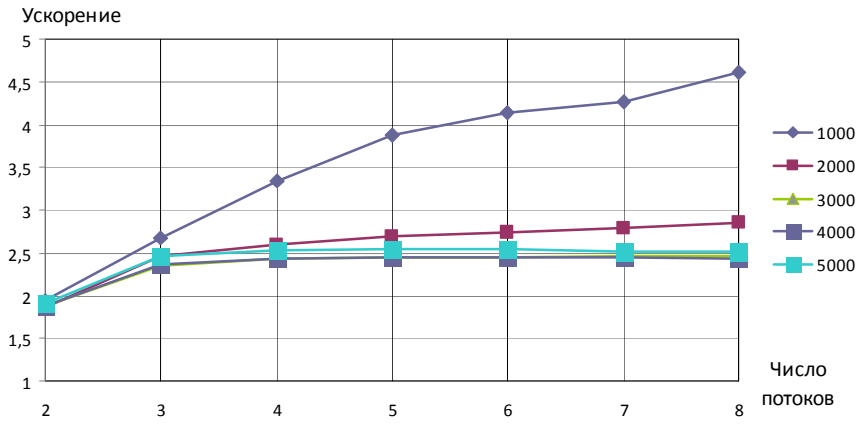


Рис. 1.7. Зависимость ускорения от числа потоков (метод Гаусса)

Как следует из приведенных результатов, относительно хорошее ускорение мы получили лишь для задачи размера 1000 (до 4.6 при решении в 8 потоках), в остальных задачах ускорение составляет примерно 2.5 при $p \geq 3$. В следующем параграфе будет рассмотрен метод, обладающий значительно большим масштабированием.

1.2.5. Блочное *LU*-разложение

Недостаток рассмотренного стандартного алгоритма *LU*-разложения обусловлен тем, что его вычисления плохо соответствует правилам использования *кэш-памяти* – быстродействующей дополнительной памяти компьютера, используемой для хранения копии наиболее часто используемых областей оперативной памяти. Эффективное использование кэша может существенно (в десятки раз) повысить быстродействие вычислений. Размещение данных в кэше может происходить или предварительно (при использовании тех или иных алгоритмов предсказания потребности в данных) или в момент извлечения значений из основной оперативной памяти. При этом подкачка данных в кэш осуществляется не одиночными значениями, а небольшими группами – строками кэша. Загрузка значений в строку кэша осуществляется из последовательных элементов памяти; типовые размеры строки кэша обычно равны 32, 64, 128, 256 байтам (см., например, [7]). Как результат, эффект наличия кэша будет наблюдаться, если выполняемые вычисления используют одни и те же данные многократно и осуществляют доступ к элементам памяти с последовательно возрастающими адресами.

В рассмотренном нами алгоритме *LU*-разложения размещение данных в памяти осуществляется по строкам, а вычисления проводятся – по столбцам, и это приводит к низкой эффективности использования кэша. Возможный способ улучшения ситуации – укрупнение вычислительных операций, приводящее к последовательной обработке некоторых прямоугольных подматриц матрицы *A*.

LU-разложение можно организовать так, что матричные операции (реализация которых допускает эффективное использование кэш-памяти) станут основными. Для этого представим матрицу $A \in R^{n\times n}$ в блочном виде

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{matrix} r \\ n-r \end{matrix},$$
$$\begin{matrix} r & n-r \end{matrix}$$

где r – блочный параметр, A_{11} – подматрица матрицы A размера $r\times r$, A_{12} – размера $r\times(n–r)$, A_{21} – размера $(n–r)\times r$, A_{22} – размера $(n–r)\times(n–r)$. Компоненты L и U искомого разложения также запишем в блочном виде

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{matrix} r \\ n-r \end{matrix}, \quad U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \begin{matrix} r \\ n-r \end{matrix},$$
$$\begin{matrix} r & n-r \end{matrix} \qquad \begin{matrix} r & n-r \end{matrix}$$

где L_{11}, L_{21}, L_{22}, U_{11}, U_{12}, U_{22} – соответствующего размера подматрицы матриц L и U.

Рассмотрим теперь связь между исходной матрицей и ее разложением в блочном виде. Используя соотношение (1.20), запишем

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}$$

Так как L_{11} и U_{11} являются блоками LU-разложения матрицы A, и

$$A_{11} = L_{11}U_{11},$$

то их можно найти, применив стандартный метод Гаусса для разложения блока A_{11}. Затем можно найти блоки L_{21}, U_{12}, решив треугольные системы с несколькими правыми частями

$$L_{11}U_{12} = A_{12},$$

$$L_{21}U_{11} = A_{21}.$$

Следующий шаг алгоритма состоит в вычислении редуцированной матрицы $\tilde{A}_{22}$, в процессе которого используются ставшие известными блоки L_{21} и U_{12}, блок A_{22} и соотношение $A_{22} = L_{21}U_{12} + L_{22}U_{22}$,

$$\tilde{A}_{22} = A_{22} - L_{21}L_{21}^T = L_{22}U_{22}.$$

Как следует из данной формулы, LU-разложение редуцированной матрицы $\tilde{A}_{22}$ совпадает с искомыми блоками L_{22}, U_{22} матрицы A, и для его нахождения можно применить описанный алгоритм рекурсивно.

Так же как и иные рассмотренные процедуры разложения, приведенная блочная схема требует порядка $2/3n^3$ операций. Оценим долю матричных операций.

Пусть размер матрицы кратен размеру блока, т.е. $n = rN$. Операции, не являющиеся матричными, используются при выполнении разложения

$$A_{11} = L_{11}U_{11}$$

что требует порядка $2/3r^3$ операций. Так как в процессе блочного разложения приходится решать N подобных систем, то долю матричных операций можно оценить как

$$1-\frac{N\,2r^3/3}{2n^3/3}=1-\frac{1}{N^2}.$$

Значит, при правильном подборе размера блока $r << n$ почти все операции в данном алгоритме будут матричными, и к их реализации нужно подойти с особой тщательностью. В данном случае для проведения операции матричного умножения целесообразно воспользоваться блочной схемой умножения матриц, выбрав для этого свой размер блока, меньший r. Описание блочного алгоритма умножения матриц см., например, в [15].

Из сказанного следует, что распараллеливание блочного алгоритма *LU*-разложения должно осуществлять на уровне матричных операций, распараллеливание которых также подробно рассмотрено в работе [15].

1.2.6. Результаты вычислительных экспериментов

Сначала приведем результаты экспериментов, подтверждающие эффективность блочного *LU*-разложения по сравнению с его исходным неблочным вариантом. В таблице 1.4 и на рисунке 1.8 приведено время работы блочного алгоритма в зависимости от размера матрицы n и размера блока r (столбец, отмеченный прочерком, соответствует исходному неблочному алгоритму).

Табл. 1.4. Время работы блочного *LU*-разложения

n	Время работы, сек						
	---	r=2	r=5	r=10	r=20	r=50	r=100
1000	0,44	0,568	0,49	0,474	0,464	0,565	0,64
2000	4,45	4,846	3,994	3,76	3,869	4,29	4,992
3000	16,29	16,364	13,556	12,76	13,9	14,352	17,035
4000	34,62	39,921	32,043	30,561	34,055	34,944	40,513
5000	72,20	75,77	62,93	59,842	65,645	66,987	82,15

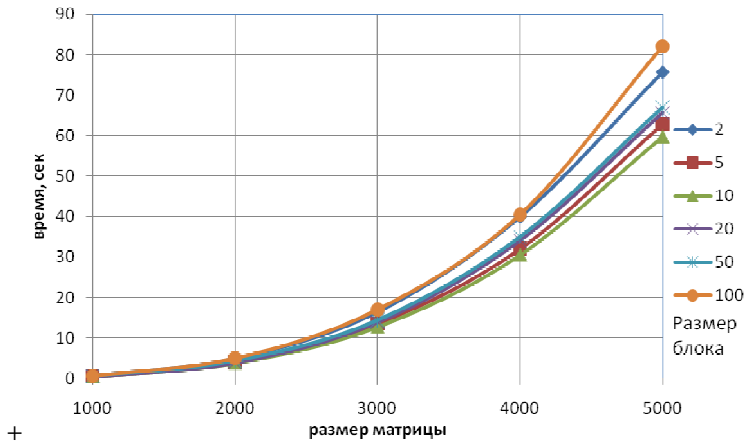


Рис. 1.8. Время работы блочного *LU*-разложения

Результаты показывают, что блочный алгоритм при r=10 работает быстрее как своего прототипа, так и блочного алгоритма при других размерах блока. Поэтому дальнейшие эксперименты будем проводить для блока размера r=10.

Приведем теперь характеристики работы (время и ускорение) параллельного блочного *LU*-разложения.

Табл. 1.5. Время работы параллельного блочного LU-разложения

N	1 поток	Параллельная версия							
		2 потока		4 потока		6 потоков		8 потоков	
		T	*S*	*T*	*S*	*T*	*S*	*T*	*S*
1000	0,62	0,31	2,00	0,17	3,65	0,11	5,72	0,08	8,00
2000	5,07	2,54	1,99	1,34	3,78	0,91	5,60	0,69	7,39
3000	17,02	8,53	1,99	4,52	3,76	3,03	5,62	2,29	7,42
4000	40,59	20,58	1,97	10,81	3,75	7,29	5,57	5,51	7,37
5000	78,80	39,89	1,98	20,83	3,78	14,01	5,62	10,61	7,43

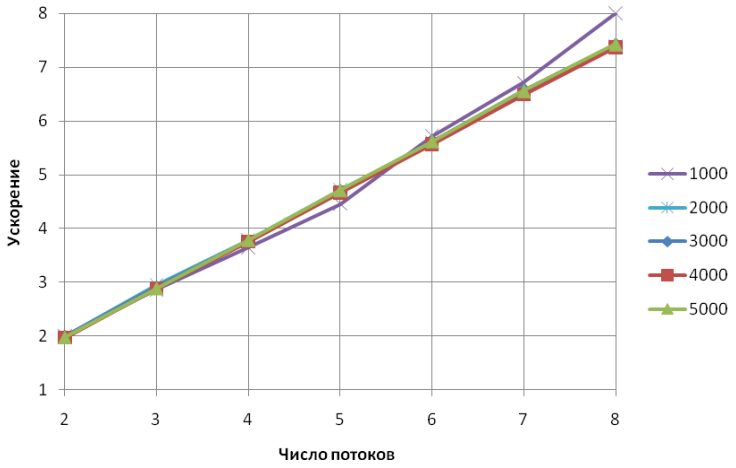


Рис. 1.9. Ускорение параллельного блочного *LU*-разложения

Приведенные результаты свидетельствуют о хорошей (практически линейной) масштабируемости блочного алгоритма, в отличие от стандартной версии метода Гаусса.

1.3. Метод Холецкого

Метод Холецкого (известный также как *метод квадратного корня*) предназначен для решения систем уравнений вида (1.2) с действительной симметричной положительно определенной матрицей. Напомним, что мат-

рица назвывается *положительно определенной*, если для любого вектора $x \in R^n$ выполнено неравенство

$$(Ax, x) \geq 0.$$

Матрицы с такими свойствами возникают, например, при использовании метода наименьших квадратов и численном решении дифференциальных уравнений.

Метод Холецкого основан на разложении матрицы A в произведение

$$A = LL^T \tag{1.24}$$

где L – нижняя треугольная матрица с положительными элементами на главной диагонали. Известно [4], что разложение (1.24) можно получить для любой матрицы, удовлетворяющей указанным свойствам. Существует также обобщение этого разложения на случай комплекснозначных матриц [4].

Если разложение (1.24) получено, то решение системы (1.2) сводится к последовательному решению двух систем уравнений с треугольными матрицами

$$Ly = b, \; L^T x = y. \tag{1.25}$$

Следует отметить, что на практике часто необходимо решить последовательность систем вида (1.2), отличающихся лишь правой частью b. Учитывая этот факт и соотношения (1.25), однократная факторизация матрицы A значительно упрощает решение оставшихся систем.

1.3.1. Последовательный алгоритм

Получим расчетные формулы метода. Из условия (1.24) следует, что

$$a_{ij} = \sum_{k=1}^{n} l_{ik} l_{kj}^T \;, \; i, j = \overline{1, n} \, .$$

Так как матрица A – симметричная, можно рассмотреть лишь случай $i \leq j$. Так как $l_{ik} = 0$ при $i < k$, данные условия можно записать в виде

$$a_{ij} = \sum_{k=1}^{i-1} l_{ik} l_{kj}^T + l_{ii} l_{ij} + \sum_{k=i+1}^{n} l_{ik} l_{kj}^T = \sum_{k=1}^{i-1} l_{ik} l_{kj}^T + l_{ii} l_{ij} \, ,$$

В частности, при $i=j$ получаем

$$a_{ii} = \sum_{k=1}^{i-1} l_{ik}^2 - l_{ii}^2 \, .$$

откуда

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}\ .$$

Далее, при $i<j$ получим

$$l_{ij} = \frac{1}{l_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right). \tag{1.26}$$

Итак, если предположить, что метод квадратного корня применяется к системе уравнений с действительной симметричной положительно определенной матрицей, то элементы матрицы L можно вычислить, начиная с ее левого угла, по следующим расчетным формулам:

$$l_{11} = \sqrt{a_{11}}\ ,\ l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}\ ,\ i = \overline{2,n}\,. \tag{1.27}$$

$$l_{i1} = \frac{a_{i1}}{l_{11}}\,,\ l_{ij} = \frac{1}{l_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right),\ i = \overline{2,n}\,,\ j = \overline{2,i-1}\,. \tag{1.28}$$

Подсчитаем число операций, требующихся для выполнения разложения. Вычисления по формуле (1.27) требуют

$$\sum_{i=2}^{n} 2(i-1) = n(n-1) = O(n^2)$$

операций. Вычисления по формуле (1.28) при каждом фиксированном j требуют

$$\sum_{i=2}^{j-1} 2(i-1) = (j-2)(j-1)$$

операций, а всего потребуется

$$\sum_{j=2}^{n} (j-2)(j-1) = \sum_{k=1}^{n-1} k(k-1) = \frac{n(n-1)(n-2)}{3} = \frac{1}{3}n^3 + O(n^2)$$

операций. Следует отметить, что в дополнение к указанным действиям для расчетов по формулам (1.27), (1.28) потребуется n операций извлечения корня.

Если матрица A факторизована в виде $A=LL^T$, то обратный ход метода квадратного корня состоит в последовательном решении двух систем уравнений

$$Ly=b,\ L^Tx=y.$$

Решения этих систем находятся по формулам, аналогичным формулам обратного хода метода Гаусса

$$y_1=\frac{b_1}{l_{11}},\ y_i=\frac{1}{l_{ii}}\left(b_i-\sum_{j=1}^{i-1}l_{ij}y_j\right),\ i=\overline{2,n}. \tag{1.29}$$

$$x_n=\frac{y_n}{l_{nn}},\ x_i=\frac{1}{l_{ii}}\left(y_i-\sum_{j=i+1}^{n}l_{ji}x_j\right),\ i=\overline{n-1,1}. \tag{1.30}$$

Вычисления по формулам (1.29), (1.30) потребуют $2n(n+1)$ операций, что не оказывает существенного влияния на кубическую трудоемкость метода.

Таким образом, общее время работы метода можно оценить как

$$T_1=\frac{1}{3}n^3\tau,$$

где τ – время выполнения одной операции.

При больших n время работы метода Холецкого будет примерно в два раза меньше, чем метода Гаусса, это сокращение объясняется тем, что A – симметричная матрица, и метод Холецкого учитывает данную особенность задачи.

1.3.2. Организация параллельных вычислений

Как и в методе исключения Гаусса, в данном алгоритме все вычисления сводятся к однотипным вычислительным операциям над строками матрицы L. Как результат, в основу параллельной реализации разложения может быть положен принцип распараллеливания по данным. В качестве базовой подзадачи можно принять тогда все вычисления, связанные с обработкой одной строки матрицы L.

Рассмотрим общую схему параллельных вычислений и возникающие при этом информационные зависимости между базовыми подзадачами.

При выполнении прямого хода метода необходимо осуществить $(n-1)$ итерацию для получения нижней треугольной матрицы L. Выполнение итерации i, $1\le i\le n$, включает ряд последовательных действий. Прежде всего, в самом начале итерации необходимо вычислить диагональный элемент l_{ii}.

Зная диагональный элемент, подзадачи выполняют вычисление i-го столбца матрицы L.

При выполнении обратного хода метода Холецкого подзадачи выполняют необходимые вычисления для нахождения значений сначала вспомогательного вектора y, а затем вектора неизвестных x. Как только какая-либо подзадача i, $1 \le i \le n$, определяет значение своей переменной y_i (или x_i), это значение должно быть использовано всеми подзадачами с номерами k, где $k > i$ при решении системы относительно y, и $k < i$ при решении системы относительно x; подзадачи подставляют полученное значение новой неизвестной и выполняют корректировку значений для элементов вектора правой части.

При использовании систем с общей памятью размер матрицы, описывающей систему линейных уравнений, является большим, чем число потоков (т.е., $p < n$). Следовательно, базовые подзадачи нужно укрупнить, объединив в рамках одной подзадачи несколько строк матрицы. Здесь также можно использовать циклическую схему распределения данных, аналогично методу Гаусса (см. п. 1.2.2).

Итак, проведя анализ последовательного варианта метод Холецкого, можно заключить, что распараллеливание возможно для следующих вычислительных процедур:

- вычисление диагонального элемента l_{ii} в соответствии с (1.27),
- вычисление элементов i-го столбца в соответствии с (1.28),
- выполнение обратного хода в соответствии с (1.29), (1.30).

Оценим трудоемкость рассмотренного параллельного варианта метода Холецкого. Пусть n есть порядок решаемой системы линейных уравнений, а p, $p < n$, обозначает число потоков в программе.

При разработке параллельного алгоритма все вычислительные операции, были распределены между потоками параллельной программы. Пусть δ – время, необходимое на организацию и закрытие параллельной секции. Параллельная секция создается при каждом вычислении диагонального элемента, при вычислении элементов текущего столбца матрицы L, а также на каждой итерации при решении двух систем с треугольными матрицами. Таким образом, общее число параллельных секций составляет $4n$.

Следовательно, теоретическая оценка времеми, необходимого для решения системы, составит

$$T_p = \frac{n^3}{3p}\tau + 4n\delta .$$

1.3.1. Результаты вычислительных экспериментов

Вычислительные эксперименты для оценки эффективности параллельного варианта блочного метода Холецкого проводились при стандартных условиях, указанных во введении. С целью формирования симметричной положительно определенной матрицы A размера $n \times n$ для тестовой системы уравнений диагональный элемент матрицы генерировался в диапазоне от n до $2n$, остальные элементы генерировались в диапазоне от 0 до 1 (с учетом симметрии).

Результаты вычислительных экспериментов приведены в таблице 1.6 (время работы алгоритмов указано в секундах).

Табл. 1.6. Результаты экспериментов (параллельное разложение Холецкого)

Размер	1 поток	Параллельный алгоритм							
		2 потока		4 потока		6 потоков		8 потоков	
		t	*S*	*t*	*S*	*t*	*S*	*t*	*S*
1000	0,12	0,06	1,86	0,04	3,16	0,03	3,90	0,03	4,68
2000	1,07	0,52	2,04	0,29	3,66	0,20	5,30	0,17	6,26
3000	4,20	2,64	1,59	2,00	2,10	1,87	2,24	1,75	2,40
4000	10,11	6,76	1,50	5,54	1,83	5,54	1,83	5,49	1,84
5000	19,75	13,53	1,46	11,37	1,74	11,28	1,75	11,28	1,75

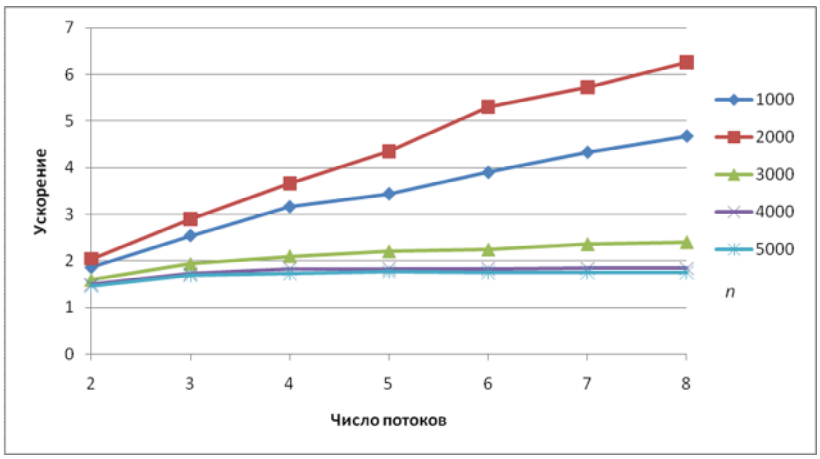


Рис. 1.10. Зависимость ускорения от числа потоков (разложение Холецкого)

Как показывают эксперименты, алгоритм хорошо масштабируется для матриц размера не более 2000. Действительно, при таком размере матрицы A ее нижний треугольник (а только он и нужен для проведения расчетов в силу симметрии матрицы) целиком помещается в кэш-память компьютера. При больших размерах матрицы возрастает число кэш-промахов. Сгладить данный эффект и получить масштабируемый алгоритм нам поможет, как и в случае метода Гаусса, блочный подход к обработке данных.

1.3.2. Блочный алгоритм

Рассмотрим теперь, как и в п. 1.2.5, вычислительную процедуру, основанную на идее разбиения матрицы на блоки и ориентированную на эффективную работу с кэш-памятью. Разложение осуществляется путем переписывания исходной матрицы элементами искомого фактора Холецкого сверху вниз по блокам.

Первый шаг блочного алгоритма заключается в следующем. Пусть мы определили размер блока как r, тогда исходную матрицу A можно представить в виде

$$A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix},$$

где A_{11} – подматрица матрицы A размера $r \times r$, A_{21} – размера $(n-r) \times r$, A_{22} – размера $(n-r) \times (n-r)$. Искомый фактор Холецкого также можно записать в блочном виде как

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix},$$

где L_{11}, L_{21}, L_{22} – соответствующего размера подматрицы фактора L.

Рассмотрим теперь связь между блоками фактора и исходной матрицы. Используя соотношение (1.24), запишем

$$\begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \cdot \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix} = \begin{bmatrix} L_{11}L_{11}^T & L_{11}L_{21}^T \\ L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{bmatrix},$$

откуда получим

$$A_{11} = L_{11}L_{11}^T, \tag{1.31}$$

$$A_{21} = L_{21}L_{11}^T, \tag{1.32}$$

$$A_{22} = L_{21}L_{21}^T + L_{22}L_{22}^T. \tag{1.33}$$

Используя данные матричные равенства, можно найти блоки L_{11}, L_{21} из искомого разложения.

Блок L_{11} может быть получен с помощью обычного неблочного алгоритма, изложенного в п. 1.3.1, т.к. формула (1.31) соответствует разложению Холецкого для матрицы A_{11}.

Далее, используя известный блок A_{21} и найденный блок L_{11}, из соотношения (1.32) можно найти блок L_{21}. Для этого потребуется решить r вспомогательных систем из r уравнений с одинаковой матрицей и разными правыми частями. Но так как матрица L_{11} является треугольной, то решение одной системы потребует лишь $O(r^2)$ операций. Всего же для нахождения блока L_{21} потребуется $O(r^3)$ операций с плавающей точкой.

Следующий шаг алгоритма состоит в вычислении редуцированной матрицы $\tilde{A}_{22}$, при котором используется ставший известным блок L_{21}, блок A_{22} и соотношение (1.33),

$$\tilde{A}_{22} = A_{22} - L_{21}L_{21}^T = L_{22}L_{22}^T. \tag{1.34}$$

Как следует из данной формулы, фактор Холецкого для матрицы $\tilde{A}_{22}$ совпадает с искомым блоком L_{22}, и для его нахождения можно применить описанный алгоритм рекурсивно.

По построению матрица $\tilde{A}_{22}$ является симметричной положительно определенной, и при реализации алгоритма нет необходимости вычислять ее полностью, достаточо вычислить в соответствии с формулой (1.34) ее нижний треугольник.

Процесс блочной факторизации проиллюстрирован на рис. 1.11.

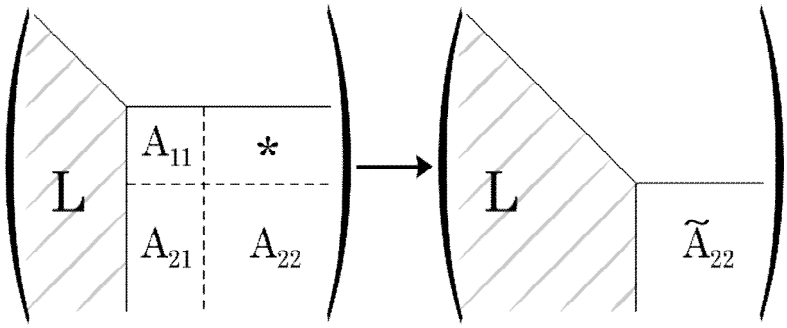


Рис. 1.11. Метод Холецкого, блочный алгоритм

Известно [8], что данная вычислительная процедура включает в себя

$$\frac{n^3}{3}+O\left(n^2\right)$$

операций, как и другие возможные реализации метода Холецкого; при этом вклад матричных операций в общее число действий (аналогично блочному *LU*-разложению) аппроксимируется величиной

$$1-\frac{1}{N^2},$$

где $n=rN$. Отсюда следует, что матричные операции, в частности, в соответствии с формулой (1.34), будут составлять большую часть вычислений, и к реализации матричного умножения нужно подойти с особой тщательностью. В данном случае для проведения операции матричного умножения целесообразно воспользоваться блочной схемой умножения матриц, выбрав для этого свой размер блока, меньший *r*. Описание блочного алгоритма умножения матриц см., например, в [15].

Рассмотрим теперь способы параллельной реализации блочного метода Холецкого. После внимательного анализа данного алгоритма можно заключить, что распараллеливание возможно для следующих вычислительных процедур:

- вычисление блока L_{11} в соответствии с (1.31) (параллельная версия неблочного алгоритма);
- вычисление блока L_{21} в соответствии с (1.32) (параллельное решение набора систем линейных уравнений с треугольной матрицей);
- выполнение матричного умножения в соответствии с (1.34);
- выполнение обратного хода в соответствии с (1.29), (1.30).

В силу большей трудоемкости прямого хода алгоритма (т.е. собственно разложения) эффективность параллельного блочного алгоритма будет определяться эффективностью распараллеливания матричных операций, составляющих основную долю операция прямого хода.

1.3.3. Результаты вычислительных экспериментов

Сначала приведем время работы блочного алгоритма Холецкого и его исходного неблочного варианта (время работы *t* указано в секундах, столбец, отмеченный прочерком, соответствует неблочному алгоритму).

Табл. 1.7. Сравнение блочного и неблочного разложения Холецкого

n	Время работы *t*, с					
	---	*r*=10	*r*=20	*r*=50	*r*=100	*r*=200
1000	0,11	0,28	0,21	0,16	0,16	0,15
2000	1,05	2,62	1,99	1,39	1,23	1,09
3000	4,16	9,81	7,25	4,84	4,17	3,71
4000	10,00	25,05	18,22	11,74	10,00	9,63
5000	19,59	49,23	35,82	23,24	20,39	20,22

Из таблицы следует, что блочный алгоритм значительно проигрывает неблочному при малых размерах блока факторизации, и почти сравнивается по скорости при увеличении размера блока. Данный факт объясняется тем, что при реализации формулы (1.34) мы использовали алгоритм умножения матриц по определению. Посмотрим, что изменится, если применить блочный алгорим матричного умножения, который более эффективно использует кэш-память.

Для выбора оптимального размера блока при выполнении матричного умножения нами были проведены эксперименты при размерах блока от 10×10 до 200×200 с шагом 5. Лучшие результаты приведены в таблице ниже.

Табл. 1.8. Сравнение блочного и неблочного разложения Холецкого

n	Время работы *t*			
	---	*r*=50 (25×50)	*r*=100 (25×50)	*r*=200 (10×200)
1000	0,12	0,11	0,12	0,13
2000	1,07	0,78	0,80	0,83
3000	4,20	2,57	2,61	2,64
4000	10,11	5,98	5,99	6,10
5000	19,75	12,57	11,50	11,84

Ситуация значительно улучшилась – теперь блочный алгоритм показывает такое же быстродействие на матрицах малого размера (которые полностью умещаются в кэш-память), и значительно обгоняет неблочный на матрицах большого размера. При этом время, затрачиваемое алгоритмом, является примерно одинаковым для любых размеров блоков факторизации *r* (отличие есть лишь в сотых долях секунды).

Осталось убедиться в том, что наше решение будет масштабируемым – проведем разложение с использованием параллельного блочного алгоритма с размером блока факторизации r=200 и блоком матричного умножения 10×200 в соответствии с описанной схемой распараллеливания. Как уже обсуждалось ранее, время работы последовательной программы и параллельной программы, запущенной в один поток, будет разным. Этим объясняются разные времена работы алгоритмов, приведенных в табл. 1.8 и 1.9.

Табл. 1.9. Параллельное блочное разложение Холецкого

n	1 поток	Параллельный алгоритм							
		2 потока		4 потока		6 потоков		8 потоков	
		t	*S*	*t*	*S*	*t*	*S*	*t*	*S*
1000	0,14	0,08	2,15	0,05	3,77	0,05	4,98	0,03	4,98
2000	0,87	0,48	1,90	0,28	3,52	0,22	4,84	0,19	5,80
3000	2,79	1,50	1,95	0,83	3,55	0,62	4,94	0,51	6,10
4000	6,43	3,35	1,94	1,90	3,51	1,37	4,91	1,11	6,03
5000	12,39	6,44	1,94	3,56	3,52	2,54	4,87	2,04	6,05

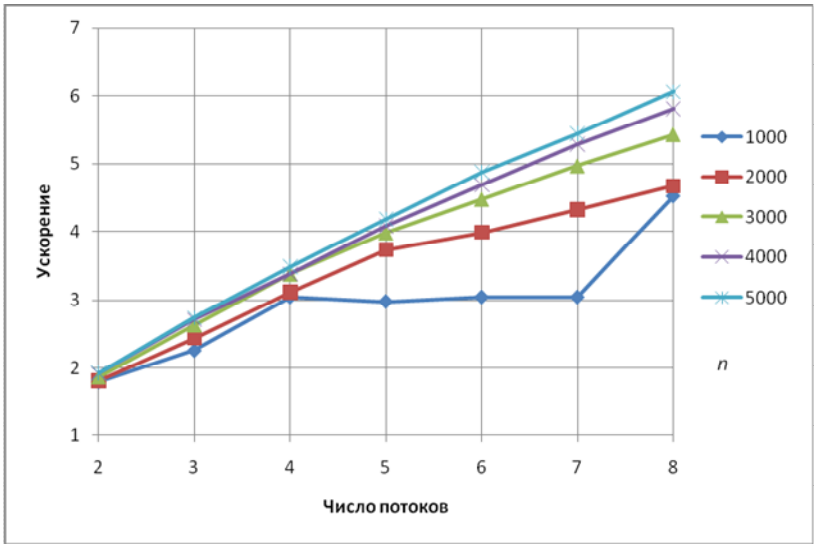


Рис. 1.12. Зависимость ускорения от числа потоков
(блочное разложение Холецкого)

Как показывают эксперименты, блочный алгоритм хорошо масштабируется – ускорения достигает значения 6 для 8-и поточной программы.

2. Итерационные методы решения СЛАУ

В данном разделе мы рассмотрим методы решения систем линейных уравнений, принципиально отличающиеся от прямых методов поиска точного решения, рассмотренных в предыдущем разделе. Пусть

$$Ax=b \tag{2.1}$$

– система линейных уравнений относительно неизвестного вектора $x\in R^n$ с симметричной положительно определенной матрицей A размерности $n\times n$ и правой частью вектором $b\in R^n$. Точное решение системы (2.1) обозначим через x^*.

Итерационный метод, предназначенный для решения системы (2.1), генерирует последовательность векторов $x^{(s)}\in R^n$, $s=0,1,2,\ldots$, каждый из которых может рассматриваться исследователем как приближенное решение системы (2.1). Начальное приближение – вектор $x^{(0)}\in R^n$ – задает исследователь. Итерационный метод называется *сходящимся*, если для любого начального приближения $x^{(0)}\in R^n$ последовательность $x^{(s)}\in R^n$, $s=0,1,2,\ldots$, сходится к точному решению x^*, т.е.

$$\lim_{s\to\infty}\left\|x^{(s)}-x^*\right\|=0 \tag{2.2}$$

где через $\|\ \|$ обозначена любая векторная норма.

На практике используют два критерия остановки итерационных методов одновременно: остановку по точности и остановку по числу итераций. Первый критерий определяется условием

$$\left\|x^{(s)}-x^{(s-1)}\right\|<\varepsilon_1 \tag{2.3}$$

где $x^{(s)}$ – приближение, полученное на итерации с номером s, $x^{(s-1)}$ – приближение, полученное на предыдущей итерации. Параметр *точности метода* ε_1 задает исследователь. Если условие (2.3) выполнено, метод завершает работу и вектор $x^{(s)}$ трактуется как приближенное решение системы (2.1). Величина ε_2, определяемая как

$$\varepsilon_2=\left\|x^{(s)}-x^{(s-1)}\right\| \tag{2.4}$$

называется *достигнутой точностью метода* и сообщается исследователю.

Второй критерий определяется максимальным числом итераций N, на которое готов пойти исследователь. Если номер итерации, которую предсто-

ит выполнить, превышает N, итерация не выполняется, метод завершает работу, вектор $x^{(N)}$ трактуется как приближенное решение системы и достигнутая точность метода ε_2 также сообщается исследователю.

При изучении сходимости итерационных методов и при их реализации в качестве векторной нормы $\| \ \|$ обычно используют евклидову норму $\| \ \|_2$, норму $\| \ \|_1$, определяемую суммой модулей всех компонент вектора, норму $\| \ \|_\infty$, определяемую максимальным модулем компоненты, а также энергетическую норму $\| \ \|_A$, порождаемую симметричной, положительно определенной матрицей A. Так как в конечномерных нормированных пространствах указанные нормы эквивалентны, для доказательства сходимости метода или для его реализации можно пользоваться любой из перечисленных норм.

2.1. Метод простой итерации

2.1.1. Последовательный алгоритм

Метод простой итерации относится к классу явных стационарных одношаговых итерационных методов решения линейных систем вида $Ax=b$ с симметричной положительно определенной матрицей A. Метод определяется формулой

$$\frac{x^{(s+1)} - x^{(s)}}{\tau} + Ax^{(s)} = b,$$

где $x^{(s)}$ – текущее приближение, $x^{(s+1)}$ – следующее приближение, $\tau \neq 0$ – фиксированное число, являющееся параметром метода.

Формула для вычисления нового приближения по предыдущему может *быть записана в виде*

$$x^{(s+1)} = -\tau(Ax^{(s)} - b) + x^{(s)} = -\tau \cdot r^{(s)} + x^{(s)},$$

где $r^{(s)} = Ax^{(s)} - b$ *– невязка* приближения с номером s.

Далее нетрудно записать явные формулы для отыскания компонент нового вектора $x^{(s+1)}$:

$$x_i^{(s+1)} = -\tau\left(\sum_{j=1}^{n} a_{ij}x_j^{(s)} - b_i\right) + x_i^{(s)}.$$

Справедлива следующая теорема о сходимости: если матрица A симметрична и положительно определена и параметр метода τ лежит в интервале $(0,\lambda_{max})$, где λ_{max} – максимальное собственное число матрицы A, метод схо-

дится к точному решению системы (2.1) с любого начального приближения. Известно, что оптимальным значением параметра τ является значение

$$\tau^* = \frac{2}{\lambda_{\min} + \lambda_{\max}},$$

где λ_{min} и λ_{max} – минимальное и максимальное собственные числа матрицы A соответственно.

Напомним, что погрешностью приближения с номером s называют вектор $z^{(s)}$, определяемый как

$$z^{(s)} = x^* - x^{(s)},$$

где x^* – точное решение системы $Ax=b$. Для метода простой итерации с оптимальным параметром справедлива следующая оценка:

$$\left\| z^{(s+1)} \right\|_2 \le \left(\frac{\mu_A - 1}{\mu_A + 1} \right)^{s+1} \left\| z^{(0)} \right\|_2 .$$

Здесь через $\| \|_2$ обозначена среднеквадратичная норма вектора, а через μ_A – число обусловленности матрицы A, согласованное с векторной нормой $\| \|_2$ и в силу этого вычисляемое как $\mu_A=\lambda_{max}/\lambda_{min}$.

Оценим трудоемкость предложенного алгоритма. Анализ расчетных формул показывает, что они включают одну операцию умножения матрицы на вектор и три операции над векторами (сложение, вычитание, умножение на константу). Общее количество операций, выполняемых на одной итерации, составляет

$$t_1=2n(n+1).$$

Таким образом, выполнение L итераций метода потребует

$$T_1=2n(n+1)L$$

операций.

2.1.2. Параллельный алгоритм

При распараллеливании итерационных методов линейной алгебры (в частности, метода простой итерации) в первую очередь следует учесть, что выполнение итераций метода осуществляется последовательно и, тем самым, наиболее целесообразный подход состоит в распараллеливании вычислений, реализуемых в ходе выполнения итераций.

Анализ последовательного алгоритма показывает, что основные затраты на s-й итерации – порядка $O(n^2)$ операций – состоят в умножении матрицы A на вектор текущего приближения $x^{(s)}$. Как результат, при организации параллельных вычислений могут быть использованы известные методы параллельного умножения матрицы на вектор (например, параллельный алгоритм матрично-векторного умножения при ленточном горизонтальном разделении матрицы). Дополнительные вычисления, имеющие порядок сложности $O(n)$, представляют собой операции сложения и умножение на скаляр для векторов. Организация таких вычислений также может быть выполнена в многопоточном режиме.

Несмотря на простоту распараллеливания, данный метод редко применяется, т.к. он обладает существенно меньшей скоростью сходимости по сравнению с иными методами, к рассмотрению которых мы переходим.

2.2. Метод верхней релаксации

Рассмотрим систему (2.1) с симметричной, положительно определенной матрицей A размера $n \times n$. Обозначим через D диагональную матрицу $n \times n$ такую, что ее главная диагональ совпадает с главной диагональю матрицы A. Через L обозначим нижнюю треугольную матрицу $n \times n$ такую, что ее ненулевые (поддиагональные) элементы также совпадают с элементами A, а главная диагональ является нулевой. Аналогично обозначим через R верхнюю треугольную матрицу $n \times n$, ненулевые (наддиагональные) элементы которой совпадают с элементами A, а главная диагональ также является нулевой. В этом случае для A справедливо представление в виде

$$A=L+D+R. \tag{2.5}$$

Метод верхней релаксации является представителем стационарных одношаговых итерационных методов линейной алгебры и записывается в виде

$$\frac{(D+\omega L)(x^{(s+1)}-x^{(s)})}{\omega}+Ax^{(s)}=b. \tag{2.6}$$

Здесь $x^{(s)}$ – приближение, полученное на итерации с номером s, $x^{(s+1)}$ – следующее приближение, ω – число (параметр метода), матрицы A, L, D и вектор b определены выше.

Необходимым условием сходимости метода релаксации с любого начального приближения $x^{(0)}$ к точному решению задачи x^* является выполнение условия $\omega \in (0,2)$. Если же матрица A симметрична и положительно определена, то выполнение данного условия является также и достаточным. При этом если $\omega \in (0,1)$, то говорят о *методе нижней релаксации*, а при

$\omega\in(1,2)$ – о *методе верхней релаксации*, при ω=1 метод релаксации будет совпадать с известным *методом Зейделя*.

Скорость сходимости метода верхней релаксации определяется выбором параметра ω. Известно [4], что при решении некоторых классов разреженных систем уравнений, метод Зейделя требует $O(n^2)$ итераций, а при надлежащем выборе итерационного параметра ω метод будет сходиться за $O(n)$ итераций.

В общем случае нет аналитической формулы для вычисления оптимального параметра ω_{opt}, обеспечивающего наилучшую сходимость. Например, для решения систем уравнений, возникающих при аппроксимации дифференциальных уравнений в частных производных, можно использовать эвристические оценки вида

$$\omega_{opt} \approx 2 - O(h) .$$

где h – шаг сетки, на которой проводилась дискретизация.

В некоторых случаях можно более точно оценить оптимальный параметр. Известной оценкой [17] является выражение

$$\omega_{opt} = \frac{2}{1+\sqrt{1-\rho^2\left(D^{-1}(R+L)\right)}},$$

где ρ – спектральный радиус матрицы, а R, D, L – из (2.5). Возникающий при этом вопрос об оценке спектрального радиуса может быть решен численно, например, при помощи степенного метода [2].

Также следует отметить, что для ряда задач, возникающих при решении задач математической физики методом сеток, оптимальные параметры метода верхней релаксации найдены аналитически (подробнее об этом см. в п. 4.3).

2.2.1. Последовательный алгоритм

Получим формулы для отыскания $x^{(s+1)}$ по предыдущему приближению $x^{(s)}$ в явном виде.

$$(D+\omega L)(x^{(s+1)} - x^{(s)}) + \omega A x^{(s)} = \omega b ,$$

$$Dx^{(s+1)} + \omega L x^{(s+1)} - Dx^{(s)} - \omega L x^{(s)} + \omega A x^{(s)} = \omega b ,$$

$$Dx^{(s+1)} = -\omega L x^{(s+1)} + Dx^{(s)} - \omega(A-L)x^{(s)} + \omega b .$$

С учетом того, что $A–L=R+D$, получаем

$$Dx^{(s+1)} = -\omega Lx^{(s+1)} + (1-\omega)Dx^{(s)} - \omega Rx^{(s)} + \omega b\,.$$

Далее нетрудно записать явные формулы для отыскания компонент нового вектора $x^{(s+1)}$:

$$a_{ii}x_i^{(s+1)} = -\omega\sum_{j=1}^{i-1} a_{ij}x_j^{(s+1)} + (1-\omega)a_{ii}x_i^{(s)} - \omega\sum_{j=i+1}^{n} a_{ij}x_j^{(s)} + \omega b_i \qquad (2.7)$$

Как следует из формулы (2.7), при подсчете i-й компоненты нового приближения все компоненты, индекс которых меньше i, берутся из нового приближения $x^{(s+1)}$, а все компоненты, индекс которых больше либо равен i – из старого приближения $x^{(s)}$. Таким образом, после того, как i-я компонента нового приближения вычислена, i-я компонента старого приближения нигде использоваться не будет. Напротив, для подсчета следующих компонент вектора $x^{(s+1)}$ компоненты с индексом, меньшим или равным i, будут использоваться «в новой версии». В силу этого обстоятельства для реализации метода достаточно хранить только одно (текущее) приближение $x^{(s)}$, а при расчете следующего приближения $x^{(s+1)}$ использовать формулу (2.7) для всех компонент по порядку и постепенно обновлять вектор $x^{(s)}$.

2.2.2. Организация параллельных вычислений

При распараллеливании метода верхней релаксации также применим подход, который состоит в распараллеливании вычислений, реализуемых в ходе выполнения итераций.

Анализ последовательного алгоритма показывает, что основные затраты на s-й итерации – порядка $O(n^2)$ операций – заключаются в операциях матричного умножения $Lx^{(s+1)}$ и $Rx^{(s+1)}$. Однако напрямую распараллелить эти операции не представляется возможным, т.к. в методе верхней релаксации не только итерации осуществляются последовательно, но и вычисление компонент вектора очередного приближения $x^{(s+1)}$ также осуществляется последовательно, начиная с первой.

Применение ленточного разделения данных, аналогично методу простой итерации, приведет к изменению вычислительной схемы алгоритма. Поэтому одним из возможных способов распараллеливания, сохраняющем в точности последовательность действий метода, состоит в распараллеливании операций, необходимых для получения одной компоненты вектора нового приближения. При этом распараллелить можно вычисление сумм в формуле (2.7).

2.2.3. Результаты вычислительных экспериментов

Вычислительные эксперименты для оценки эффективности параллельного варианта метода верхней релаксации проводились при условиях, указанных во введении. С целью формирования симметричной положительно определенной матрицы элементы подматрицы L генерировались в диапазоне от 0 до 1, значения элементов подматрицы R получались из симметрии матриц L и R, а элементы на главной диагонали (подматрица D) генерировались в диапазоне от n до $2n$, где n – размер матрицы.

В качестве критерия остановки использовался критерий остановки по точности (2.3) с параметром $\varepsilon=10^{-6}$; а итерационный параметр $\omega=1.1$. Во всех экспериментах метод нашел решение с требуемой точностью за 11 итераций. Как и для предыдущих экспериментов, ускорение будем фиксировать по сравнению с параллельной программой, запущенной в один поток.

Табл. 2.1. Результаты экспериментов (метод верхней релаксации)

n	1 поток	Параллельный алгоритм							
		2 потока		4 потока		6 потоков		8 потоков	
		T	S	T	S	T	S	T	S
2500	0,73	0,47	1,57	0,30	2,48	0,25	2,93	0,22	3,35
5000	3,25	2,11	1,54	1,22	2,67	0,98	3,30	0,80	4,08
7500	7,72	5,05	1,53	3,18	2,43	2,36	3,28	1,84	4,19
10000	14,60	9,77	1,50	5,94	2,46	4,52	3,23	3,56	4,10
12500	25,54	17,63	1,45	10,44	2,45	7,35	3,48	5,79	4,41
15000	38,64	26,36	1,47	15,32	2,52	10,84	3,56	8,50	4,54

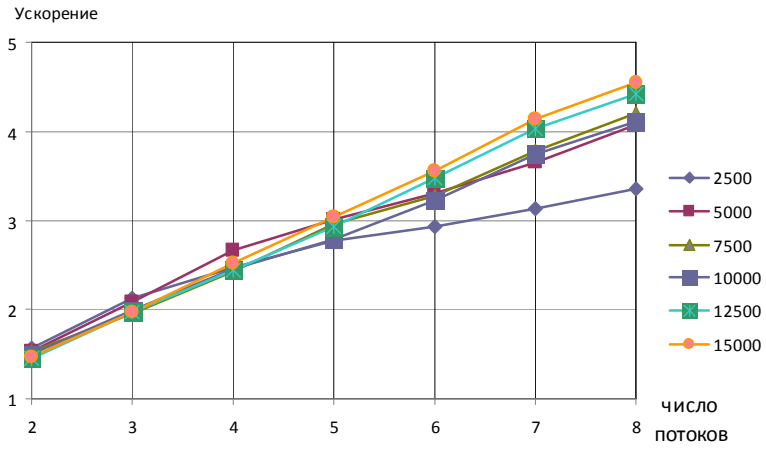


Рис. 2.1. Ускорение параллельного метода верхней релаксации

Эксперименты демонстрируют неплохое ускорение (порядка 4 на 8-и потоках).

2.3. Метод сопряженных градиентов

Рассмотрим систему линейных уравнений (2.1) с симметричной, положительно определенной матрицей A размера $n \times n$. Основой *метода сопряженных градиентов* является следующее свойство: решение системы линейных уравнений (2.1) с симметричной положительно определенной матрицей A эквивалентно решению задачи минимизации функции

$$F(x) = \frac{1}{2}(Ax, x) - (b, x). \tag{2.8}$$

в пространстве R^n. В самом деле, функция $F(x)$ достигает своего минимального значения тогда и только тогда, когда ее градиент

$$\nabla F(x) = Ax - b \tag{2.9}$$

обращается в ноль. Таким образом, решение системы (2.1) можно искать как решение задачи безусловной минимизации (2.8).

2.3.1. Последовательный алгоритм

С целью решения задачи минимизации (2.8) организуется следующий итерационный процесс.

Подготовительный шаг (s=0) определяется формулами

$$x^{(1)} = x^{(0)} + \alpha_0 h^{(0)},\ r^{(0)} = h^{(0)} = b - Ax^{(0)}.$$

где $x^{(0)}$ – произвольное начальное приближение; а коэффициент α_0 вычисляется как

$$\alpha_0 = \frac{(r^{(0)}, r^{(0)})}{(Ah^{(0)}, h^{(0)})}$$

Основные шаги (s=1, 2,…, *n*–1) определяются формулами

$$\alpha_s = \frac{(r^{(s)}, r^{(s)})}{(Ah^{(s)}, h^{(s)})},\quad r^{(s+1)} = r^{(s)} - \alpha_s Ah^{(s)},$$

$$\beta_s = \frac{(r^{(s+1)}, r^{(s+1)})}{(r^{(s)}, r^{(s)})},\quad h^{(s+1)} = r^{(s+1)} + \beta_s h^{(s)},$$

$$x^{(s+1)} = x^{(s)} + \alpha_s h^{(s)}.$$

Здесь $r^{(s)} = b - Ax^{(s)}$ – невязка *s*-го приближения, коэффициент β_s находят из условия сопряженности

$$(Ah^{(s)}, h^{(s-1)}) = 0$$

направлений $h^{(s)}$ и $h^{(s-1)}$; а α_s является решением задачи минимизации функции *F* по направлению h_s

$$F(x_s + \alpha h_s) \to \min.$$

Анализ расчетных формул метода показывает, что они включают две операции умножения матрицы на вектор, четыре операции скалярного произведения и пять операций над векторами. Однако на каждой итерации произведение $Ah^{(s)}$ достаточно вычислить один раз, а затем использовать сохраненный результат. Общее количество числа операций, выполняемых на одной итерации, составляет

$$t_1 = 2n^2 + 13n.$$

Таким образом, выполнение *L* итераций метода потребует

$$T_1 = L(2n^2 + 13n) \qquad (2.10)$$

операций. Можно показать, что для нахождения точного решения системы линейных уравнений с положительно определенной симметричной матрицей необходимо выполнить не более *n* итераций, тем самым, сложность

алгоритма поиска точного решения имеет порядок $O(n^3)$. Однако ввиду ошибок округления данный процесс обычно рассматривают как итерационный, процесс завершается либо при выполнении обычного условия остановки (2.3) , либо при выполнении условия малости относительной нормы невязки

$$\left\| r^{(k)} \right\| / \| b \| \le \varepsilon \ .$$

2.3.2. Организация параллельных вычислений

При разработке параллельного варианта метода сопряженных градиентов для решения систем линейных уравнений в первую очередь следует учесть, что выполнение итераций метода осуществляется последовательно и, тем самым, наиболее целесообразный подход состоит в распараллеливании вычислений, реализуемых в ходе выполнения итераций.

Анализ последовательного алгоритма показывает, что основные затраты на s-й итерации состоят в умножении матрицы A на вектора h_{s-1} и h_s. Как результат, при организации параллельных вычислений могут быть использованы известные методы параллельного умножения матрицы на вектор.

Дополнительные вычисления, имеющие меньший порядок сложности, представляют собой различные операции обработки векторов (скалярное произведение, сложение и вычитание, умножение на скаляр). Организация таких вычислений, конечно же, должна быть согласована с выбранным параллельным способом выполнения операция умножения матрицы на вектор.

Выберем для дальнейшего анализа эффективности получаемых параллельных вычислений параллельный алгоритм матрично-векторного умножения при ленточном горизонтальном разделении матрицы. При этом операции над векторами, обладающие меньшей вычислительной трудоемкостью, также будем выполнять в многопоточном режиме.

Вычислительная трудоемкость последовательного метода сопряженных градиентов определяется соотношением (2.10). Определим время выполнения параллельной реализации метода сопряженных градиентов. Вычислительная сложность параллельной операции умножения матрицы на вектор при использовании схемы ленточного горизонтального разделения матрицы составляет

$$2n\left(2n-1\right)/p + \delta \ ,$$

где n – длина вектора, p – число потоков, δ – накладные расходы на создание и закрытие параллельной секции.

Все остальные операции над векторами (скалярное произведение, сложение, умножение на константу) могут быть выполнены в однопоточном режиме, т.к. не являются определяющими в общей трудоемкости метода. Следовательно, общая вычислительная сложность параллельного варианта метода сопряженных градиентов может быть оценена как

$$T_p = L\left(\frac{2n^2}{p} + 13n + \delta\right),$$

где L – число итераций метода.

2.3.3. Результаты вычислительных экспериментов

Вычислительные эксперименты для оценки эффективности параллельного варианта метода сопряженных градиентов для решения систем линейных уравнений с симметричной положительно определенной матрицей проводились при условиях, указанных во введении. Элементы на главной диагонали матрицы A) генерировались в диапазоне от n до $2n$, где n – размер матрицы, остальные элементы генерировались симметрично в диапазоне от 0 до 1. В качестве критерия остановки использовался критерий остановки по точности (2.3) с параметром $\varepsilon=10^{-6}$.

Результаты вычислительных экспериментов приведены в таблице 2.2 (время работы алгоритмов указано в секундах).

Табл. 2.2. Результаты экспериментов (метод сопряженных градиентов)

n	1 поток	Параллельный алгоритм							
		2 потока		4 потока		6 потоков		8 потоков	
		t	S	t	S	t	S	t	S
500	0,02	0,01	1,64	0,01	2,56	0,01	2,56	0,01	2,56
1000	0,21	0,16	1,26	0,10	2,09	0,07	2,88	0,05	3,83
1500	0,65	0,48	1,36	0,27	2,41	0,19	3,42	0,15	4,20
2000	1,32	0,94	1,41	0,53	2,51	0,38	3,48	0,29	4,50
3000	3,42	2,34	1,46	1,33	2,58	0,96	3,56	0,74	4,63
4000	6,49	4,54	1,43	2,53	2,56	1,80	3,60	1,40	4,62
5000	11,02	7,41	1,49	4,17	2,65	2,98	3,70	2,31	4,78

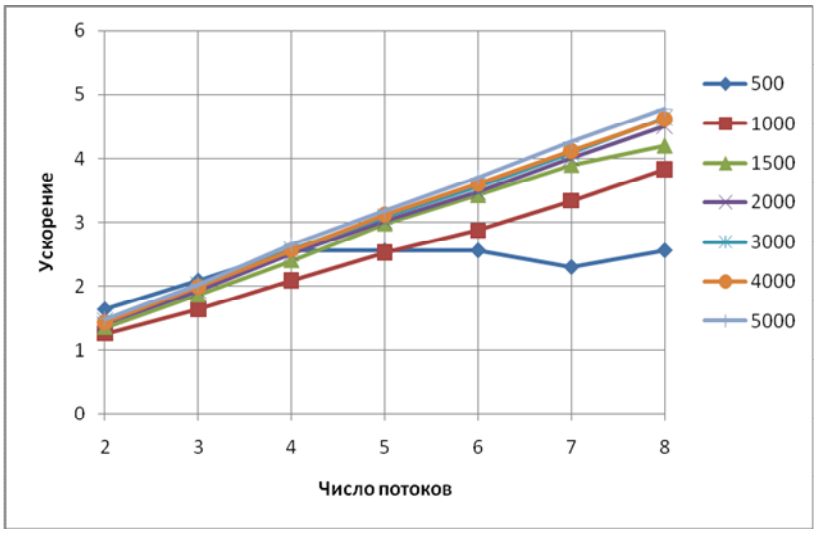


Рис. 2.2. Ускорение параллельного метода сопряженных градиентов

Так как операция умножения матрицы на вектор – основная по трудоемкости операция на одной итерации метода – распараллеливается хорошо, то и эффективность распараллеливания метода сопряженных градиентов будет линейная, что подтверждается приведенным графиком.

Спад ускорения при N=500 объясняется недостаточной вычислительной нагрузкой, которая приходится на каждый процесс (этот эффект будет проиллюстрирован и в дальнейшем при решении дифференциальных уравнений в частных производных). Использовать более чем 4 потока для решения данной задачи при $N \le 500$ – нецелесообразно.

3. Решение разреженных СЛАУ

В данном разделе будут рассмотрены вопросы, касающиеся решения СЛАУ с разреженными матрицами. В разделах 1 и 2 неявно подразумевалось, что мы работаем с плотными матрицами.

Понятие разреженной матрицы можно определить многими способами, суть которых состоит в том, что в разреженной матрице «много» нулевых элементов. Обычно говорят, что матрица *разрежена*, если она содержит $O(n)$ отличных от нуля элементов. В противном случае матрица считается *плотной*. Типичным случаем разреженности является ограниченность числа ненулевых элементов в одной строке от 1 до k, где $k<<n$. Задачи линейной алгебры с разреженными матрицами возникают во многих областях, например, при решении дифференциальных уравнений в частных производных, при решении многомерных задач локальной оптимизации. В п. 1.1 мы уже познакомились с методами решения СЛАУ с трехдиагональной матрицей, которая, являясь ленточной, относится также и к классу разреженных матриц.

Очевидно, что любую разреженную матрицу можно обрабатывать как плотную, и наоборот. При правильной реализации алгоритмов в обоих случаях будут получены правильные результаты, однако вычислительные затраты будут существенно отличаться. Поэтому приписывание матрице свойства разреженности эквивалентно утверждению о существовании алгоритма, использующего ее разреженность и делающего операции с ней эффективнее по сравнению со стандартными алгоритмами.

Многие алгоритмы, тривиальные для случая плотных матриц, в разреженном случае требуют более тщательного подхода. Во многих алгоритмах обработки разреженных матриц можно выделить два этапа: *символический* и *численный*. На символическом этапе формируется *портрет* результирующей матрицы (т.е. определяются места ненулевых элементов в структуре матрицы); на численном этапе определяются значения ненулевых элементов результирующей матрицы. В качестве примера типовых операций с разреженными матрицами в данной главе будет рассмотрен алгоритм умножения разреженной матрицы на плотный вектор, а также круг вопросов, возникающих при использовании метода Холецкого для решения систем линейных уравнений с разреженной матрицей.

3.1. Хранение разреженной матрицы

Существуют различные форматы хранения разреженных матриц. Одни предназначены для хранения матриц специального вида (например, ленточных), другие обеспечивают работу с матрицами общего вида. Ниже рассмотрим некоторые весьма распространенные способы представления разреженных матриц, информацию о других способах можно найти, например, в [10].

По-видимому, наиболее очевидным способом хранения произвольной разреженной матрицы является *координатный формат*: хранятся только ненулевые элементы матрицы, и их координаты (номера строк и столбцов). При данном подходе хранение матрицы *A* можно обеспечить в трех одномерных массивах:

- массив ненулевых элементов матрицы *A* (обозначим его как *values*);
- массив номеров строк матрицы *A*, соответствующих элементам массива *values* (обозначим его как *rows*);
- массив номеров столбцов матрицы *A*, соответствующих элементам массива *values* (обозначим его как *cols*);

Данный способ представления называют *полным*, поскольку представлена вся матрица A, и *неупорядоченным*, поскольку элементы матрицы могут храниться в произвольном порядке.

В качестве примера рассмотрим разреженную матрицу

$$A = \begin{bmatrix} 1 & -1 & 0 & -3 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}, \qquad (3.1)$$

которая может быть представлена в координатном формате как

values=(1, –1, –3, –2, 5, 4, 6, 4, –4, 2, 7, 8, –5);

rows=(1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5);

cols=(1, 2, 4, 1, 2, 3, 4, 5, 1, 3, 4, 2, 5).

Хотя многие математические библиотеки поддерживают матрично-векторные операции в координатном формате, данный формат обеспечивает медленный доступ к элементам матрицы, и является затратным по используемой памяти. В рассмотренном выше примере избыточность по па-

мяти образом проявляется в массиве *rows*, в котором строчные координаты хранятся неоптимальным образом.

Перейдем далее к рассмотрению более экономных форматов хранения. *Разреженный строчный формат* – это одна из наиболее широко используемых схем хранения разреженных матриц. Эта схема предъявляет минимальные требования к памяти и в то же время оказывается очень удобной для нескольких важных операций над разреженными матрицами: сложения, умножения, перестановок строк и столбцов, транспонирования, решения линейных систем с разреженными матрицами коэффициентов как прямыми, так и итерационными методами и т. д.

В соответствии с рассматриваемой схемой для хранения матрицы A требуется три одномерных массива:

- массив ненулевых элементов матрицы A, в котором они перечислены по строкам от первой до последней (обозначим его опять как *values*);
- массив номеров столбцов для соответствующих элементов массива *values* (обозначим его как *cols*);
- массив указателей позиций, с которых начинается описание очередной строки (обозначим его *pointer*). Описание k-й строки хранится в позициях с *pointer*[k]-й по (*pointer*[k+1]–1)-ю массивов *values* и *cols*. Если *pointer*[k]=*pointer*[k+1], то k-я строка пустая. Если матрица A состоит из n строк, то длина массива pointer будет n+1.

Данный способ представления также является полным, и *упорядоченным*, поскольку элементы каждой строки хранятся в соответствии с возрастанием столбцовых индексов.

Для примера рассмотрим представление матрицы (3.1) в разреженном строчном формате:

$$values=(1, -1, -3, -2, 5, 4, 6, 4, -4, 2, 7, 8, -5);$$

$$cols=(1, 2, 4, 1, 2, 3, 4, 5, 1, 3, 4, 2, 5);$$

$$pointer=(1, 4, 6, 9, 12, 14).$$

Очевидно, что объем памяти, требуемый для хранения вектора *pointer*, значительно меньше, чем для хранения вектора *rows*. Более того, разреженный строчный формат обеспечивает эффективный доступ к строчкам матрицы; доступ к столбцам по прежнему затруднен. Поэтому предпочтительно использовать этот способ хранения в тех алгоритмах, в которых преобладают строчные операции.

Иногда бывает удобно использовано *полный неупорядоченный* способ хранения, при котором внутри каждой строки элементы могут храниться в произвольном порядке. Результаты многих матричных операций получаются неупорядоченными, и упорядочивание может быть весьма затратным. В то же время, многие алгоритмы для разреженных матриц не требуют, чтобы представление было упорядоченным.

После рассмотрения строчного формата хранения очевидным является и *разреженный столбцовый формат*. В этом случае ненулевые элементы матрицы *A* перечисляются в порядке их появления в столбцах матрицы, а не в строках. Все ненулевые элементы хранятся по столбцам в массиве *values*; индексы строк ненулевых элементов – в массиве *rows*; элементы массива *pointer* указывают на позиции, с которых начинается описание очередного столбца.

Столбцовые представления могут рассматриваться как строчные представления транспонированных матриц. Разреженный столбцовый формат обеспечивает эффективный доступ к столбцам матрицы; доступ к строкам затруднен. Поэтому предпочтительно использовать этот способ хранения в тех алгоритмах, в которых преобладают столбцовые операции.

В случае если обрабатываемая матрица симметрична, достаточно хранить лишь ее верхнюю треугольную подматрицу. При этом для хранения можно использовать любой из рассмотренных форматов. Например, симметричная матрица

$$A=\begin{bmatrix} 1 & -1 & 0 & -3 & 0 \\ -1 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -3 & 0 & 6 & 7 & 0 \\ 0 & 0 & 4 & 0 & -5 \end{bmatrix}$$

может быть представлена в разреженном строчном формате как

values=(1, –1, –3, 5, 4, 6, 4, 7, –5);

cols=(1, 2, 4, 2, 3, 4, 5, 4, 5);

pointer=(1, 4, 5, 8, 9, 10).

Если большинство диагональных элементов заданной симметричной матрицы отличны от нуля (например, у симметричной положительно определенной матрицы все диагональные элементы положительны), то они могут храниться в отдельном массиве *BD*, а разреженным форматом представляется только верхний треугольник *B*.

Завершая обзор форматов хранения разреженных матриц, можно еще раз отметить, что выбор способа хранения матрицы полностью определяется алгоритмами, которые планируется в дальнейшем использовать для обработки данной матрицы.

3.2. Операции над разреженными матрицами

Реализация матричных операций является тривиальной в случае плотной или ленточной матрицы, но не столь очевидна для разреженных матриц, хранящихся в одном из экономичных форматов (здесь и далее мы будем рассматривать разреженный строчный формат хранения матрицы).

3.2.1. Умножение матрицы на вектор

Рассмотрим операцию умножения разреженной матрицы на плотный вектор. Результатом этой операции будет заполненный вектор, а не разреженная матрица. Поэтому в алгоритмах умножения вычисления выполняются прямо, без символического этапа.

Среди алгоритмов, в которых встречается данная операция, можно отметить итерационные методы решения систем линейных уравнений (см. п. 2). Достоинство данных методов, с вычислительной точки зрения, состоит в том, что единственная требуемая матричная операция, – это повторное умножение матрицы на последовательность заполненных векторов; сама матрица не меняется.

Итак, рассмотрим умножение разреженной матрицы общего вида, хранимой в строчном формате посредством массивов *values*, *cols*, *pointer* на заполненный вектор-столбец *b*, хранимый в одномерном массиве. Результатом будет новый заполненный вектор

$$c=Ab,$$

также размещаемый в одномерном массиве.

Пусть n – число строк матрицы. Порядок, в котором накапливаются скалярные произведения, определяется порядком хранения элементов матрицы. Для каждой ее строки i мы находим с помощью массива индексов *pointer* значения первой *pointer*[i] и последней *pointer*[i+1]–1 позиций, занимаемых элементами строки i в массивах *values* и *cols*. Затем, чтобы вычислить скалярное произведение строки i и вектора *b*, мы просто просматриваем *values* и *cols* на отрезке от *pointer*[i] до *pointer*[i+1]–1; каждое значение, хранимое в *cols*[*pointer*[j]], есть столбцовый индекс и используется для извлечения из массива *b* элемента, который должен быть умножен на

соответствующее число из массива *values*. Результат каждого умножения прибавляется к *c*[*i*].

3.2.2. Параллельный алгоритм

Рассмотрим теперь параллельный алгоритм умножения разреженной матрицы на плотный вектор, матрица представлена в строчном формате. При таком способе представления данных в качестве базовой подзадачи может быть выбрана операция скалярного умножения одной строки матрицы на вектор. После завершения вычислений каждая базовая подзадача определяет один из элементов вектора результата *c*.

Как правило, количество элементов в одной строке разреженной матрицы размера $n \times n$ ограничено некоторой константой k, $k \ll n$. Значит, в процессе умножения такой матрицы на вектор количество вычислительных операций для получения скалярного произведения примерно одинаково для всех базовых подзадач. При использовании систем с общей памятью число потоков p будет меньше числа базовых подзадач n, и мы можем объединить базовые подзадачи таким образом, чтобы каждый поток выполнял несколько таких задач, соответствующих непрерывной последовательности строк матрицы A. В этом случае по окончании вычислений каждая базовая подзадача определяет набор элементов результирующего вектора c. Распределение подзадач между потоками может быть выполнено произвольным образом.

3.3. Метод Холецкого для разреженных матриц

В п. 1.3 был рассмотрен метод Холецкого для решения системы линейных уравнений с симметричной положительно определенной матрицей

$$Ax=b, \tag{3.2}$$

при этом подразумевалось, что матрица A – плотная. Данный алгоритм, естественно, будет применим и для разреженной матрицы A, однако в этом случае возникает ряд важных моментов, к обсуждению которых мы сейчас и приступаем.

В качестве примера рассмотрим систему уравнений

$$\begin{bmatrix} 4 & 1 & 2 & 0.5 & 2 \\ 1 & 0.5 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 \\ 0.5 & 0 & 0 & 0.625 & 0 \\ 2 & 0 & 0 & 0 & 16 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 17 \\ 3 \\ 7 \\ 6 \\ 12 \end{bmatrix} \tag{3.3}$$

Используя расчетные формулы метода (1.27) и (1.28), можно вычислить фактор Холецкого для данной задачи

$$L = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 \\ 0.25 & -0.25 & -0.5 & 0.5 & 0 \\ 1 & -1 & -2 & -3 & 1 \end{bmatrix},$$

а затем, выполнив обратный ход алгоритма по формулам (1.29), (1.30), найти искомое решение

$$x^T=[2, 2, 1, 8, 0.5].$$

Этот пример иллюстрирует наиболее важный факт, относящийся к применению метода Холецкого для разреженных матриц: матрица обычно претерпевает *заполнение*. Это значит, что в матрице L появляются ненулевые элементы в позициях, где в нижней треугольной части A стояли нули.

Предположим теперь, что мы перенумеровали переменные в соответствии с правилом $y_i=x_{i+1}$, $i=1, 2,..., 4$, $y_5=x_1$ и переупорядочили уравнения так, чтобы первое стало последним, а остальные сдвинулись бы на единицу вверх. Тогда мы получим эквивалентную систему уравнений

$$\begin{bmatrix} 0.5 & 0 & 0 & 0 & 1 \\ 0 & 3 & 0 & 0 & 2 \\ 0 & 0 & 0.625 & 0 & 0.5 \\ 0 & 0 & 0 & 16 & 2 \\ 1 & 2 & 0.5 & 2 & 4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \\ 6 \\ 12 \\ 17 \end{bmatrix} \tag{3.4}$$

Очевидно, что эта перенумерация переменных и переупорядочение уравнений равносильны симметричной перестановке строк и столбцов матрицы A, причем та же перестановка применяется и к вектору b. Решение новой системы есть переупорядоченный вектор x. Указанную перенумерацию

легко записать в виде произведения матицы A на матрицу перестановки P. Напомним, что матрица P называется матрицей перестановки, если в каждой строке и столбце матрицы находится лишь один единичный элемент. В нашем примере

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{3.5}$$

а матрицу системы (3.4) можно получить как

$$\overline{A} = PAP^T .$$

При этом связь с исходной постановкой задачи выражается соотношением

$$(PAP^T)(Px) = Pb .$$

Ниже приведен фактор Холецкого для новой системы уравнений (с точностью до 10^{-3})

$$L = \begin{bmatrix} 0.707 & 0 & 0 & 0 & 0 \\ 0 & 1.732 & 0 & 0 & 0 \\ 0 & 0 & 0.79 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 1.414 & 1.154 & 0.632 & 0.5 & 0.129 \end{bmatrix}.$$

Выполнив обратный ход, можно получить переупорядоченный вектор решения

$$y^T=[2, 1, 8, 0.5, 2].$$

Важный момент состоит в том, что переупорядочение уравнений и переменных привело к треугольному множителю L, который разрежен в точности в той мере, что и нижний треугольник A.

Задача нахождения перестановки, минимизирующей заполнение при вычислении фактора Холецкого, является *NP*-трудной (т.к. вообще говоря, минимум должен вычисляться по всем $n!$ перестановкам). Поэтому на практике используют эвристические алгоритмы, которые хотя и не гарантируют, что находят оптимальную перестановку, но, как правило, дают приемлемый результат.

Таким образом, при решении разреженной системы методом Холецкого можно выделить следующие этапы:

1. *Переупорядочивание* – вычисление матрицы перестановки P;
2. *Символическое разложение* – построение портрета матрицы L и выделение памяти под хранение ненулевых элементов;
3. *Численное разложение* – вычисление значений матрицы L и размещение их в выделенной памяти.
4. *Обратный ход* – решение двух треугольных систем уравнений.

Этапы 1 и 2 специфичны именно для разреженных матриц, тогда как этапы 3 и 4 выполняются для любых задач.

Ниже мы рассмотрим два известных алгоритма определения матрицы перестановки – методы *минимальной степени* и *вложенных сечений*.

3.3.1. Метод минимальной степени

Для формулирования метода минимальной степени нам потребуются некоторые сведения из теории графов.

Рассмотрим неориентированный граф G. Формально граф G можно рассматривать как упорядоченную пару двух множеств

$$G=(V, E),$$

где $V=\{1,...,n\}$ – множество вершин, $E=\{(e_1,e_2)\subseteq V\times V\}$ – множество связей между вершинами (ребер). Так как рассматриваемый граф – *неориентированный*, то множество E удовлетворяет свойству

$$(e_1,e_2)\in E\leftrightarrow(e_2,e_1)\in E.$$

Пара вершин $x\in V$, $y\in V$ называется *смежной*, если существует ребро, их соединяющее, т.е.

$$(x,y)\in E.$$

Для любой вершины $x\in V$ можно определить множество $Adj(x)$ смежных ей вершин как

$$Adj(x)=\{y\in V:(x,y)\in E\}.$$

Мощность множества $Adj(x)$ называется *степенью* вершины

$$deg(x)=|Adj(x)|.$$

Граф, каждые две вершины которого соединены ребром, называется *полным*.

Подграф исходного графа – граф, содержащий некое подмножество вершин данного графа и некое подмножество инцидентных им рёбер.

Кликой в неориентированном графе называется подмножество вершин, каждые две из которых соединены ребром графа. Иными словами, это полный подграф первоначального графа.

Одной из форм задания графа $G=(V, E)$ является *матрица смежности* $A=(a_{ij})$ размера $n \times n$ такая, что

$$a_{ij} = \begin{cases} \neq 0, (i, j) \in E, \\ 0, \quad (i, j) \notin E. \end{cases}$$

В нашем случае матрицу A системы линейных уравнений (3.2) можно рассмотреть как матрицу смежности соответствующего графа G, за исключением диагональных элементов, которым будут соответствовать петли. На рис. 3.1 приведен граф, соответствующий матрице из примера (3.3).

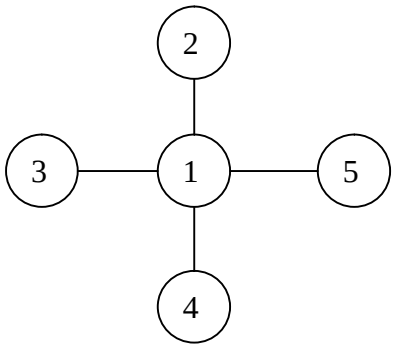
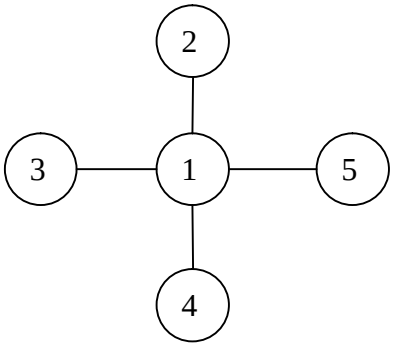


Рис. 3.1. Пример графа матрицы

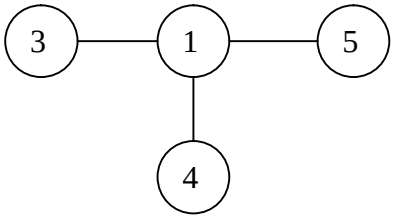
После того, как введены все необходимые определения, можно перейти к описанию алгоритма минимальной степени.

Пусть исходной матрице A из (3.2) соответствует граф G. Алгоритм минимальной степени строит последовательность графов исключения G_i, каждый их которых получен из предыдущего удалением вершины с минимальной степенью и созданием клики между всеми вершинами, которые были смежными с удаленной. В случае, когда вершин с минимальной степенью несколько, выбирается любая. Алгоритм продолжается до тех пор, пока в очередном графе есть вершины. По мере удаления вершин, их номер записывается в перестановку π, по которой впоследствии строится матрица перестановки P.

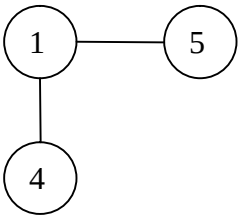
В качестве примера работы алгоритма на рис. 3.2 приведена последовательность графов исключения и соответствующая перестановка π, порождаемая алгоритмом минимальной степени для задачи (3.3).



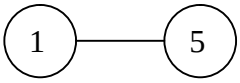
$G_0, \pi=\{\varnothing\}$



$G_1, \pi=\{2\}$



G_2, $\pi=\{2,3\}$



G_3, $\pi=\{2,3,4\}$

G_4, $\pi=\{2,3,4,1\}$ $G_5=\varnothing$, $\pi=\{2,3,4,1,5\}$

Рис. 3.2. Последовательность графов исключения

Полученная перестановка $\pi=\{2,3,4,1,5\}$ соответствует матрице

$$P=\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

которая отличается от уже рассмотренной нами перестановки (3.5). Проверим ожидаемые свойства переупорядоченной матрицы: новая матрица системы уравнений, получаемая по формуле $\overline{A} = PAP^T$, будет

$$\overline{A} = PAP^T = \begin{bmatrix} 0.5 & 0 & 0 & 1 & 0 \\ 0 & 3 & 0 & 2 & 0 \\ 0 & 0 & 0.625 & 0.5 & 0 \\ 1 & 2 & 0.5 & 4 & 2 \\ 0 & 0 & 0 & 2 & 16 \end{bmatrix} \tag{3.6}$$

Фактор Холецкого, соответствующий данной матрице, будет (с точностью до 10^{-3})

$$L = \begin{bmatrix} 0.707 & 0 & 0 & 0 & 0 \\ 0 & 1.732 & 0 & 0 & 0 \\ 0 & 0 & 0.79 & 0 & 0 \\ 1.414 & 1.154 & 0.632 & 0.516 & 0 \\ 0 & 0 & 0 & 3.873 & 1 \end{bmatrix}.$$

Видно, что при факторизации матрицы $\overline{A}$ не образуется порожденных элементов. Однако это не всегда выполняется, алгоритм лишь необходимым образом уменьшает количество порожденных элементов.

3.3.2. Результаты экспериментов

В качестве вычислительного примера большого масштаба рассмотрим разложение серии матриц, доступных в коллекции http://math.nist.gov/MatrixMarket. Ниже приведено описание выбранных матриц: указан размер, коэффициент заполненности, портрет матрицы.

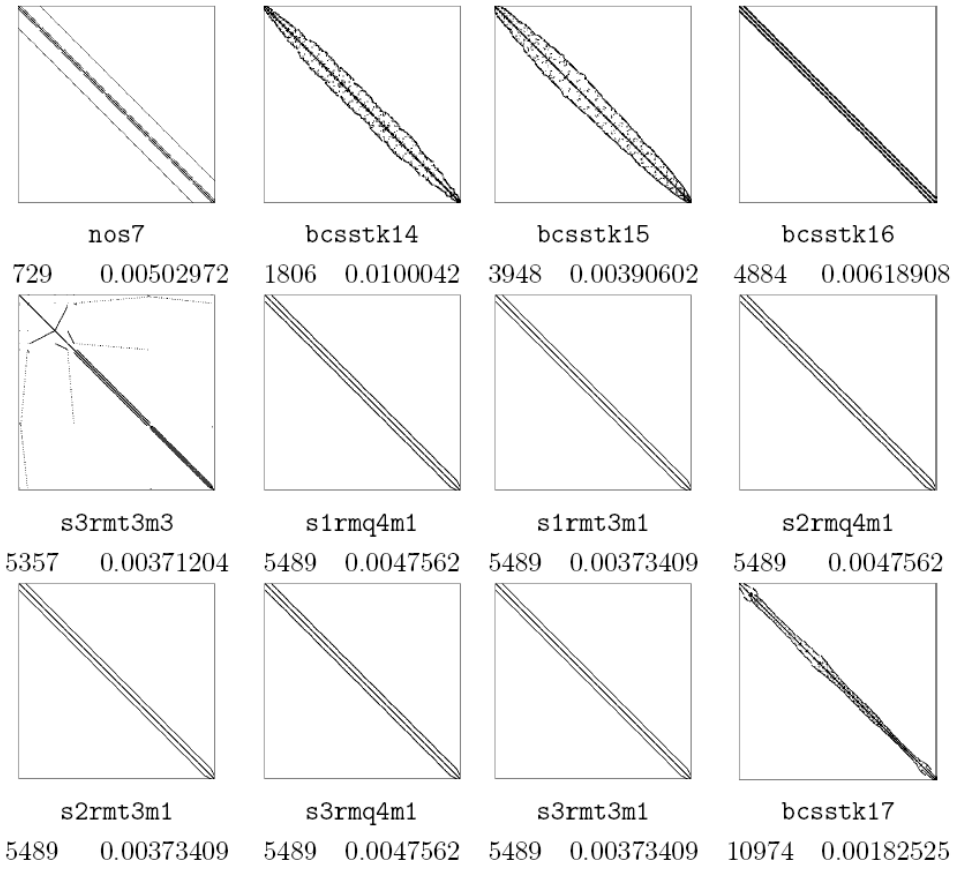


Рис. 3.3. Характеристики матриц

Каждая из матриц была факторизована в исходном виде (обозначим полученный фактор L), и после применения перестановки, найденной методом минимальной степени (обозначим полученный фактор L'). На рис. 3.4 приведена диаграмма, позволяющая сравнить коэффициент заполнения исходной матрицы A, и ее факторов L и L'.

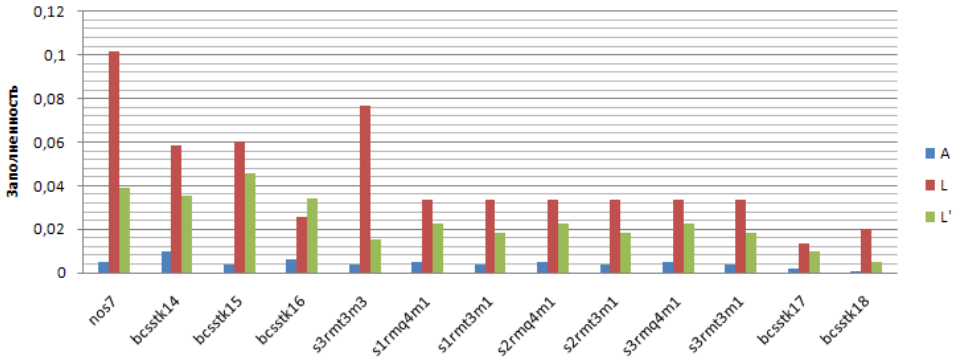


Рис. 3.4. Заполненность матриц

Сравнение коэффициентов заполненности факторов L и L' очевидным образом демонстрирует преимущества используемого метода минимальной степени.

3.3.3. Метод вложенных сечений

Как и в предыдущем пункте, для описания метода вложенных сечений нам потребуются некоторые дополнительные сведения из теории графов.

Расстоянием $d(u,v)$ между вершинами графа u, v называется длина кратчайшего пути, соединяющего эти вершины.

Наибольшее расстояние между любыми двумя вершинами графа называется *диаметром* графа.

Наибольшее расстояние от вершины u до любой другой вершины называется *эксцентриситетом* вершины и обозначается $e(u)$. Таким образом,

$$Diam(G)=\max\{e(u_i),\ u_i \in V\}.$$

Если эксцентриситет вершины совпадает с диаметром графа, то такая вершина называется *периферийной*.

Псевдопериферийная вершина u определяется следующим условием: если v – любая вершина, для которой выполнено условие $d(u,v)=e(u)$, то будет выполнено и условие $e(v)=e(u)$. Указанное определение гарантирует, что эксцентриситет псевдопериферийной вершины будет «близким» к диаметру графа.

Граф называется *связным*, если для каждой пары его вершин найдется путь, который их связывает. В противном случае граф называют *несвязным*. Всякий несвязный граф состоит из двух или более *компонент связности*.

Разделителем называется множество вершин, удаление которых вместе с инцидентными им ребрами приводит к появлению несвязного графа (или к увеличению числа связных компонент, если граф уже был несвязным). Разделитель называют *минимальным*, если никакое его собственное подмножество не является разделителем. Разделитель, состоящий из одной вершины, называется *разрезающей вершиной*.

После того, как введены в рассмотрения все требуемые понятия, сформулируем схему алгоритма вложенных сечений.

Пусть исходной матрице A из (3.2) соответствует граф $G(V,E)$. В графе G находим разделитель $S=\{v_k\}$, где $v_k \in V$. Будем считать, что вершины в графе перенумерованы так, что удаление вершины v_k (и инцидентных с ним ребер) приводит к появлению двух связных компонент G_1 и G_2 с вершинами $\{v_i\}$, $1 \le i \le k-1$, и $\{v_j\}$, $k+1 \le j \le n$, соответственно. Так как $S=\{v_k\}$ является разделителем, то любая i-я вершина G_1 не будет достижима ни для какой j-й вершины из G_2. В этом случае все элементы фактора Холецкого L_{ij} с соответствующими номерами будут равны 0, и заполнения не произойдет.

Далее введем новую нумерацию вершин (которая соответствует перестановке переменных в системе уравнений): перенумеруем вершины в компонентах последовательно, от 1 до $n-1$ (для этого достаточно уменьшить номера вершин в компоненте G_2, нумерация в G_1 останется прежней), а k-й вершине присвоим последний номер n, т.е. v_k перенумеруем как v_n.

Затем в каждой выделенной компоненте G_1 и G_2 проводим аналогичную процедуру: находим разделители $S_1=\{v_p\}$ в G_1 и $S_2=\{v_q\}$ в G_2, выделяем четыре новые компоненты связности: $G_{1,1}$ и $G_{1,2}$ с вершинами $\{v_i\}$, $1 \le i \le p-1$, и $\{v_j\}$, $p+1 \le j \le k-1$; $G_{2,1}$ и $G_{2,2}$ с вершинами $\{v_i\}$, $k+1 \le i \le q-1$, и $\{v_j\}$, $q+1 \le j \le n-1$. Следующий шаг алгоритма заключается в перенумерации вершин в пределах компонент указанным выше способом: вершины компонент связности нумеруются последовательно, разделителю присваивается последний номер. Если в какой-либо компоненте связности G^* нельзя найти разделитель, то в качестве соответствующего множества S^* выбирается само множество G^* и нумеруются все его вершины. Выполнение процедуры прекращается, если все компоненты связности исчерпаны.

При практической реализации алгоритма нумерацию можно организовать следующим образом: нумеровать все вершины разделителей S в обратном порядке (от n до 1), как только такое множество будет получено; вершины компонент связности G остаются непронумерованными. Окончательная нумерация вершин (т.е. матрица перестановки) определяется непосредственно после выполнения алгоритма.

Следует отметить, что разделитель S можно выбирать разными способами. Для того, чтобы метод вложенных сечений был максимально эффективным, следует придерживаться следующих правил:

- по возможности выбирать минимальный из всех разделителей (идеальный случай – разделитель из одной вершины):
- возникающие при удалении разделителя компоненты связности должны быть примерно одинакового размера.

Выполнение данных рекомендаций приводит к возникновению больших нулевых блоков в результирующем факторе L.

Полное описание алгоритмов, которые обеспечивают выполнение данных правил, может быть найдено в книгах [9, 10].

4. Решение дифференциальных уравнений в частных производных

Уравнение, связывающее неизвестную функцию $u(x_1,\ldots, x_n)$, независимые переменные $x_1,\ldots, x_n$ и частные производные от неизвестной функции, называется *дифференциальным уравнением в частных производных*. В общем случае оно имеет вид

$$F\left(x_1,\ldots,x_n,u,\frac{\partial u}{\partial x_1},\ldots,\frac{\partial u}{\partial x_n},\ldots,\frac{\partial^k u}{\partial x_1^{k_1}\ldots\partial x_n^{k_n}} \right)=0,$$

где F – заданная функция своих аргументов. Порядок старшей частной производной, входящей в уравнение, называют *порядком уравнения* в частных производных.

Например, наиболее общее уравнение в частных производных второго порядка имеет вид

$$F\left(x,y,u,\frac{\partial u}{\partial x},\frac{\partial u}{\partial y},\frac{\partial^2 u}{\partial x^2},\frac{\partial^2 u}{\partial x\partial y},\frac{\partial^2 u}{\partial y^2} \right)=0.$$

Уравнение в частных производных называется *квазилинейным*, если оно линейно относительно всех старших производных от неизвестной функции. Так, например, уравнение

$$A\left(x,y,u,u_x,u_y\right)\frac{\partial^2 u}{\partial x^2}+B(\ldots)\frac{\partial^2 u}{\partial x\partial y}+C(\ldots)\frac{\partial^2 u}{\partial y^2}+f(\ldots)=0$$

есть квазилинейное уравнение второго порядка.

Уравнение в частных производных называется *линейным*, если оно линейно относительно неизвестной функции и ее частных производных. Так, например, уравнение

$$A(x,y)\frac{\partial^2 u}{\partial x^2}+B(x,y)\frac{\partial^2 u}{\partial x\partial y}+C(x,y)\frac{\partial^2 u}{\partial y^2}+$$
$$+D(x,y)\frac{\partial u}{\partial x}+E(x,y)\frac{\partial u}{\partial y}+G(x,y)u=f(x,y)$$

есть линейное уравнение второго порядка относительно неизвестной функции $u(x,y)$.

Решением уравнения в частных производных называется всякая функция $u=u(x_1,\ldots,x_n)$, которая, будучи подставлена в уравнение вместо неизвестной функции и ее частных производных, обращает это уравнение в тождество по независимым переменным.

Объектом нашего рассмотрения будут являться дифференциальные уравнения второго порядка, и параллельные численные методы их решения. Важность данного класса задач обусловлена тем фактом, что математическими моделями многих процессов механики (колебания струн, стержней, мембран и трехмерных объемов), физики (электромагнитные колебания, распространение тепла или диффузия частиц в среде), гидро- и газодинамики (различные виды течений) являются уравнения именно второго порядка. В качестве примера рассмотрим общий вид уравнений колебаний и диффузии.

Уравнение колебаний в общем виде можно записать как

$$\rho \frac{\partial^2 u}{\partial t^2} = \sum_{i=1}^{n} \frac{\partial}{\partial x_i}\left(p \frac{\partial u}{\partial x_i} \right) - qu + F(x,t),$$

где неизвестная функция $u(x, t)$ зависит от n (n=1, 2, 3) пространственных переменных $x=(x_1,\ldots,x_n)$ и времени t; коэффициенты ρ, p и q определяются свойствами среды; а $F(x,t)$ – плотность внешнего возмущения.

В частном случае, уравнение малых поперечных колебаний струны (или же малых продольных колебаний стержня) постоянной плотности примет вид

$$\frac{\partial^2 u}{\partial t^2} = \alpha^2 \frac{\partial^2 u}{\partial x^2} + f(x,t), \tag{4.1}$$

где α^2 – положительная константа, зависящая от свойств струны (или стержня). Уравнение (4.1) обычно называют *волновым уравнением*.

Аналогично записывается уравнение поперечных колебаний мембраны

$$\frac{\partial^2 u}{\partial t^2} = \alpha^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + f(x,y,t).$$

Трехмерное волновое уравнение

$$\frac{\partial^2 u}{\partial t^2} = \alpha^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) + f(x,y,z,t)$$

описывает процессы распространения звука в однородной среде и электромагнитных волн в однородной непроводящей среде. Этому уравнению удовлетворяет плотность газа, его давление и потенциал скоростей, а также

составляющие напряженности электрического и магнитного полей и соответствующие потенциалы.

В дальнейшем мы будем записывать волновые уравнения единой формулой

$$\frac{\partial^2 u}{\partial t^2} = \alpha^2 \Delta u + f \ ,$$

где Δ – оператор Лапласа

$$\Delta u = \frac{\partial^2 u}{\partial x_1^2} + \ldots + \frac{\partial^2 u}{\partial x_n^2} \, .$$

Процессы распространения тепла или диффузии частиц в среде описываются следующим общим *уравнением теплопроводности*

$$\rho \frac{\partial u}{\partial t} = \sum_{i=1}^{n} \frac{\partial}{\partial x_i} \left(p \frac{\partial u}{\partial x_i} \right) - qu + F(x,t) \, .$$

Если среда – однородная, то уравнение теплопроводности приобретает вид

$$\frac{\partial u}{\partial t} = \alpha^2 \Delta u + f(x,t) \, ,$$

где положительная константа α^2 зависит от свойств среды.

Для стационарных процессов (т.е. в случае $u(x,t)=u(x)$ и $F(x,t)=F(x)$) уравнения колебаний и диффузии приобретают вид

$$\sum_{i=1}^{n} \frac{\partial}{\partial x_i} \left(p \frac{\partial u}{\partial x_i} \right) - qu = -F(x) \, . \qquad (4.2)$$

При p=const, q=0 уравнение называется *уравнением Пуассона*

$$\Delta u = -f(x), \qquad (4.3)$$

при $f(x)$=0 уравнение (4.3) называется *уравнением Лапласа*

$$\Delta u = 0.$$

Известно, например, что безвихревое стационарное течение несжимаемой жидкости описывается уравнением Лапласа, где u – скалярная функция, называемая потенциалом скорости, а скорость $\vec{v}$ течения жидкости связана с u как

$$\vec{v} = \mathrm{grad}(u).$$

Для однозначности решения дифференциальных уравнений необходимо наложение дополнительных условий.

Начальные условия – условия, определяющие значения искомой функции (и, возможно, некоторых ее производных) при одном значении независимой переменной (например, в начальный момент времени, т.е. при t=0). Если в уравнении присутствует только первая производная по времени (как в уравнении теплопроводности), то достаточно задать одно начальное условие на значение функции. Физической интерпретацией будет являться распределение температуры в теле в начальный момент. Если же в уравнении присутствует вторая производная по времени (волновое уравнение), то необходимы два начальных условия – на значение функции и ее первой производной. Физической интерпретацией будут являться положение струны в начальный момент и скорость движения ее точек.

Граничные условия – условия, определяющие значения искомой функции (и, возможно, некоторых ее производных) на границе пространственной области, внутри которой ищется решение (при различных значениях независимых переменных). Выделяют три типа граничных условий:

- известны значения функции на границе (*задача Дирихле*);
- известны значения производной функции по нормали к границе (*задача Неймана*);
- известна линейная комбинация значений функции и ее производной по нормали к границе (*задача Робена*).

Следует отметить, что только некоторый узкий класс поставленных задач можно решить аналитически (например, в случае постоянных коэффициентов). Большинство же задач, которые описывают явления и процессы окружающего нас мира, допускают лишь численное решение.

Универсальным численным методом решения рассмотренных дифференциальных уравнений является *метод конечных разностей*, при котором решение дифференциального уравнения сводится к решению разностных уравнений. Для получения разностного уравнения вместо дифференциального следует:

- заменить область непрерывного изменения аргументов дискретным множеством точек (*сеткой*);
- заменить (*аппроксимировать на сетке*) дифференциальное уравнение разностным уравнением.

Сетка, как правило, выбирается равномерная. Например, равномерная сетка в двумерной прямоугольной области $D=\{(x,y)\colon 0\le x,y\le 1\}$ может быть задана в виде

$$D_h = \left\{(x_i, y_i) : x_i = ih, y_j = jh, 0 \le i, j \le n, h = 1/n\right\}.$$

где величина N задает количество узлов по каждой из координат области D.

Используя значения функции в узлах сетки, можно аппроксимировать значения первой и второй производных, используя следующие формулы:

$$u'(x_i) = \frac{u(x_i) - u(x_{i-1})}{h} + O(h)$$

$$u'(x_i) = \frac{u(x_{i+1}) - u(x_i)}{h} + O(h)$$

$$u'(x_i) = \frac{u(x_{i+1}) - u(x_{i-1})}{2h} + O(h^2)$$

$$u''(x_i) = \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} + O(h^2)$$

В связи с построением разностной схемы возникают следующие проблемы, которые типичны для разностных методов вообще. Во-первых, необходимо убедиться, что система линейных алгебраических уравнений имеет единственное решение, и указать алгоритм, позволяющий получить это решение. И, во-вторых, надо показать, что при стремлении шага сетки h к нулю решение разностной задачи будет сходиться к решению исходной дифференциальной задачи. Вопросы разрешимости и сходимости разностных схем выходят за рамки данного пособия; им посвящена обширная литература, среди которой можно отметить книги [2, 4].

4.1. Решение волнового уравнения

Рассмотрим первую краевую задачу для одномерного волнового уравнения

$$\frac{\partial^2 u}{\partial t^2} = \alpha^2 \frac{\partial^2 u}{\partial x^2} + f(x,t),\ 0<x<1,\ 0<t\le T, \qquad (4.4)$$

$$u(x,0)=u_0(x),\ \frac{\partial u}{\partial t}(x,0)=\overline{u}_0(x),\ 0\le x\le 1,$$

$$u(0,t)=\mu_1(t),\ u(1,t)=\mu_2(t),\ 0\le t\le T.$$

Известно [3], что эта задача поставлено корректно, т.е. ее решение существует, оно единственно и непрерывно зависит от начальных и граничных данных.

С целью построения разностной схемы введем сетку

$$\omega_{h\tau}=\{(x_i,\ t_j):\ x_i=ih,\ 0\le i\le N,\ h=1/N,\ t_j=j\tau,\ 0\le j\le L,\ \tau=T/L\}.$$

с шагами h по x и τ по t.

Очевидно, минимальный шаблон, на котором можно аппроксимировать уравнение (4.4), это пятиточечный шаблон, изображенный на рис. 4.1.

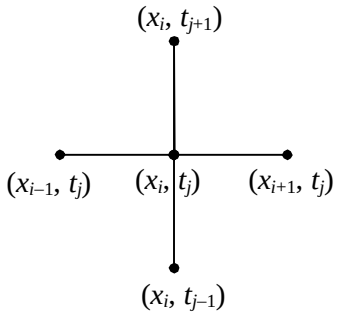


Рис. 4.1. Шаблон трехслойной разностной схемы

В отличие от схем для уравнения теплопроводности, в которых используются только два временных слоя (слои n и $n+1$), здесь требуется использовать три слоя: $n-1$, n и $n+1$. Такие схемы называются *трехслойными*. Используя на данном шаблоне приближенные формулы вычисления производных

$$\left(\frac{\partial^2 u}{\partial t^2}\right)_{ij}\approx\frac{u_i^{j+1}-2u_i^j+u_i^{j-1}}{\tau^2},\ \left(\frac{\partial^2 u}{\partial x^2}\right)_{ij}\approx\frac{u_{i+1}^j-2u_i^j+u_{i-1}^j}{h^2},$$

можно записать разностную аппроксимацию уравнения и граничных условий

$$\frac{v_i^{j+1}-2v_i^j+v_i^{j-1}}{\tau^2}=\alpha^2\frac{v_{i+1}^j-2v_i^j+v_{i-1}^j}{h^2}+f_i^j,\ 1\le i\le N-1,\ 1\le j\le L-1,$$
$$v_0^j=\mu_1(t_j),v_N^j=\mu_2(t_j),\ 0\le j\le L, \tag{4.5}$$

где v_i^j – сеточная функция, аппроксимирующая точное решение дифференциального уравнения в узлах сетки. Затем, используя первое начальное условие из (4.4) сразу получаем

$$v_i^0 = u_0(x_i),\ 1 \le i \le N-1.$$

Простейшая замена второго начального условия в (4.4) уравнением

$$\frac{v_i^1 - v_i^0}{\tau} = \overline{u}_0(x_i),\ 1 \le i \le N-1,$$

имеет лишь первый порядок аппроксимации по τ, тогда как разностное уравнение (4.5) второй порядок погрешности аппроксимации по τ по h. Однако известно [4], что разностное уравнение

$$\frac{v_i^1 - v_i^0}{\tau} = \overline{u}_0(x_i) + \frac{\tau}{2}\left(\alpha^2 \frac{v_{i+1}^0 - 2v_i^0 + v_{i-1}^0}{h^2} + f_i^0\right),\ 1 \le i \le N-1,$$

аппроксимирует второе начальное условие из (4.4) со вторым порядком по τ и по h.

4.1.1. Явная разностная схема

Итак, совокупность уравнений

$$\frac{v_i^{j+1} - 2v_i^j + v_i^{j-1}}{\tau^2} = \alpha^2 \frac{v_{i+1}^j - 2v_i^j + v_{i-1}^j}{h^2} + f_i^j,\ 1 \le i \le N-1,\ 1 \le j \le L-1,$$

$$v_0^j = \mu_1(t_j), v_N^j = \mu_2(t_j),\ 0 \le j \le L,$$

$$v_i^0 = u_0(x_i), v_i^1 = v_i^0 + \tau \overline{u}_0(x_i) + \frac{\tau^2}{2}\left(\frac{v_{i+1}^0 - 2v_i^0 + v_{i-1}^0}{h^2} + f_i^0\right),\ 1 \le i \le N-1,$$

составляет разностную схему, аппроксимирующую исходную задачу (4.4). Известно [6], что для вычислительной устойчивости данной разностной схемы необходимо и достаточно выполнения условия Куранта

$$\alpha\tau < h. \qquad (4.6)$$

Решение находится по слоям: значения сеточной функции на новом слое выражается явным образом через значения на предыдущих слоях. Действительно, разностное уравнение (4.5) содержит только одну неизвестную v_i^{j+1}, значения которой можно выразить как

$$v_i^{j+1} = 2v_i^j - v_i^{j-1} + \left(\frac{\alpha\tau}{h}\right)^2 \left(v_{i+1}^j - 2v_i^j + v_{i-1}^j\right) + \tau^2 f_i^j \text{, } 1 \le i \le N-1,\ 1 \le j \le L-1;$$

при этом значения на нулевом и первом слоях находятся непосредственно из начальных условий.

4.1.2. Организация параллельных вычислений

Анализ последовательного алгоритма показывает, что вычисления проводятся «по слоям», и вычислить значения сеточной функции на (j+1)-м слое можно лишь вычислив значения на j-м слое. Таким образом, в качестве подзадачи, допускающей распараллеливание, можно рассмотреть вычисление значений на очередном слое. Каждый поток получает свою полосу данных, и проводит вычисление значений (j+1)-м слое лишь в пределах своей полосы.

Ниже приведен рисунок, иллюстрирующий процесс разделения данных для явной схемы. Светлые точки соответствуют узлам сетки, значения в которых могут быть получены из начальных и граничных условий, темные точки соответствуют узлам, значения в которых предстоит вычислить, пунктиром изображен шаблон разностной схемы. Стрелками указано направление обхода узлов сетки в пределах каждой полосы.

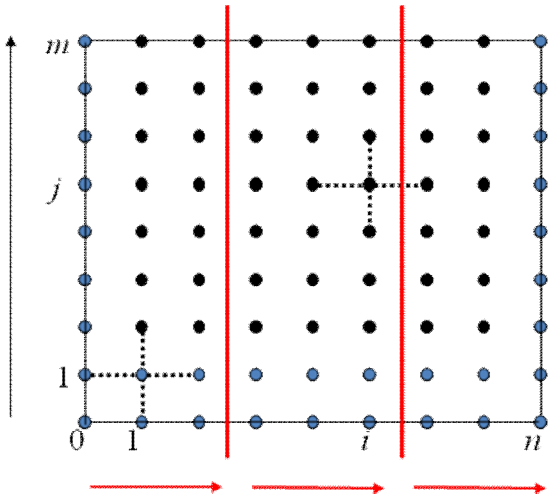


Рис. 4.2. Схема разделения данных

Следует отметить, что в отличие от систем с распределенной памятью, для систем с общей памятью снимается проблема пересылки узлов на границах полос разделения данных.

4.1.3. Результаты вычислительных экспериментов

Для проведения вычислительных экспериментов была сформирована тестовая задача

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}, x \in [0,1], t \in [0,2];$$

$$u(0,t) = u(1,t) = 0;$$

$$u(x,0) = 0.1\sin(\pi x), \frac{\partial u}{\partial x}(x,0) = 0.$$

Нетрудно видеть, что данная задача имеет решение

$$u(x,t) = 0.1\sin(\pi x)\cos(\pi t).$$

Для выполнения условия (4.6) вычислительной устойчивости было выбрано фиксированное число разбиений m=25000 по переменной t, число разбиений n по переменной x варьировалось от 2500 до 10000.

Вычислительные эксперименты для оценки эффективности параллельного алгоритма решения волнового уравнения проводились на аппаратуре, технические характеристики которой указанны во введении. Результаты вычислительных экспериментов приведены в таблице (время работы алгоритмов указано в секундах).

Табл. 4.1. Экспериментальная оценка ускорения

n	1 поток	Параллельный алгоритм							
		2 потока		4 потока		6 потоков		8 потоков	
		T	S	T	S	T	S	T	S
2500	1,49	0,72	2,05	0,36	4,14	0,41	3,61	0,48	3,07
5000	3,22	1,57	2,04	1,01	3,18	0,92	3,51	1,03	3,12
7500	5,08	2,59	1,96	1,56	3,26	1,50	3,39	1,73	2,93
10000	6,72	3,76	1,79	2,23	3,01	2,25	2,99	2,50	2,69

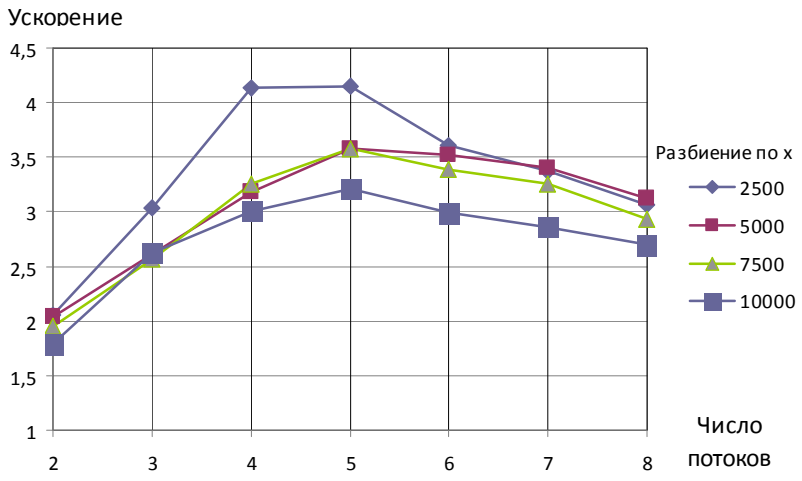


Рис. 4.3. Зависимость ускорения от числа потоков

Эксперименты показывают падение ускорения при числе потоков, большем пяти. Данный факт объясняется низкой трудоемкостью операций, выполняемых каждым потоком. В этом случае накладные расходы (время δ, требуемое для создания и закрытия параллельной секции) превышают выигрыш от распараллеливания. Чтобы подтвердить наше предположение, проведем еще один эксперимент с большей вычислительной нагрузкой на поток.

С этой целью сформируем вторую тестовую задачу

$$\frac{\partial^2 u}{\partial t^2} = \frac{\exp(\cos(x))}{10}\frac{\partial^2 u}{\partial x^2}, x \in [0,1], t \in [0,2];$$

$$u(0,t) = u(1,t) = 0;$$

$$u(x,0) = 0.1\sin(\pi x), \frac{\partial u}{\partial x}(x,0) = 0.$$

При аналогичном выборе числа разбиений L=25000 по переменной t, и варьировании числа разбиений N по переменной x от 2500 до 10000 условие (4.6) вычислительной устойчивости будет выполняться. Результаты эксперимента приведены ниже, время работы алгоритмов указано в секундах.

Табл. 4.2. Ускорение с вычислительной нагрузкой

n	1 поток	Параллельный алгоритм			
		2 потока	4 потока	6 потоков	8 потоков

		T	*S*	*T*	*S*	*T*	*S*	*T*	*S*
2500	5,43	2,56	2,12	1,20	4,52	0,84	6,45	0,81	6,69
5000	11,20	5,48	2,05	2,73	4,10	1,81	6,19	1,67	6,71
7500	16,83	8,41	2,00	4,26	3,95	2,89	5,83	2,65	6,35
10000	22,64	11,34	2,00	5,77	3,92	3,96	5,71	3,70	6,12

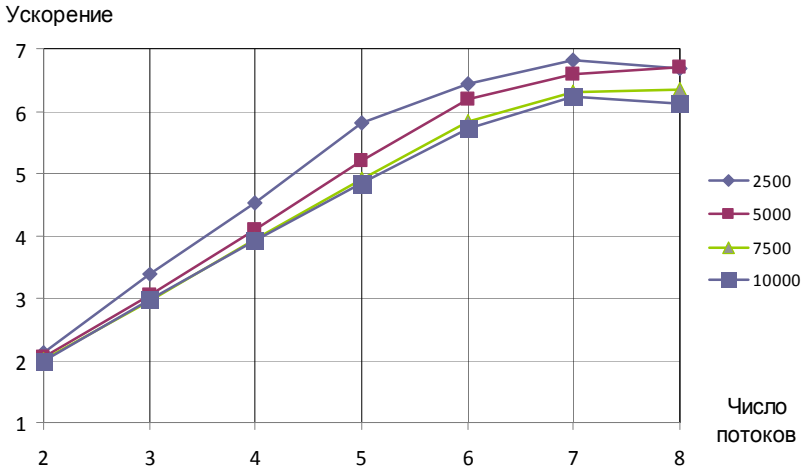


Рис. 4.4. Ускорение с вычислительной нагрузкой

Приведенные результаты подтверждают наше предположение: в случае большей вычислительной нагрузки на поток ускорение становится практически линейным до 7 потоков.

4.2. Решение задачи теплопроводности

Процесс распространения тепла в одномерном однородном стержне $0<x<l$ описывается уравнением

$$\frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial x^2} + f(x,t),$$

Без ограничения общности можно считать l=1, α^2=1. Мы будем рассматривать первую краевую задачу в области

$$D=\{0\le x\le 1,\ 0\le t\le T\}.$$

Требуется найти непрерывное в D решение $u=u(x,t)$ задачи

$$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}+f(x,t),\ 0<x<1,\ 0<t\le T,$$

$$u(x,0)=u_0(x),\ 0\le x\le 1,\ u(0,t)=u_1(t),\ u(1,t)=u_2(t),\ 0\le t\le T.$$

С целью построения разностной схемы в области D введем сетку

$$\omega_{h\tau}=\{(x_i,t_j):\ x_i=ih,\ 0\le i\le n,\ h=1/n,\ t_j=j\tau,\ 0\le j\le m,\ \tau=T/m\}.$$

с шагами h по x и τ по t. Используя приближенные формулы вычисления производных

$$\left(\frac{\partial u}{\partial t}\right)_{ij}\approx\frac{u_i^{j+1}-u_i^j}{\tau},\ \left(\frac{\partial^2 u}{\partial x^2}\right)_{ij}\approx\frac{u_{i+1}^j-2u_i^j+u_{i-1}^j}{h^2},$$

можно записать разностную схему вида

$$\frac{v_i^{j+1}-v_i^j}{\tau}=\sigma\frac{v_{i+1}^{j+1}-2v_i^{j+1}+v_{i-1}^{j+1}}{h^2}+(1-\sigma)\frac{v_{i+1}^j-2v_i^j+v_{i-1}^j}{h^2}+\varphi_i^j,$$
$$v_i^0=u_0(x_i),v_0^j=u_1(t_j),v_n^j=u_2(t_j),0\le i\le n,0\le j\le m. \qquad (4.7)$$

где v_i^j – сеточная функция, являющаяся точным решением разностной схемы и аппроксимирующая точное решение дифференциального уравнения в узлах сетки; σ – параметр, называемый весом, а φ_i^j – некоторая правая часть, например, $\varphi_i^j=f_i^j$. В общем случае схема (4.7) определена на шеститочечном шаблоне, изображенном на рис. 4.5.

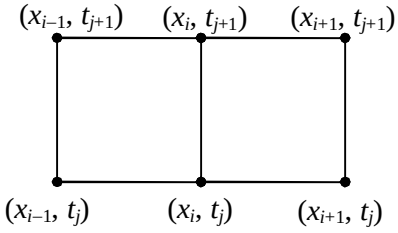


Рис. 4.5. Шаблон схемы с весом

Далее рассмотрим важные частные случаи схемы (4.7).

4.2.1. Явная разностная схема

В случае $\sigma=0$ схема (4.7) приобретет вид

$$\frac{v_i^{j+1}-v_i^j}{\tau}=\frac{v_{i+1}^j-2v_i^j+v_{i-1}^j}{h^2}+\varphi_i^j, \tag{4.8}$$

где $\varphi_i^j = f_i^j$; граничные условия при этом останутся неизменными. Шаблон схемы 4.6 представлен на рис. 4.6.

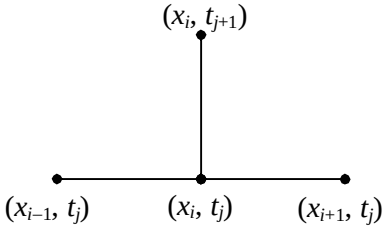


Рис. 4.6. Шаблон явной схемы

Данная схема является явной. Действительно, значения сеточной функции на (j+1)-м слое можно найти по явной формуле

$$v_i^{j+1}=\left(1-\frac{2\tau}{h^2}\right)v_i^j+\frac{\tau}{h^2}\left(v_{i+1}^j+v_{i-1}^j\right)+\tau\varphi_i^j, 0\le j\le L-1, \tag{4.9}$$

используя известные значения на j-м слое.

Известно [4], что явная разностная схема будет вычислительно устойчива лишь при выполнении условия

$$\tau \le h^2/2.$$

т.е. схема является условно устойчивой. При этом схема аппроксимирует исходное уравнение с порядком $O(\tau+h^2)$.

Условие вычислительной устойчивости налагает существенные требования на шаг по времени, поэтому данная вычислительная схема требует существенных вычислительных затрат. Применение явных разностных схем оправдано лишь в случае более слабых ограничений на шаг τ (см. п. 4.1.1).

4.2.2. Неявные разностные схемы

В случае σ=1 схема (4.7) приобретет вид

$$\frac{v_i^{j+1}-v_i^j}{\tau}=\frac{v_{i+1}^{j+1}-2v_i^{j+1}+v_{i-1}^{j+1}}{h^2}+f_i^{j+1}, \tag{4.10}$$
$$v_i^0=u_0(x_i), v_0^j=u_1(t_j), v_n^j=u_2(t_j), 0\le i\le n, 0\le j\le m.$$

Данная схема определена на четырехточетном шаблоне (см. рис. 4.7) и является *неявной*.

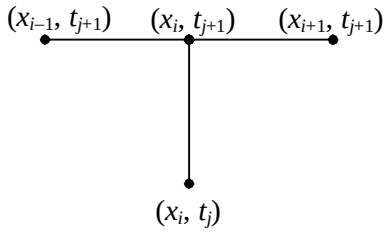


Рис. 4.7. Шаблон неявной схемы

В данном случае для определения v_i^{j+1} получаем систему уравнений вида

$$\frac{\tau}{h^2} v_{i-1}^{j+1} - \left(1 + \frac{2\tau}{h^2}\right) v_i^{j+1} + \frac{\tau}{h^2} v_{i+1}^{j+1} = v_i^j + \tau f_i^{j+1}, 0 < i < n,$$
$$v_0^{j+1} = u_1(t_{j+1}), v_n^{j+1} = u_2(t_{j+1}). \tag{4.11}$$

Система уравнений является трехдиагональной, и может быть решена методом прогонки.

Известно [4], что неявная разностная схема будет вычислительно устойчива при любом соотношении шагов τ и h (абсолютная устойчивость). Погрешность аппроксимации составит также $O(\tau+h^2)$.

В случае $0<\sigma<1$ говорят о схеме с весом σ (шаблон представлен на рис. 4.5). Очевидно, что схема с весом тоже будет неявной схемой. Особо отметим случай $\sigma=½$ (данная схема также называется *схемой Кранка-Николсона*), который приводит к виду

$$\frac{v_i^{j+1} - v_i^j}{\tau} = \frac{1}{2}\left[\frac{v_{i+1}^{j+1} - 2v_i^{j+1} + v_{i-1}^{j+1}}{h^2} + \frac{v_{i+1}^j - 2v_i^j + v_{i-1}^j}{h^2}\right] + f_i^{j+1/2},$$
$$v_i^0 = u_0(x_i), v_0^j = u_1(t_j), v_n^j = u_2(t_j), 0 \le i \le n, 0 \le j \le m. \tag{4.12}$$

Значения на новом слое также находятся методом прогонки для системы уравнений

$$\frac{\tau}{2h^2} v_{i-1}^{j+1} - \left(1 + \frac{\tau}{h^2}\right) v_i^{j+1} + \frac{\tau}{2h^2} v_{i+1}^{j+1} =$$
$$= \left(1 - \frac{\tau}{h^2}\right) v_i^j + \frac{\tau}{2h^2}\left(v_{i-1}^j + v_{i+1}^j\right) + \tau f_i^{j+1/2}, 0 < i < N \tag{4.13}$$
,

$$v_0^{j+1} = u_1(t_{j+1}), v_n^{j+1} = u_2(t_{j+1}).$$

Известно [4], что симметричная разностная схема с весом ½ будет абсолютно устойчива, а ее погрешность аппроксимации составит $O(\tau^2+h^2)$. Данные свойства делают схему с весом ½ предпочтительной для проведения расчетов, т.к. она обеспечивает хорошую точность при не слишком малых шагах сетки.

4.2.3. Организация параллельных вычислений

Анализ последовательного алгоритма показывает, что вычисления проводятся «по слоям», и вычислить значения сеточной функции на (j+1)-м слое можно лишь вычислив значения на j-м слое. Таким образом, в качестве подзадачи, допускающей распараллеливание, можно рассмотреть вычисление значений на очередном слое.

В случае использования чисто неявной разносной схемы или же схемы Кранка-Николсона вычисление значений сеточной функции на (j+1)-м слое соответствует решению системы линейных алгебраических уравнений с трехдиагональной матрицей, и можно применить метод прогонки в его параллельном варианте (см. п. 1.1).

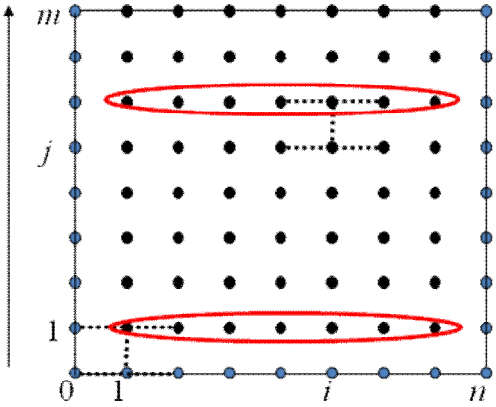


Рис. 4.8. Распараллеливание схемы Кранка-Николсона

На рисунке 4.8 проиллюстрирован процесс решения задачи. Светлые точки соответствуют узлам сетки, значения в которых могут быть получены из начальных и граничных условий, темные точки соответствуют узлам, значения в которых предстоит вычислить, пунктиром изображен шаблон разностной схемы. Обведены узлы, решение которых может быть найдено параллельным методом прогонки.

4.2.4. Результаты вычислительных экспериментов

В качестве иллюстрации рассмотрим решение дифференциального уравнения вида

$$\frac{\partial C}{\partial t}+\frac{1}{2}\sigma^2\frac{\partial^2 C}{\partial z^2}+\left(r-D-\frac{1}{2}\sigma^2\right)\frac{\partial C}{\partial z}-rC=0.$$

Уравнение такого вида возникает в задачах финансовой математики при определении цены конвертируемой облигации (здесь $C(z,t)$ – цена конвертируемой облигации в момент времени t). Подробная содержательная постановка задачи и разностная схема для ее численного решения приведены в лабораторной работе «Решение дифференциальных уравнений в частных производных».

Здесь же отметим, что задача решалась с использованием схемы Кранка-Николсона, число разбий по t было равно 100, число разбиений по z варьировалось в диапазоне от 256 до 32768. Возникающая при этом трехдиагональная система уравений решалась методами прогонки и параллельной встречной прогонки. Ниже в таблице приведено время работы (в секундах) указанных методов, а также ускорение параллельной встречной прогонки относительно последовательной.

Табл. 4.3. Экспериментальная оценка ускорения

Число разбиений	Прогонка	Встречная прогонка	Ускорение
256	0,08	0,08	1,00
512	0,14	0,16	0,90
1024	0,28	0,16	1,79
2048	0,51	0,36	1,44
4096	1,03	0,56	1,83
8192	2,03	1,72	1,18
16384	4,07	2,20	1,85
32768	8,16	5,63	1,45

Результаты показывают, что использование метода встречной прогонки дает ускорение до 1.85 раза при решении задачи. Следует отметить, что при решении задачи не учитывалась специфика трехдиагональной системы уравнений: в ней на диагоналях стоят одинаковые числа. Учет данной специфики может дать еще большее ускорение.

4.3. Решение задачи Дирихле для уравнения Пуассона

В данном разделе мы рассмотрим решение задачи Дирихле для уравенения Пуассона

$$\begin{aligned} &\Delta u(x,y)=-f(x,y)\ ,\ x\in(a,b),\ y\in(c,d);\\ &u(a,y)=\psi_1(y),\ u(b,y)=\psi_2(y),\\ &u(x,c)=\psi_3(x),\ u(x,d)=\psi_4(x). \end{aligned} \qquad (4.14)$$

Величины a,b,c,d и функции $f(x,y)$, ψ_i, $i=1,\ldots4$, - известны, функция $u(x,y)$ - искомая. Не ограничивая общности, будем считать в дальнейшем $a=c=0$, $b=d=1$. Задача (4.14) может рассматриваться как задача на отыскание деформации упругой мембраны, находящейся под внешним воздействием, края которой закреплены по заданному закону, либо как задача об отыскании стационарного распределения температур на квадратной пластине с внешними источниками тепла, на краях которой поддерживается заданный температурный режим.

4.3.1. Построение разностной схемы

Для отыскания численного решения дифференциальной задачи (4.14) в области задания уравнения вводится равномерная прямоугольная сетка и используется разностная схема второго порядка аппроксимации. Сеточная функция $v(x,y)$, являющаяся точным решением разностной схемы, трактуется как приближенное (численное) решение исходной задачи (4.14).

Для задания сетки необходимо указать два положительных целых числа n и m, определяющих число разбиений по оси x и y соответственно. Шаг сетки по оси x обозначим через h, по оси y – через k,

$$h=1/n\ ,\ k=1/m\ . \qquad (4.15)$$

Узлы сетки обозначим через (x_i, y_j), $i=0,\ldots,n$, $j=0,\ldots,m$, где

$$x_i=ih,\ i=0,\ldots,n;\ y_j=jk,\ j=0,\ldots,m.$$

Узлы с координатами (x_i,y_j), где $i=1,\ldots,\ n-1$, $j=1,\ldots,\ m-1$, называются внутренними, остальные узлы – граничными. Узлы с координатами (x_0,y_0), (x_n,y_0), (x_0,y_m), (x_n,y_m) называются угловыми.

Численное (приближенное) решение задачи (4.14) обозначим через $v(x,y)$. Значение сеточной функции $v(x,y)$ в узле (x_i,y_j) – внутреннем или граничном – обозначим через v_{ij}:

$$v_{ij}=v(x_i, y_j),\ i=0,\ldots,n,\ j=0,\ldots,m.$$

Значения функции $f(x,y)$ во внутреннем узле с координатами (x_i, y_j) обозначим через f_{ij}:

$$f_{ij}=f(x_i, y_j),\ i=1,\ldots,n-1,\ j=1,\ldots,m-1.$$

Значения функций граничных условий в соответствующих граничных узлах обозначим через $\psi_{1j},\psi_{2j},\psi_{3j},\psi_{4j}$:

$$\psi_1(y_j)=\psi_{1j},\psi_2(y_j)=\psi_{2j},\ j=0,\ldots,m;$$
$$\psi_3(x_i)=\psi_{3i},\psi_4(x_i)=\psi_{4i},i=0,\ldots,n.$$

Разностная схема, т.е. система уравнений для отыскания сеточной функции $v(x,y)$, принимает вид

$$\begin{cases}\dfrac{v_{i-1,j}-2v_{ij}+v_{i+1,j}}{h^2}+\dfrac{v_{i,j-1}-2v_{ij}+v_{i,j+1}}{k^2}=-f_{ij}\\ i=1,\ldots,n-1;\quad j=1,\ldots,m-1;\\ v_{0j}=\psi_{1j}\quad v_{nj}=\psi_{2j}\quad j=1,\ldots,m-1;\\ v_{i0}=\psi_{3i}\quad v_{im}=\psi_{4i}\quad i=1,\ldots,n-1.\end{cases}\tag{4.16}$$

Заметим, что значения граничных функций ψ_i, $i=1,\ldots4$, в угловых узлах разностной схемой (4.16) не используются.

Уравнения разностной схемы (4.16) построены на основе аппроксимации дифференциального уравнения (4.14) во внутренних узлах сетки с помощью формул приближенного вычисления вторых частных производных

$$\frac{\partial^2 u}{\partial x^2}\approx u_{\bar{x}x_{ij}}=\frac{u_{i-1,j}-2u_{ij}+u_{i+1,j}}{h^2},$$
$$\frac{\partial^2 u}{\partial y^2}\approx u_{\bar{y}y_{ij}}=\frac{u_{i,j-1}-2u_{ij}+u_{i,j+1}}{h^2}.$$

Эти формулы аппроксимируют вторые частные производные по x и по y с погрешностью второго порядка по h и k соответственно

$$u_{\bar{x}x_{ij}}-\frac{\partial^2 u}{\partial x^2}=\frac{h^2}{12}\cdot\frac{\partial^4 u}{\partial x^4}(x_i,y_j)+o(h^2)$$
$$u_{\bar{y}y_{ij}}-\frac{\partial^2 u}{\partial y^2}=\frac{k^2}{12}\cdot\frac{\partial^4 u}{\partial y^4}(x_i,y_j)+o(h^2)$$

Остальные уравнения разностной схемы (4.16) определяются граничными условиями задачи (4.14).

Разностная схема (4.16) представляет собой линейную систему уравнений относительно неизвестного вектора v размерности $(n+1)\cdot(m+1)$, компоненты которого имеют вид

$$v = (v_{00}, v_{10}, \ \dots \ , v_{n0}, v_{01}, v_{11}, \ \dots \ , v_{n1}, \ \dots \ v_{0m}, v_{1m}, \ \dots \ , v_{nm}) .$$

В связи с тем, что значения искомой сеточной функции $v(x,y)$ в граничных узлах известны, разностную схему (4.16) допустимо трактовать как систему линейных уравнений размерности $(n-1)\cdot(m-1)$ относительно неизвестного вектора v с компонентами

$$v = (v_{11}, \ \dots \ , v_{n-1,1}, v_{11}, \ \dots \ , v_{n-1,2}, \ \dots \ v_{1,m-1}, v_{2,m-1}, \ \dots \ , v_{n-1,m-1}) .$$

Запись разностной схемы в матричном виде для сетки с параметрами n=5, m=5 приведена на рис. 4.9. Матрица имеет блочно-диагональную структуру и является разреженной.

$$\begin{pmatrix}
A & \frac{1}{h^2} & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{h^2} & A & \frac{1}{h^2} & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{h^2} & A & \frac{1}{h^2} & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{h^2} & A & 0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{k^2} & 0 & 0 & 0 & A & \frac{1}{h^2} & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{k^2} & 0 & 0 & \frac{1}{h^2} & A & \frac{1}{h^2} & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{k^2} & 0 & 0 & \frac{1}{h^2} & A & \frac{1}{h^2} & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & \frac{1}{h^2} & A & 0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 & A & \frac{1}{h^2} & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & \frac{1}{h^2} & A & \frac{1}{h^2} & 0 & 0 & \frac{1}{k^2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & \frac{1}{h^2} & A & \frac{1}{h^2} & 0 & 0 & \frac{1}{k^2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & \frac{1}{h^2} & A & 0 & 0 & 0 & \frac{1}{k^2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & 0 & A & \frac{1}{h^2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & \frac{1}{h^2} & A & \frac{1}{h^2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & \frac{1}{h^2} & A & \frac{1}{h^2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{k^2} & 0 & 0 & \frac{1}{h^2} & A
\end{pmatrix}
\begin{pmatrix} v_{11} \\ v_{21} \\ v_{31} \\ v_{41} \\ v_{12} \\ v_{22} \\ v_{32} \\ v_{42} \\ v_{13} \\ v_{23} \\ v_{33} \\ v_{43} \\ v_{14} \\ v_{24} \\ v_{34} \\ v_{44} \end{pmatrix} =
\begin{pmatrix}
-f_{11} - \frac{1}{h^2}\mu_{11} - \frac{1}{k^2}\mu_{31} \\
-f_{21} - \frac{1}{k^2}\mu_{32} \\
-f_{31} - \frac{1}{k^2}\mu_{33} \\
-f_{41} - \frac{1}{h^2}\mu_{21} - \frac{1}{k^2}\mu_{34} \\
-f_{12} - \frac{1}{h^2}\mu_{12} \\
-f_{22} \\
-f_{32} \\
-f_{42} - \frac{1}{h^2}\mu_{22} \\
-f_{13} - \frac{1}{h^2}\mu_{13} \\
-f_{23} \\
-f_{33} \\
-f_{43} - \frac{1}{h^2}\mu_{23} \\
-f_{14} - \frac{1}{h^2}\mu_{14} - \frac{1}{k^2}\mu_{41} \\
-f_{24} - \frac{1}{k^2}\mu_{42} \\
-f_{34} - \frac{1}{k^2}\mu_{43} \\
-f_{44} - \frac{1}{h^2}\mu_{24} - \frac{1}{k^2}\mu_{44}
\end{pmatrix}$$

Рис. 4.9 Матрица разностной схемы

На ее главной диагонали расположены числа вида $A = -2\left(1/h^2 + 1/k^2\right)$, остальные ненулевые элементы могут принимать либо значение $1/h^2$, либо значение $1/k^2$ (здесь h и k – шаги сетки из (4.15)).

Матрица, частный случай которой приведен на рис. 4.9, является симметричной, невырожденной и *отрицательно определенной*. Умножив уравнения системы на –1, разностную схему (4.16) можно записать как линейную систему уравнений с симметричной положительно определенной матрицей. Известно, что собственные числа данной матрицы будут определяться соотношением

$$\lambda_{ij} = \left(\frac{4}{h^2} \sin^2 \frac{\pi i}{2n} + \frac{4}{k^2} \sin^2 \frac{\pi j}{2m} \right),\ i=1,\ldots,n-1,\ j=1,\ldots,m-1.$$

Отметим, что в силу специфики матрицы системы (4.16) ее можно не хранить целиком, а вычислять ее коэффициенты в зависимости от позиции. Это позволяет применять итерационные методы, запоминая лишь значения сеточной функции на очередной итерации. Ниже приведена запись метода верхней релаксации с учетом специфики системы (4.16).

4.3.2. Применение метода верхней релаксации

При изучении свойств итерационных методов следует иметь в виду, что минимальное и максимальное собственные числа системы уравнений, соответсвующей разностной схеме (4.16), составят

$$\lambda_{\min} = \frac{4}{h^2} \sin^2 \frac{\pi}{2n} + \frac{4}{k^2} \sin^2 \frac{\pi}{2m}, \tag{4.17}$$

$$\lambda_{\max} = \frac{4}{h^2} \cos^2 \frac{\pi}{2n} + \frac{4}{k^2} \cos^2 \frac{\pi}{2m} \tag{4.18}$$

Следовательно, спектральное число обусловленности μ_A матрицы A, вычисляемое в соответствии с формулой

$$\mu_A = \lambda_{\max} / \lambda_{\min}, \tag{4.19}$$

с уменьшением шагов сетки h и k возрастает.

Поскольку скорость сходимости итерационных методов существенно зависит от числа обусловленности (чем больше μ_A, тем ниже скорость сходимости), при использовании любого итерационного метода с уменьшением шагов сетки скорость решения разностной схемы (4.16) замедляется. Кроме того, с возрастанием числа обусловленности возрастает чувствительность системы к вычислительным погрешностям. Именно в этом состоит одна из основных проблем численного решения дифференциального урав-

нения: разностная схема (4.16) хорошо аппроксимирует дифференциальную задачу (4.14) на сетке с малым шагом, но на сетке с малым шагом решение задачи (4.16) требует больших вычислительных ресурсов. Следует отметить, что использование многих итерационных методов линейной алгебры предполагает, что границы спектра матрицы A исследователю известны, что позволяет подбирать параметры итерационного процесса оптимальным образом.

В частности, для рассматриваемой системы линейных уравнений (4.16) оптимальный параметр ω метода верхней релаксации известен [4] и в случае одинаковых шагов сетки h и k равен

$$\omega_{opt} = \frac{2}{1+2\sin(\pi h/2)}. \tag{4.20}$$

Получим расчетные формулы метода верхней релаксации с учетом специфики разностной схемы (4.16), построенной для решения задачи Дирихле (4.14). Уравнения разностной схемы можно записать в виде

$$\begin{gathered} 2\cdot\left(\frac{1}{h^2}+\frac{1}{k^2}\right)\cdot v_{ij} - \frac{1}{h^2}v_{i-1,j} - \frac{1}{k^2}v_{i,j-1} - \frac{1}{h^2}v_{i+1,j} - \frac{1}{k^2}v_{i,j+1} = f_{ij}, \\ i=\overline{1,n-1} \quad j=\overline{1,m-1}. \end{gathered} \tag{4.21}$$

Запись (4.21) соответствует записи разностной схемы (4.16) как системы линейных уравнений относительно неизвестного вектора v размерности $(n-1)\cdot(m-1)$ с компонентами

$$v=(v_{11},\ \ldots\ ,v_{n-1,1},v_{11},\ \ldots\ ,v_{n-1,2},\ \ \ldots\ \ v_{1,m-1},v_{2,m-1},\ \ldots\ ,v_{n-1,m-1})$$

и с симметричной, положительно определенной матрицей. Слагаемые вида v_{ij}, соответствующие граничным узлам, являются известными.

Запишем указанные уравнения так, чтобы диагональный элемент матрицы системы (4.21), умноженный на соответствующую компоненту неизвестного вектора v, оставался слева, а остальные слагаемые уравнения и его правая часть – справа:

$$\begin{gathered} av_{ij} = \frac{1}{h^2}v_{i-1,j} + \frac{1}{h^2}v_{i+1,j} + \frac{1}{k^2}v_{i,j-1} + \frac{1}{k^2}v_{i,j+1} + f_{ij}, \\ a = 2\left(1/h^2+1/k^2\right),\ i=\overline{1,n-1},\quad j=\overline{1,m-1}. \end{gathered} \tag{4.22}$$

Чтобы получить формулы итерационного процесса, аналогичные (2.7), необходимо выяснить, какие компоненты вектора v встречаются раньше,

чем компонента v_{ij}, и какие – позже, чем v_{ij}. «Ранние» компоненты необходимо брать из нового приближения, умножая их на ω. «Поздние» компоненты необходимо брать из старого приближения, также умножая их на ω. Кроме того, в правую часть нужно добавить слагаемое вида $(1-\omega)Av_{ij}^{(s)}$, согласно общей формуле. Нетрудно видеть, что $v_{i-1,j}$ и $v_{i,j-1}$ встречаются в векторе v раньше, чем v_{ij}, а $v_{i+1,j}$ и $v_{i,j+1}$ – позже. Окончательно для отыскания компонент нового приближения $v^{(s+1)}$ по компонентам старого приближения $v^{(s)}$ получим следующие расчетные формулы:

$$av_{ij}^{(s+1)} = \frac{\omega}{h^2}v_{i-1,j}^{(s+1)} + \frac{\omega}{k^2}v_{i,j-1}^{(s+1)} + \frac{\omega}{h^2}v_{i+1,j}^{(s)} + \frac{\omega}{k^2}v_{i,j+1}^{(s)} + (1-\omega)av_{ij}^{(s)} + \omega f_{ij},$$
$$a = 2\left(1/h^2 + 1/k^2\right),\ i = \overline{1,n-1},\quad j = \overline{1,m-1} \tag{4.23}$$

При программной реализации метода формула (4.23) используется во внешнем цикле по j от 1 до m–1 и внутреннем цикле по i от 1 до n–1. Порядок циклов изменять нельзя, иначе компоненты приближения будут меняться не в том порядке, в каком они следуют друг за другом в векторе v.

4.3.3. Организация параллельных вычислений

Рассмотрим способ построения параллельного алгоритма, который выполнял бы только те вычислительные действия, что и последовательный метод (быть может в некотором ином порядке) и, как результат, обеспечивал бы получение точно таких же решений исходной вычислительной задачи. Как уже было отмечено выше, в последовательном алгоритме каждое очередное k-ое приближение значения v_{ij} вычисляется по последнему k-ому приближению значений $v_{i-1,j}$ и $v_{i,j-1}$ и предпоследнему (k–1)-ому приближению значений $v_{i+1,j}$ и $v_{i,j+1}$. При требовании совпадения результатов вычислений последовательных и параллельных вычислительных схем в начале каждой итерации метода только одно значение v_{11} может быть пересчитано (возможности для распараллеливания нет). Но далее после пересчета v_{11} вычисления могут выполняться уже в двух узлах сетки v_{12} и v_{21} (в этих узлах выполняются условия последовательной схемы), затем после пересчета узлов v_{12} и v_{21} – в узлах v_{31}, v_{22} и v_{13} и т.д.

Обобщая сказанное, можно увидеть, что выполнение итерации метода можно разбить на последовательность шагов, на каждом из которых к вычислениям окажутся подготовленными узлы вспомогательной диагонали сетки с номером, определяемом номером этапа. Иллюстрация данного процесса приведена на рис. 4.10.

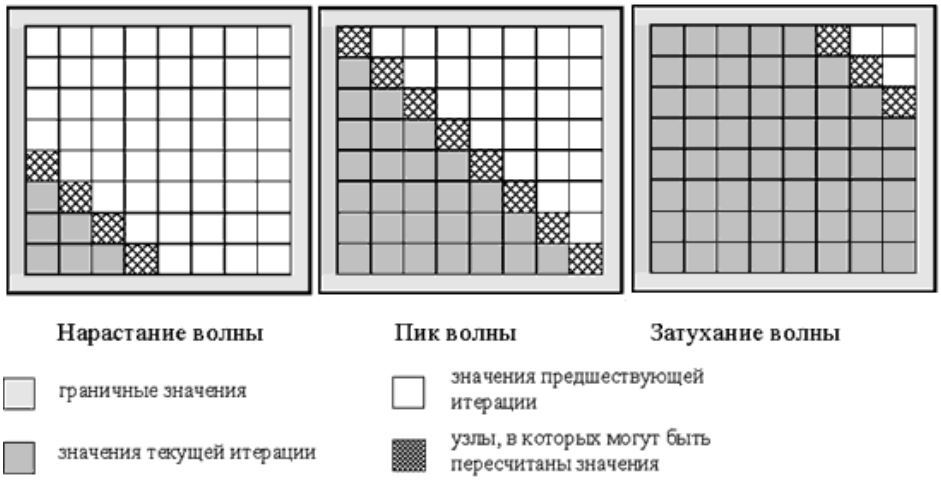


Рис. 4.10. Движение фронта волны вычислений

Получаемая в результате вычислительная схема получила наименование волны или фронта вычислений, а алгоритмы, получаемые на ее основе, – методами волновой обработки данных. В нашем случае размер волны (степень возможного параллелизма) динамически изменяется в ходе вычислений – волна нарастает до своего пика, а затем затухает при приближении к правому нижнему узлу сетки.

Следует обратить внимание на важный момент при анализе эффективности разработанного параллельного алгоритма. Фронт волны вычислений плохо соответствует правилам использования кэша процессора. В рассматриваемом нами алгоритме размещение данных в памяти осуществляется по строкам, а фронт волны вычислений располагается по диагонали сетки, и это приводит к низкой эффективности использования кэша. Один из способов улучшения ситуации – укрупнение вычислительных операций и рассмотрение в качестве распределяемых между потоками действий процедуру обработки некоторой прямоугольной подобласти (блока) сетки области расчетов. Иллюстрация приведена на рис. 4.11.

Вычисления в предлагаемом алгоритме происходят в соответствии с волновой схемой обработки данных – вначале вычисления выполняются только в левом верхнем блоке с координатами (0,0), далее для обработки становятся доступными блоки с координатами (0,1) и (1,0) и т.д.

Блочный подход к методу волновой обработки данных существенным образом меняет состояние дел – обработку узлов можно организовать построчно, доступ к данным осуществляется последовательно по элементам памяти, перемещенные в кэш значения используются многократно.

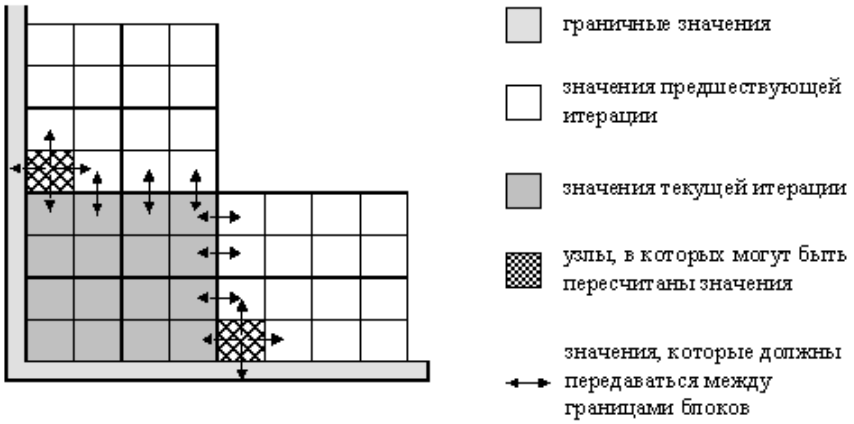


Рис. 4.11. Блочное представление волны

4.3.4. Результаты вычислительных экспериментов

Для проведения вычислительных экспериментов была сформирована тестовая задача

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=4, x\in[1,2], y\in[2,3];$$

$$u(1,y)=1+y^2,\ u(2,y)=4+y^2;$$

$$u(x,2)=x^2+4,\ u(x,3)=x^2+9.$$

Нетрудно видеть, что данная задача имеет решение

$$u(x,y)=x^2+y^2.$$

В соответствии с описанным алгоритмом была построена разностная схема (4.16), число разбиений по x и по y выбиралось одинаковое, и варьировалось в диапазоне от 250 до 1500. Отметим, что при числе разбиений 1000 система уравнений, которая соответствует данной разностной схеме, будет содержать ≈1 млн. уравнений, а коэффициент заполнения матрицы будет $\approx 10^{-6}$.

Для поиска решения построенной разностной схемы использовался метод верхней релаксации с оптимальным параметром, который вычислялся для каждой сетки в соответствии с (4.20), использовался критерий остановки по точности (2.3) с параметром $\varepsilon=10^{-6}$.

В таблице приведены данные, характеризующие работу последовательного алгоритма: число итераций s, время работы t метода и погрешность полученного решения ε в зависимости от числа разбиений n.

Табл. 4.4. Время решения задачи Дирихле последовательным алгоритмом

n	t, сек	s	ε
250	0,36	597	0,00048
500	2,95	1201	0,00045
750	10,30	1812	0,00042
1000	27,18	2424	0,00041
1250	53,65	3037	0,00040
1500	103,21	3651	0,00039

Результаты параллельных вычислительных экспериментов приведены в таблице, указано время работы T (в секундах) и ускорение S.

Табл. 4.5. Экспериментальная оценка ускорения

n	1 поток	Параллельный алгоритм							
		2 потока		4 потока		6 потоков		8 потоков	
		T	S	T	S	T	S	T	S
250	0,36	0,39	0,92	0,55	0,66	0,78	0,46	0,89	0,40
500	2,95	2,18	1,35	2,17	1,36	3,53	0,84	3,67	0,80
750	10,30	6,54	1,58	6,57	1,57	8,28	1,24	8,77	1,17
1000	27,18	16,24	1,67	14,79	1,84	16,04	1,69	16,41	1,66
1250	53,65	31,67	1,69	26,33	2,04	26,72	2,01	26,44	2,03
1500	103,21	57,06	1,81	44,65	2,31	44,85	2,30	43,27	2,39

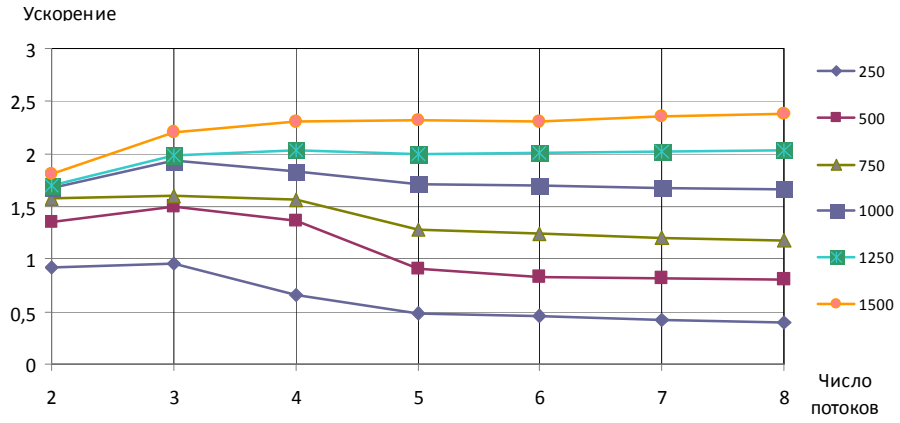


Рис. 4.12. Зависимость ускорения от числа потоков

Результаты проведенных экспериментов показывают замедление практически для всех сеток при любом числе потоков. Данный факт можно объяснить низкой трудоемкостью операций, выполняемых каждым потоком при волновой схеме вычислений. В этом случае накладные расходы (время δ, требуемое для создания и закрытия параллельной секции) превышают выигрыш от распараллеливания. Чтобы подтвердить наше предположение, проведем еще один эксперимент с большей вычислительной нагрузкой на поток.

С этой целью сформируем и решим вторую тестовую задачу

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=e^{(\sin(x)+\cos(x))}\ln\left(\left|\sin(x)+\cos(x)\right|\right),\ x\in[1,2],\ y\in[2,3];$$

$$u(1,y)=1+y^2,\ u(2,y)=4+y^2;$$

$$u(x,2)=x^2+4,\ u(x,3)=x^2+9.$$

В соответствии с описанным алгоритмом для численного решения задачи была построена разностная схема (4.16), число разбиений по x и по y выбиралось одинаковое, и варьировалось в диапазоне от 250 до 1500. Для решения разностной схемы использовался метод верхней релаксации с параметрами, аналогичными предыдущему эксперименту. Время работы метода T и ускорение S, полученные при решении второй тестовой задачи, приведены ниже, время работы указано в секундах.

Табл. 4.6. Ускорение с вычислительной нагрузкой

n	1 поток	Параллельный алгоритм							
		2 потока		4 потока		6 потоков		8 потоков	
		T	*S*	*T*	*S*	*T*	*S*	*T*	*S*
250	2,52	1,44	1,75	1,07	2,36	1,23	2,04	1,22	2,07
500	20,78	11,11	1,87	7,54	2,76	6,68	3,11	6,16	3,37
750	72,03	37,49	1,92	23,97	3,00	19,31	3,73	17,05	4,22
1000	176,98	91,84	1,93	53,61	3,30	42,09	4,20	35,83	4,94
1250	351,10	182,37	1,93	108,12	3,25	80,59	4,36	65,08	5,39
1500	629,50	324,86	1,94	184,53	3,41	137,81	4,57	115,64	5,44

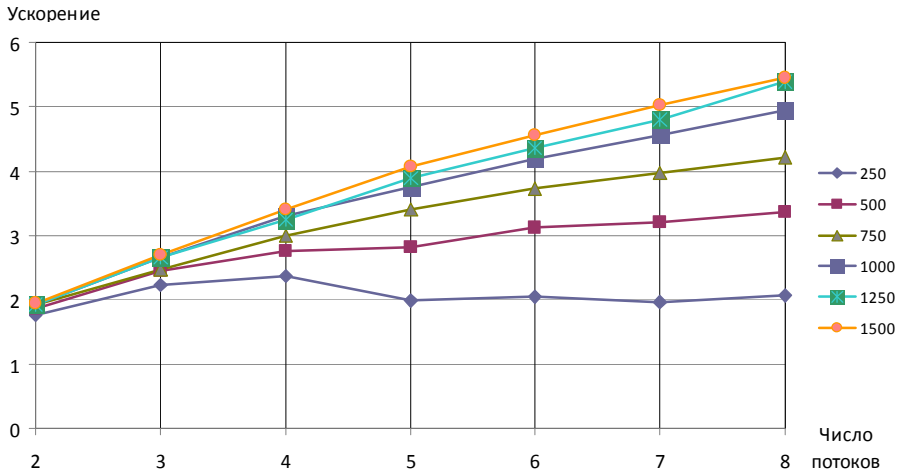


Рис. 4.13. Ускорение с вычислительной нагрузкой

Результаты проведенного эксперимента показывают, что ускорение будет практически линейным для больших сеток (при n>500), при меньших размерах все еще будет сказываться малая вычислительная нагрузка на поток, что может привести или к слабому росту ускорения (как в случае n=500), или даже к замедлению процесса решения (при n=250). Следует отметить, что данный эффект может быть сглажен при применении блочной схемы в соответствии с (4.11).

5. Параллельные методы Монте-Карло

Методы Монте-Карло можно определить как численные методы решения математических задач при помощи моделирования случайных величин. Данное определение подчеркивает, что речь идет о численных методах (и конкурировать они могут с классическими численными методами, а не с аналитическими методами решения задач); и решать методами Монте-Карло можно не только задачи вероятностного происхождения, связанные со случайными величинами. Название «Монте-Карло» произошло от города Монте-Карло (княжество Монако), известного своим казино, ибо одним из простейших приборов для генерирования случайных чисел служит рулетка.

Официальной датой рождения методов Монте-Карло считают 1949 год, когда появилась статья под заглавием «Метод Монте-Карло» [20]. Возникновение метода связывают обычно с именами Дж. Неймана, С. Улама, Н. Метрополиса; все они в 40-х годах работали в Лос-Аламосе в составе американского атомного проекта. Необходимо сразу же подчеркнуть, что теоретические основы методов Монте-Карло были известны значительно раньше. Более того, фактически такие методы не раз использовались для расчетов в математической статистике.

Однако до появления компьютеров методы Монте-Карло не могли стать универсальными численными методами, ибо моделирование случайных величин вручную – весьма трудоемкий процесс. Развитию методов Монте-Карло способствовало стремительное развитие вычислительной техники. Алгоритмы Монте-Карло сравнительно легко программируются и позволяют рассчитывать многие задачи, недоступные для классических численных методов. В настоящее время основной акцент в развитии вычислительной техники делается на многопроцессорные системы, поэтому стремительно развиваются параллельные методов Монте-Карло, расширяется область их применения.

Важнейший прием построения методов Монте-Карло – сведение задачи к расчету математических ожиданий. Для того чтобы приближенно вычислить некоторую скалярную величину a, надо придумать такую случайную величину ξ, что $M\xi=a$; тогда, вычислив N независимых значений $\xi_1,\ldots,\xi_N$, величины ξ, можно считать, что

$$a \approx \frac{\xi_1 + \xi_2 + \ldots + \xi_N}{N}.$$

5.1. Вычисление определенного интеграла

В качестве примера, поясняющего суть метода, рассмотрим задачу вычисления двумерного определенного интеграла (на случай большей размерности метод обобщается очевидным образом).

Обозначим через G произвольную область (ограниченную или неограниченную, связную или несвязную) пространства R^2. Точки пространства будем обозначать одной буквой $P=(x, y)$, а элемент объема $dP=dxdy$. Рассмотрим задачу о приближенном вычислении интеграла

$$I = \int_G f(P)\,p(P)dP\,. \tag{5.1}$$

где $p(P)$ – некоторая заданная плотность вероятностей, определенная в G, так что $\int_G p(P)dP = 1$. Можно отметить, что любой интеграл $\int_G f(P)dP$ по ограниченной области G можно считать интегралом вида (5.1). Действительно, если меру области G обозначить через SG, то $p_1(P) = 1/S_G$ при $P \in G$ представляет собой плотность вероятностей случайной точки, равномерно распределенной в G. Если ввести функцию $f_1(P) = S_G f(P)$ то, очевидно,

$$\int_G f(P)dP = \int_G f_1(P)\,p_1(P)dP\,.$$

Чтобы построить метод Монте-Карло для расчета интеграла (5.1), рассмотрим случайную точку ξ с плотностью $p(P)$ и введем скалярную случайную величину $Z=f(\xi)$, математическое ожидание которой равно искомому значению интеграла, т.е.

$$M(Z) = \int_G f(P)\,p(P)dP = I\,.$$

Предположим, что получено N случайных независимых в совокупности точек $P_1,\ldots, P_N$, являющихся реализациями случайной величины ξ и пусть $z_1=f(P_1),\ldots, z_N=f(P_N)$, тогда оценкой интеграла (5.1) служит величина

$$S_N = \frac{1}{N}\sum_{i=1}^{N} z_i .$$

Согласно закону больших чисел [20], S_N сходится по вероятности к искомому значению интеграла I.

Оценим теперь погрешность метода. Пусть случайная величина Z имеет дисперсию $D(Z)$, где $D(Z)=M(Z^2)–M(Z)^2$. Как правило, значение дисперсии $D(Z)$ неизвестно, однако в большинстве задач эту величину нетрудно оценить эмпирически, в ходе расчетов. В самом деле, достаточно одновременно с вычислением $\sum_{i=1}^{N} z_i$ вычислять и $\sum_{i=1}^{N} z_i^2$, а так как при больших N

$$M(Z^2) \approx \frac{1}{N}\sum_{i=1}^{N} (z_i)^2 ,$$

то в итоге дисперсию можно оценить как

$$D(Z) \approx \frac{1}{N}\sum_{i=1}^{N} (z_i)^2 - \left(\frac{1}{N}\sum_{i=1}^{N} z_i\right)^2 .$$

Из курса теории вероятностей известно [20], что последовательность одинаково распределенных независимых случайных величин с конечными дисперсиями подчиняется центральной предельной теореме, откуда следует, что случайная величина

$$\frac{S_N - I}{\sqrt{D(Z)/N}} .$$

распределена асимптотически нормально с функцией распределения

$$\Phi(y) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y} \exp\left(-\frac{t^2}{2}\right) dt .$$

Вероятность того, что случайная величина с такой функцией распределения не превосходит по модулю значения $y>0$, асимптотически равна

$$p_0(y) = 1 - \frac{\sqrt{2}}{\sqrt{\pi}} \int_{y}^{\infty} \exp\left(-\frac{t^2}{2}\right) dt .$$

Таким образом, при больших N погрешность вычисления значения интеграла с вероятностью, близкой к $p_0(y)$, составит

$$\left|S_N - I\right| \le y\sqrt{D(Z)/N} .$$

Полагая *y*=3 и *y*=5 находим, что неравенства

$$\left|S_N - I\right| \le 3\sqrt{D(Z)/N}\ ,\ \left|S_N - I\right| \le 5\sqrt{D(Z)/N}\ .$$

выполняются с вероятностями 0.997 и 0.99999.

Важность полученной оценки состоит в следующем: порядок скорости сходимости метода Монте-Карло есть $O\left(1/\sqrt{N}\right)$ и не зависит от кратности интеграла. Для сравнения рассмотрим порядок гарантированной оценки скорости сходимости квадратурных формул. Известно [2], что не существует методов с оценкой погрешности на классе непрерывно дифференцируемых функций лучшей, чем $CM^{-1/s}$, где C – некоторая константа, s – кратность вычисляемого интеграла, M – число узлов интегрирования. Предположим, что требуется гарантировать оценку погрешности меньшую, чем ε=0.01C. Тогда число узлов интегрирования M должно удовлетворять неравенству $CM^{-1/s} \le 0.01C$, т.е. $M \ge 100^s$. Очевидно, что уже при малых s это требование становится практически невыполнимым.

С другой стороны, если достаточно оценить погрешность с вероятностью 0.997, то число случайных точек должно удовлетворять неравенству

$$3\sqrt{D(Z)/N} \le \varepsilon\ , \tag{5.2}$$

откуда получаем оценку $N \ge 9D(Z)\varepsilon^{-2}$, которая не зависит от кратности интеграла.

5.2. Способы уменьшения дисперсии

Из формулы (5.2) видно, что вероятная ошибка оценки пропорциональна $\sqrt{D(Z)/N}$. Скорость убывания этой ошибки с ростом N невелика. Поэтому очень важно научиться выбирать для расчета интегралов такие вычислительные схемы или, другими словами, такие случайные величины Z, для которых дисперсия $D(Z)$ по возможности мала. Часто способы построения таких схем называют *способами уменьшения дисперсии*, имея в виду, что для этих способов дисперсия должна быть меньше, чем дисперсия обычного метода Монте-Карло.

Среди методов уменьшения дисперсии можно отметить: метод обратной случайной величины, метод выборки по важности, метод контрольных величин. Подробное описание указанных методов выходит за рамки данного

пособия, интересующиеся читатели могут получить полную информацию в работе [11].

5.3. Генераторы псевдослучайных чисел

Методы Монте-Карло существенным образом зависят от сгенерированной последовательности случайных чисел. На первый взгляд, получить такую последовательность не представляет труда. Можно использовать либо базу данных (таблицу) случайных чисел, либо физический генератор, либо программный генератор псевдослучайных чисел.

Отметим области применения указанных способов генерации случайной последовательности.

Возможность использования базы данных следует отклонить. В соответствии с оценкой (5.2) для получения решения методом Монте-Карло с точностью ε нужно получить порядка ε^{-2} случайных чисел, что определяет минимальный размер базы данных. Для работы в многопроцессорной системе размер такой предполагаемой базы нужно увеличить в p раз, где p – число вычислительных ядер. И если такая база будет организована, то ввод данных из нее займет неприемлемое время.

Одной из областей применения базы случайных чисел могут служить задачи, напрямую не связанные с методами Монте-Карло. Например, один из известных подходов к оценке эффективности методов глобальной оптимизации основан на численном решении этими методами всех задач из некоторой случайно генерируемой выборки. Размер такой выборки обычно не превышает нескольких тысяч задач, поэтому случайные коэффициенты, определяющие задачи, допустимо хранить в базе данных.

Рассмотрим возможность применения генераторов случайных чисел, основанных на физических принципах. Некоторые материнские платы содержат такие генераторы в виде отдельных микросхем. Основой аппаратной генерации может служить измерение сопротивления резистора или емкости конденсатора. Известно, что если проводить последовательно такие измерения, то мы будем каждый раз получать новое значение, отличающееся от предыдущего в последних разрядах (явление «теплового шума»). Их-то и можно использовать для генерации случайной последовательности.

Достоинством такого генератора является действительная случайность получаемых значений, в отличие от последовательностей псевдослучайных чисел, получаемых с помощью некоторого программно реализуемого алгоритма. Но одновременно это является также и недостатком. Ведь при каждом запуске такого генератора мы будем получать разные последователь-

ности, т.к. с физического датчика каждый раз будут поступать новые числа. Это существенным образом затрудняет отладку приложения, использующего подобный генератор: результаты двух запусков программ будут различными. Конечно, они будут похожи, если программа написана корректно, но точного совпадения не будет.

Также нельзя быть уверенным в том, что свойства физического генератора остаются неизменными с течением времени. Нельзя быть уверенным в том, что увеличение температуры окружающей среды, например, на 10С, не повлечет за собой изменение свойств последовательности. И тогда алгоритм, протестированный «вчера» на одной последовательности, будет плохо работать «сегодня» на измененной.

В силу этих двух причин мы в дальнейшем не будем рассматривать генераторы случайных чисел, основанные на физических принципах, а сконцентрируемся на рассмотрении алгоритмических генераторов, которые обычно называют *генераторами псевдослучайных чисел* (ПСЧ).

5.3.1. Линейный конгруэнтный генератор

Линейный конгруэнтный генератор порождает последовательность целых чисел $X_n \in [0, m)$, основанную на соотношении

$$X_{n+1} = (aX_n + c) \bmod m, \tag{5.3}$$

где $a>0$, $c\geq 0$, $m>0$ – некоторые целочисленные константы. Данная схема генерирования впервые была предложена в начале 50-х годов, и является одной из самых распространенных. Исследованию линейного конгруэнтного генератора посвящены многие работы, среди которых следует отметить книгу [13].

Свойства последовательности, порождаемой линейным конгруэнтным генератором, определяются числами a, c и m, а конкретный ее вид – стартовым значением X_0. Обычно стартовое значение задает пользователь.

Известно, что последовательность чисел, генерируемая таким алгоритмом, периодична с периодом, не превышающим m. При этом длина периода равна m тогда и только тогда, когда выполнены следующие условия:

- c и m взаимно просты;
- a–1 кратно p для всех простых p – делителей m;
- a–1 кратно 4, если m кратно 4.

Следствием из приведенного утверждения является тот факт, что при использовании 32-разрядной арифметики период последовательности будет

равен «всего лишь» $2^{32}\approx 10^9$, что явно недостаточно для современных вычислительных систем. Поэтому рекомендуется применять 48-и или 64-х разрядную арифметику для получения последовательности с большим периодом.

Линейный конгруэнтный генератор может быть легко применен для получения последовательности вещественных чисел. Так как правило (5.3) порождает целые числа $X_i \in [0, M-1]$, то значения $x_i = X_i/M$ будут равномерно распределены в интервале [0,1).

Недостатки линейного конгруэнтного генератора хорошо известны и описаны в литературе [13]. Пусть генератор используется для порождения случайных точек в d-мерном пространстве (в качестве координат точки можно рассматривать d подряд идущих чисел последовательности). Тогда полученные d-мерные точки расположены не более чем в $\sqrt[d]{m}$ гиперплоскостях. Для иллюстрации на рис. 5.1 приведен пример размещения случайных точек в двумерном пространстве, полученных с помощью генератора

$$X_{n+1} = (845X_n + 2625) \bmod 1024. \tag{5.4}$$

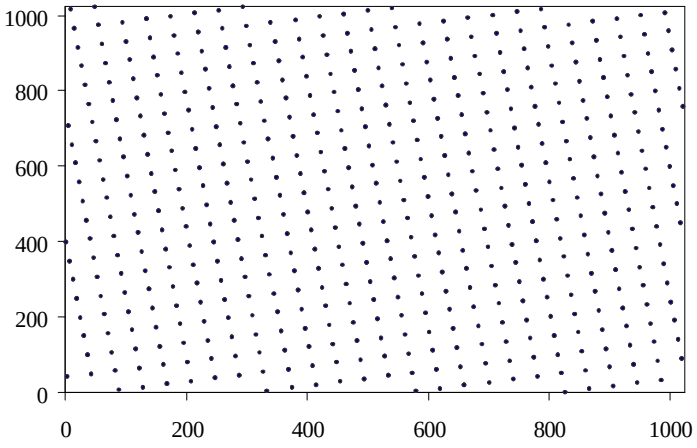


Рис. 5.1. Точки, порожденные ЛК-генератором

Из приведенной иллюстрации видно, что точки лежат на регулярной решетке.

На рис. 5.2 приведен пример размещения точек, полученных с помощью генератора с большим модулем

$$X_{n+1} = (845X_n + 2625) \bmod 8192. \tag{5.5}$$

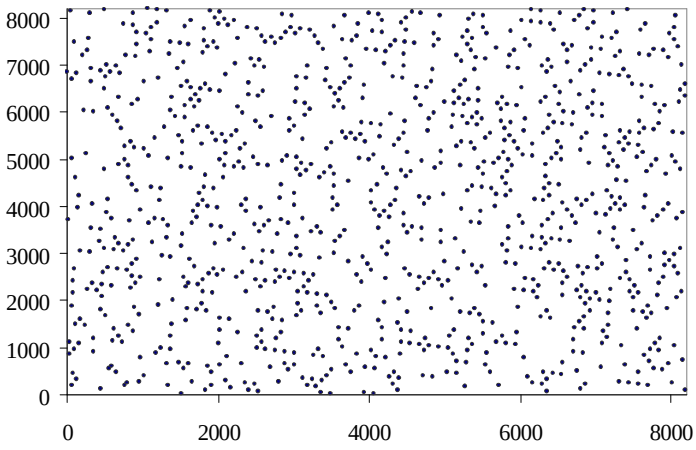


Рис. 5.2 Точки, порожденные ЛК-генератором

Видно, что точки лежат менее регулярно. Но даже если взять модуль 232, то все равно точки будут лежать на гиперплоскостях, только самих гиперплоскостей будет больше. В практических приложениях это допустимо, но нужно иметь в виду, что в многомерной области будут такие места, куда точки никогда не попадут, сколько бы их не взяли.

5.3.2. Генератор Фибоначчи с запаздываниями

В общем виде генератор Фибоначчи определяется следующим соотношением:

$$X_n = X_{n-p} * X_{n-q}. \tag{5.6}$$

где $p>q$ – целые положительные числа, называемые запаздываниями (лагами), а символ $*$ обозначает одну из операций: сложение по модулю M, вычитание по модулю M, умножение по модулю M, исключающее или. Генерируемые случайные числа могут быть как целыми (в этом случае модуль $M>1$), так и вещественными ($M=1$).

Например, для генерации вещественных чисел $X_n \in [0,1)$ можно использовать генератор вида

$$X_n = \begin{cases} X_{n-p} - X_{n-q}, & X_{n-p} \geq X_{n-q}; \\ X_{n-p} - X_{n-q} + 1, & X_{n-p} < X_{n-q}. \end{cases} \tag{5.7}$$

В отличие от линейного конгруэнтного генератора, для инициализации которого нужно было только одно значение, для генератор Фибоначчи требует p стартовых значений X_0, X_1,…, X_{p-1}, которые можно получить с помощью упомянутого конгруэнтного генератора. Для получения очередного числа генератору требуется помнить p предыдущих сгенерированных случайных чисел. При программной реализации для хранения сгенерированных случайных чисел можно использовать конечную циклическую очередь на базе массива.

Получаемые случайные числа обладают хорошими статистическими свойствами, причём все биты случайного числа равнозначны по статистическим свойствам. Период T фибоначчиева генератора может быть оценен по следующей формуле:

$$T = \left(2^p - 1\right)2^{l-1} \tag{5.8}$$

где l – количество бит в мантиссе вещественного числа.

5.3.3. Генератор Mersenne Twister

В настоящее время широкое распространение получил генератор *Mersenne twister* (вихрь Мерсенна), предложенный в работе [23]. Его достоинствами являются колоссальный период (2^{19937}–1), равномерное распределение в 623 измерениях, быстрая генерация случайных чисел (в 2-3 раза быстрее, чем линейный конгруэнтный генератор). Однако существуют сложные алгоритмы, распознающие последовательность, порождаемую с помощью Mersenne twister, как неслучайную. Это, в частности, делает Mersenne twister неподходящим для криптографии.

Описание алгоритма является довольно большим, желающие могут ознакомиться с ним в исходной работе [23]. Здесь же отметим, что данный генератор реализован во многих математических библиотеках, в частности, в Intel MKL.

5.3.4. Генератор Соболя

При решении некоторых задач методом Монте-Карло требуется, чтобы при генерации равномерного распределения обеспечивалось более равномерное покрытие области значений, частично жертвуя при этом «случайностью». Числа, полученные в результате таких действий, называются квазислучайными (КСЧ). В ряде задач (например, в задаче численного интегрирования) применение последовательностей КСЧ обеспечивает существенно лучшую сходимость по сравнению с реализациями метода Монте-Карло, основанными на использовании ПСЧ.

В качестве примера рассмотрим генератор КСЧ, основанный на так называемой $ЛП_\tau$-*последовательности*. Детальное описание способа построения и свойств указанной последовательности можно найти в работе [12]. Здесь же приведем общую схему вычисления *n*-мерных квазислучайных точек вида

$$Q_i = \left(q_{i,1},\ldots,q_{i,n}\right),\ i=1,2,\ldots \tag{5.9}$$

Для этого представим номер i в двоичной системе как

$$i = e_m, e_{m-1},\ldots e_2, e_1,$$

т.е. соответствующее число в десятичной системе будет

$$i = e_m 2^{m-1} + e_{m-1} 2^{m-2} + \ldots + e_2 2^1 + e_1.$$

Тогда координаты $q_{i,j}$ точек из (5.9) можно получить в соответствии с формулой

$$q_{i,j} = e_1 V_j^{(1)} \oplus e_2 V_j^{(2)} \oplus \ldots \oplus e_m V_j^{(m)},$$

где $\oplus$ обозначает операцию «побитовое исключающее или», а числа $V_j^{(s)}$, $1 \le j \le n$, $1 \le s \le m$, которые можно интерпретировать как координаты направляющих точек $V^{(1)}, V^{(2)},\ldots,V^{(m)}$, – некоторые заданные числа. Следует отметить, что количество чисел $V_j^{(s)}$ линейно зависит от размерности *n* генерируемых точек, и логарифмически – от их количества.

В качестве иллюстрации приведем диаграмму расположения случайных и псевдослучайных точек в двумерной области. Данный рисунок наглядно подтверждает большую равномерость покрытия области квазислучайными точками по сравнению с псевдослучайными.

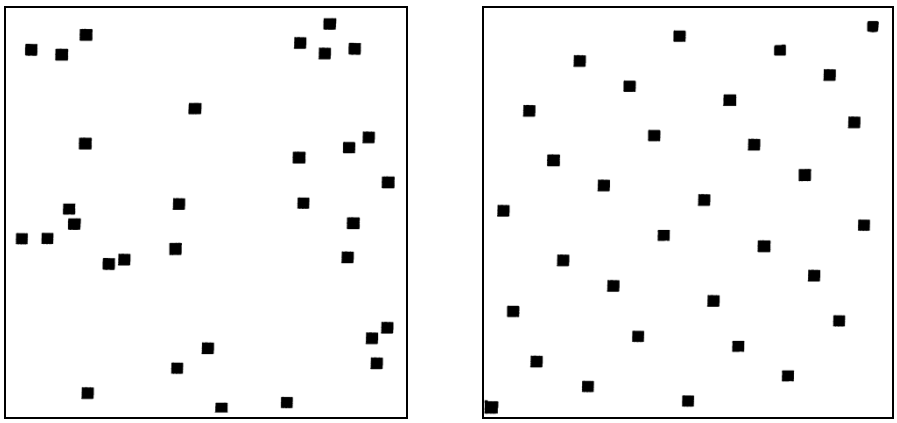


Рис. 5.3 Псевдо- и квазислучайные точки

5.4. Подходы к распараллеливанию методов Монте-Карло

Параллельные методы Монте-Карло существенным образом зависят от возможности генератора ПСЧ сгенерировать достаточно много «хороших» последовательностей. В дополнение к свойствам, которым должен обладать последовательный генератор, должны выполняться также следующие свойства:

- *Независимость*: числа из различных последовательностей должны быть не коррелированны.
- *Масштабируемость*: нужно уметь генерировать столько последовательностей, сколько нужно для работы с большим числом потоков.
- *Локальность*: поток должен иметь возможность породить новую последовательность псевдослучайных чисел без взаимодействия с другими потоками.

Рассмотрим некоторые способы конструирования параллельного генератора псевдослучайных чисел из последовательного.

5.4.1. Метод «мастер-рабочий»

Очевидным способом распараллеливания метода Монте-Карло является метод «мастер-рабочий». Поток-мастер, используя единственный генератор, порождает последовательность случайных чисел, и распределяет полученные числа по рабочим потокам.

Можно отметить несколько недостатков метода «мастер-рабочий», первый из которых заключается в следующем. Некоторые генераторы порождают последовательности, автокоррелированные с большим лагом. Но так как каждый поток получает числа из единственной последовательности, корреляция с большим лагом в одной последовательности может стать корреляцией с малым лагом в параллельных последовательностях.

Второй недостаток заключается в том, что метод «мастер-рабочий» плохо масштабируется на произвольное число потоков. Сложно сбалансировать скорость работы генератора случайных чисел в потоке-мастере со скоростью обработки этих чисел в потоках-рабочих. Если обработка очередного случайного числа является трудоемкой (по сравнению с его генерацией), то потоки-рабочие не будут справляться с обработкой потока поступающих чисел. В противном случае потоки-рабочие будут простаивать, ожидая, пока мастер сгенерирует очередное случайное число.

Эти недостатки делают данный подход практически неприемлемым, и в дальнейшем мы рассмотрим методы распараллеливания, при которых в каждом потоке генерируется собственная последовательность.

5.4.2. Метод с перешагиванием (leapfrog)

Метод с перешагиванием аналогичен циклической схеме распределения данных по потокам. Предположим, что метод Монте-Карло выполняется в p-поточной программе. Каждый поток имеет свою копию одного и того же генератора случайных чисел, порождающего последовательность X_n; при этом i-й поток обрабатывает каждое p-е число последовательности, начиная с X_i:

$$X_i, X_{i+p}, X_{i+2p}, \ldots \tag{5.10}$$

На рис. 5.4 изображены элементы последовательности, генерируемые методом leapfrog во 2-м потоке 4-х поточной программы.

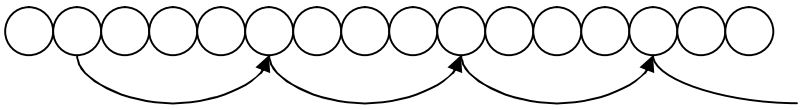


Рис. 5.4 Метод с перешагиванием

Легко модифицировать линейный конгруэнтный генератор, чтобы обеспечить параллельную генерацию по методу с перешагиванием. Используя исходную формулу (5.3), можно легко получить взаимосвязь между n-м и $(n+k)$-м числом последовательности:

$$X_{n+k} = \left[a^k X_n + \left(\frac{a^k - 1}{a - 1} \right) c \right] \bmod m . \qquad (5.11)$$

В самом деле, если выразить в явном виде (n+k)-й член последовательности через n-й, то получим

$$X_{n+1} = [aX_n + c] \bmod m ,$$

$$X_{n+2} = [aX_{n+1} + c] \bmod m = [a^2 X_n + (a+1)c] \bmod m ,$$

$$X_{n+3} = [aX_{n+2} + c] \bmod m = [a^3 X_n + (a^2 + a + 1)c] \bmod m ,$$

$$\ldots$$

$$X_{n+k} = [aX_{n+k-1} + c] \bmod m = \left[a^k X_n + c \sum_{i=0}^{k-1} a^i \right] \bmod m .$$

Принимая во внимание, что

$$a^k - 1 = (a-1) \sum_{i=0}^{k-1} a^i ,$$

получаем искомое выражение (5.11).

Часто при использовании методов Монте-Карло возникает необходимость генерировать случайные точки в многомерном пространстве. Например, для вычисления двумерного интеграла потребуется генерация пар координат (x_n, y_n). В таком случае метод с перешагиванием будет давать разные последовательности точек при разном числе потоков. Если мы хотим, чтобы параллельный метод генерировал те же самые точки, что и последовательный, то i-й поток вместо последовательности (5.10) должен генерировать последовательность

$$X_{2i}, X_{2i+1}, X_{2i+2p}, X_{2i+2p+1}, \ldots \qquad (5.12)$$

На рис. 5.5 изображены элементы последовательности, генерируемые модифицированным методом с перешагиванием во 2-м потоке 4-х поточной программы.

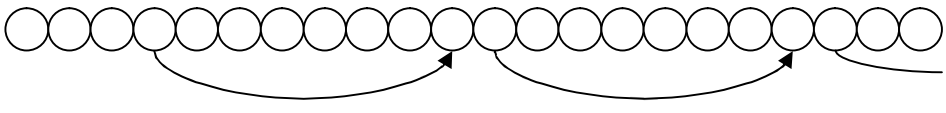


Рис. 5.5 Модифицированный метод с перешагиванием

К числу недостатков метода можно отнести следующие.

Во-первых, метод с перешагиванием не приспособлен для динамического создания новой последовательности случайных чисел в новом потоке, т.к. каждое число последовательности заранее «закреплено» за определенным потоком.

Во-вторых, даже если исходная последовательность случайных чисел была не коррелирована, то элементы полученных подпоследовательностей могут быть коррелированны для некоторых значений p. В частности, известно, что это будет происходить в случаях, когда используется линейный конгруэнтный генератор с параметром m из (5.3), являющимся степенью 2, и p также является степенью 2. Но и в других случаях корреляция на больших промежутках в исходной последовательности может превратиться в корреляцию на малых промежутках в параллельных подпоследовательностях.

5.4.3. Разделение последовательности

Метод разделения последовательности аналогичен блочной схеме распределения данных между потоками. Пусть T – период используемого генератора случайных чисел, а $N \le T$ – длина последовательности, которую мы хотим обработать. Тогда последовательность длины N разделяется на подпоследовательности равной длины по одной для каждого из p потоков. В этом случае каждый поток работает с непрерывной частью исходной последовательности.

Рис. 5.6 Разделение последовательности

Сложностью данного метода является необходимость вычислять $i\lfloor N/p \rfloor$-й член последовательности как начальный элемент в i-м потоке, что вообще говоря является трудоемкой операцией. С другой стороны, это нужно делать лишь однократно, а затем можно генерировать элементы последовательности по порядку.

Метод разделения последовательности легко можно использовать для динамического создания новой подпоследовательности. Например, поток, создающий новую подпоследовательность, может «отдать» ей половину своего интервала.

Рассмотрим использование линейного конгруэнтного генератора для целей параллельной генерации случайных чисел методом разделения последова-

тельности. В первую очередь, нужно предложить метод, который позволял бы вычислять n-й член последовательности по формуле

$$X_n = \left[a^n X_0 + \left(\frac{a^n - 1}{a - 1} \right) c \right] \bmod m . \tag{5.13}$$

не за $O(n)$ действий, а за $O(\log_2 n)$.

Покажем справедливость следующих двух выражений:

$$(a + b) \bmod m = (a \bmod m + b \bmod m) \bmod m , \tag{5.14}$$

$$(ab) \bmod m = [(a \bmod m)(b \bmod m)] \bmod m . \tag{5.15}$$

Пусть $a=km+r$, $b=tm+q$. Тогда

$$a+b=(km+r)+(tm+q)=(k+t)m+(r+q),$$

$$ab=(km+r)(tm+q)=(ktm+rt+kq)m+(rq),$$

откуда следуют равенства (5.14), (5.15).

Используя данные равенства, можно быстро вычислить коэффициент a^n из первого слагаемого формулы (5.13). Очевидно, что

$$a^n = a^{\lceil n/2 \rceil + (n - \lceil n/2 \rceil)} .$$

Далее, применяя формулу (5.15), получаем

$$a^n \bmod m = \left(a^{\lceil n/2 \rceil} \bmod m \cdot a^{n - \lceil n/2 \rceil} \bmod m \right) \bmod m .$$

Рекурсивно продолжая этот процесс, мы можем быстро получить низкие степени n, которые потом использовать для расчетов.

Например, $a^{13} = a^{7+6} = a^{4+3} a^{3+3} = \left(a^{2+2} a^{2+1} \right) \left(a^{2+1} a^{2+1} \right)$.

Этот же самый результат можно получить иначе, основываясь на идее бинарного умножения. Запишем число 13 в двоичной системе счисления

$$13_{10}=1101_2.$$

Тогда a^{13} можно вычислить через возведение a в степени двойки

$$a^{13}=a^{8+4+1}=a^8 a^4 a.$$

А данную операцию можно выполнить за время $O(\log_2 n)$.

Рассмотрим теперь алгоритм быстрого вычисления второго слагаемого в формуле (5.13). Запишем числитель в виде

$$a^n - 1 = a^t\left(a^k - 1\right) + a^t - 1.$$

Если мы выберем $n = k + t$, $k = \lfloor n/2 \rfloor$, то числитель разобьется на два слагаемых, и можно разбить дробь на две дроби вдвое меньшей степени

$$\left(\frac{a^n - 1}{a - 1}\right) c \bmod m = \left[\left(a^t \frac{a^k - 1}{a - 1} + \frac{a^t - 1}{a - 1}\right) c\right] \bmod m =$$

$$\left\{\left(\left(a^t\right)_m \left(\frac{a^k - 1}{a - 1}\right)_m + \left(\frac{a^t - 1}{a - 1}\right)_m\right) c\right\} \bmod m.$$

Рекурсивно продолжая этот процесс, получаем в итоге логарифмическую трудоемкость вычисления второго слагаемого. Это достаточно хороший способ, но можно его улучшить.

Рассмотрим вычисление значения X_2:

$$X_1 = \left(aX_0 + c\right) \bmod m.$$

$$X_2 = \left(aX_1 + c\right) \bmod m = \left(a^2 X_0 + (a+1)c\right) \bmod m.$$

Повторим эту процедуру рекурсивно, и получим значение X_4:

$$X_4 = \left(a^2 X_2 + (a+1)c\right) \bmod m.$$

Продолжая этот процесс, получаем следующие рекуррентные соотношения

$$a_0 = a,\ c_0 = c,$$

$$a_{i+1} = a_i^2 \bmod m,\ c_{i+1} = \left[(a_i + 1)c_i\right] \bmod m.$$

Далее можно опять использовать идею бинарного умножения: разложить номер n по степеням двойки

$$n = \sum_{i=0}^{b} \beta_i 2^i,\ b = \lfloor \log_2 n \rfloor,$$

после чего использовать эти коэффициенты в функциях

$$F_i(x) = \beta_i\left[\left(a_i x + c_i\right) \bmod m\right] + (1 - \beta_i)x.$$

Теперь интересующее нас число X_n можно вычислить как

$$X_n = F_0\left(F_1\left(F_2\left(\ldots F_b\left(X_0\right)\ldots\right)\right)\right).$$

Таким образом, за вызов не более чем $b = \lfloor \log_2 n \rfloor$ функций F_i можно вычислить любое число X_n, а дальше – вычислять элементы последовательности X_{n+i} по исходному соотношению (5.3), т.е. мы получаем возможность быстро «установить» i-й поток на определенную позицию, а затем генерировать числа последовательно.

5.4.4. Параметризация

Следующий подход к параллельной генерации случайных чисел состоит в использовании в разных потоках разных независимых случайных последовательностей. Например, для генерации p последовательностей можно взять p разных генераторов, основанных на разных принципах. Последовательности, полученные данным способом, будут независимыми, но это решение не будет масштабируемым, т.к. число различных генераторов невелико.

Добиться масштабируемости можно при использовании одного и того же генератора случайных чисел, но с разными входными параметрами (параметризация). Естественно, при этом нужно убедиться в том, что сгенерированные подобным образом последовательности будут независимыми. В общем случае этого гарантировать нельзя, все зависит от конкретной реализации генератора псевдослучайных чисел.

Известно, что линейный конгруэнтный генератор порождает различные последовательности в зависимости от аддитивной константы c из формулы (5.3). В работе [22] предложен способ механизм выбора константы c, с помощью которого можно порождать до 100 независимых последовательностей в разных потоках.

Генератор Фибоначчи с запаздыванием также хорошо подходит для данного подхода. Используя для каждого генератора свою таблицу стартовых значений можно получать независимые последовательности. Очевидно, что корреляция между стартовыми значениями для разных потоков приведет к коррелированным последовательностям. Поэтому для заполнения стартовых значений в разных потоках можно использовать метод с перешагиванием или метод разделения последовательности с использованием одного генератора. Примером библиотеки, в которой реализован параметризованный генератор Фибоначии, может служить SPRNG [24].

Выбор различных параметров генератора Mersenne Twister также позволяет получать множество независимых последовательностей. Библиотека Intel MKL [26] позволяет использовать параллельно 1024 генераторов Mersenne Twister, каждый из которых генерирует последовательность с периодом $2^{2203} – 1$. Сгенерированные последовательности являются независимыми в совокупности.

5.5. Результаты вычислительных экспериментов

В качестве примера рассмотрим задачу вычисления интеграла

$$C = e^{-rT} \int_{-\infty}^{+\infty} f(z)\varphi(z)dz\,,\ f(z) = \left(S_0 e^{\left(r-\frac{\sigma^2}{2}\right)T+\sigma\sqrt{T}z} - K \right)^+$$

где $\varphi(z)$ – плотность стандартного нормального распределения (нормальное распределение с мат. ожиданием 0 и дисперсией 1, обозначаемое также $N(0,1)$), а r, T, S_0, σ – числовые параметры. Для вычисления данного интеграла методом Монте-Карло можно использовать оценку

$$C \approx e^{-rT} \frac{1}{N} \sum_{i=1}^{N} f(z_i)\,,\ f(z_i) = \left(S_0 e^{\left(r-\frac{\sigma^2}{2}\right)T+\sigma\sqrt{T}z_i} - K \right)^+$$

где $\{z_i\}$ выбирается из $N(0, 1)$. Получить нормально распределенные ПСЧ можно, например, с помощью метода обратной функции или преобразования Бокса-Мюллера. Подробнее данная задача рассмотрена в лабораторной работе «Параллельные методы Монте-Карло».

Для вычисления интеграла использовались генераторы ПСЧ и КСЧ, параллельная генерация последовательностей осуществлялась методом разделения последовательности. Ниже приведены результаты, соответствующие работе этих генераторов.

Табл. 5.1. Экспериментальная оценка ускорения при использовании ПСЧ

N, млн.	1 поток	2 потока		4 потока		6 потоков		8 потоков	
	t, сек	t, сек	S	t, сек	S	t, сек	S	t, сек	S
105	3,82	1,91	1,99	0,99	3,83	0,67	5,70	0,49	7,65
210	7,65	3,82	2,00	1,96	3,89	1,34	5,71	1,01	7,55
315	11,46	5,70	2,00	2,94	3,88	2,01	5,69	1,65	6,93
420	15,27	7,67	1,99	3,94	3,87	2,68	5,69	2,15	7,09
525	19,06	9,56	1,99	4,88	3,90	3,36	5,65	2,66	7,14
630	22,9	11,49	1,99	5,88	3,89	4,04	5,66	3,15	7,26
735	26,8	13,38	2,00	6,84	3,91	4,69	5,70	3,68	7,28

840	30,62	15,38	1,99	7,83	3,91	5,35	5,72	4,14	7,38
945	34,42	17,23	1,99	8,75	3,93	6,02	5,71	4,64	7,40

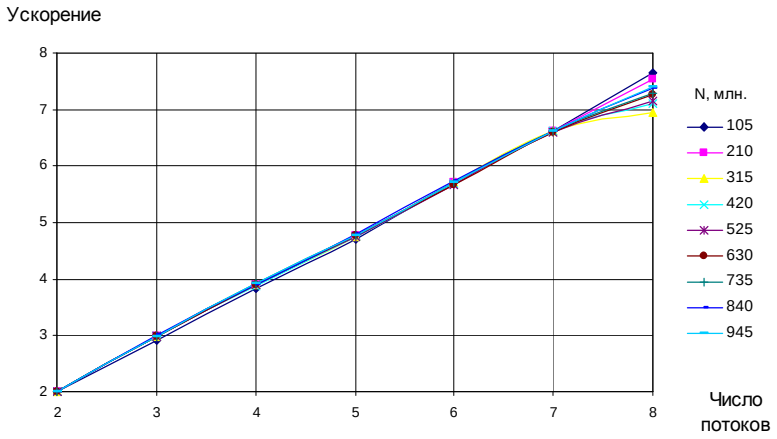


Рис. 5.7.Ускорение при использовании ПСЧ

Табл. 5.2. Экспериментальная оценка ускорения при использовании КСЧ

N, млн	1 поток	2 потока		4 потока		6 потоков		8 потоков	
	t, сек	t, сек	S	t, сек	S	t, сек	S	t, сек	S
105	4,16	2,10	1,97	1,07	3,87	0,73	5,68	0,59	7,02
210	8,33	4,18	1,99	2,12	3,92	1,46	5,68	1,10	7,52
315	12,49	6,25	1,99	3,30	3,77	2,20	5,68	1,65	7,56
420	16,64	8,34	1,99	4,21	3,95	2,93	5,67	2,21	7,51
525	20,81	10,46	1,98	5,35	3,88	3,66	5,67	2,74	7,58
630	25,02	12,52	1,99	6,38	3,92	4,41	5,66	3,30	7,56
735	29,18	14,61	1,99	7,53	3,87	5,11	5,70	3,85	7,57
840	33,30	16,73	1,99	8,56	3,88	5,85	5,69	4,41	7,54
945	37,50	18,84	1,99	9,61	3,90	6,63	5,65	4,99	7,51

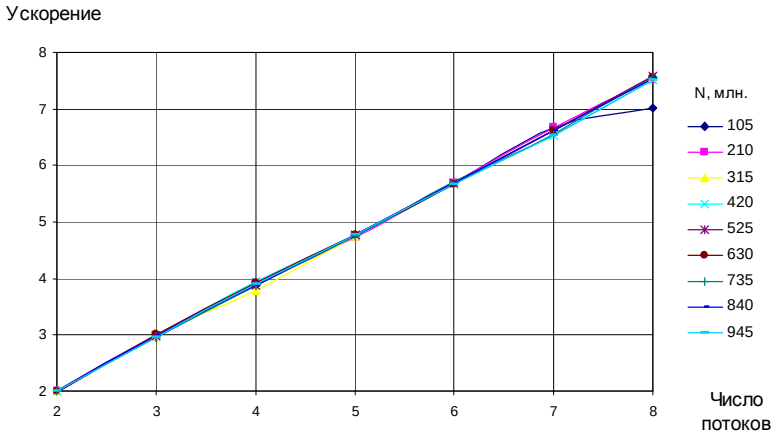


Рис. 5.8. Ускорение при использовании КСЧ

Количество повторений в проведенных экспериментах специально взято весьма большим с тем, чтобы времена работы функций составляли секунды и влияние накладных расходов (создание потоков, например) было практически нивелировано.

6. Литература

6.1. Использованные источники информации

1. Вержбицкий В.М. Численные методы (математический анализ и обыкновенные дифференциальные уравнения). – М.: Высшая школа, 2001.
2. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. – М.: Наука, 1987.
3. Тихонов А.Н., Самарский А.А. Уравнения математической физики. – М.: Наука, 1977.
4. Самарский А.А., Гулин А.В. Численные методы. – М.: Наука, 1989.
5. Самарский А.А. Введение численные методы. – СПб.: Лань, 2005.
6. Калиткин Н.Н. Численные методы. – М.: Наука, 1978
7. Хамахер К., Вранешич З., Заки С. Организация ЭВМ. –СПб: Питер, 2003.
8. Голуб Дж., Ван Лоун Ч. Матричные вычисления. – М.: Мир, 1999.
9. Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений. – М.: Мир, 1984.
10. Писсанецки С. Технология разреженных матриц. — М.: Мир,1988.
11. Соболь И.М. Численные методы Монте-Карло. – М.: Наука, 1973.
12. Соболь И.М. Точки, равномерно заполняющие многомерный куб. – М.: Знание, 1985.
13. Д. Кнут. Искусство программирования. Том 2: получисленные алгоритмы. – М.: «Вильямс», 2007.
14. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. – Н.Новгород, Изд-во ННГУ, 2003.
15. Гергель В.П. Теория и практика параллельных вычислений. – М.: БИНОМ, 2007.
16. Белов С.А., Золотых Н.Ю. Численные методы линейной алгебры. – Н.Новгород, Изд-во ННГУ, 2005.
17. J. Dongarra et al. Templates for the solution of linear systems: building blocks for iterative methods. SIAM, 1994.

18. G. Karniadakis, R. Kirby. Parallel scientific computing in C++ and MPI. Cambridge university press, 2003.

19. M. Quinn. Parallel programming in C with MPI and OpenMP. McGraw-Hill, 2004.

6.2. Дополнительная литература

20. Ширяев А. Н. Вероятность, – М.: Наука. 1989.

21. Metropolis N., Ulam S. The Monte Carlo method, J. Amer. statistical assoc., 1949, 44, N247, 335-341.

22. O. Percus, M. Kalos. Random number generators for MIMD parallel processors// Journal of parallel and distributed computing, v.6, 1989. pp. 477–479.

23. M. Matsumoto, T. Nishimura (1998). «Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator». ACM Trans. on Modeling and Computer Simulations v. 8(1).

24. M. Mascagni, A. Srinivasan. Algorithm 806: SPRNG: A scalable library for pseudorandom number generation. ACM Transactions on Mathematical Software, v. 26, № 3, 2000. pp. 436–461.

25. Niederreiter H. Random Number Generation and Quasi-Monte Carlo Methods. – SIAM, 1992. – 247 p.

6.3. Информационные ресурсы сети Интернет

26. Intel Math Kernel Library Reference Manual.

[http://software.intel.com/sites/products/documentation/hpc/mkl/mklman.pdf].