9. Решение систем линейных уравнений

Системы линейных уравнений возникают при решении ряда прикладных задач, описываемых дифференциальными, интегральными или системами нелинейных (трансцендентных) уравнений. Они могут появляться также в задачах математического программирования, статистической обработки данных, аппроксимации функций, при дискретизации краевых дифференциальных задач методом конечных разностей или методом конечных элементов и др.

Матрицы коэффициентов систем линейных уравнений могут иметь различную структуру и свойства. Матрицы решаемых систем могут быть плотными и их порядок может достигать несколько тысяч строк и столбцов. При решении многих задач могут появляться системы, обладающие симметричными положительно определёнными ленточными матрицами с порядком в десятки тысяч и шириной ленты в несколько тысяч элементов. И, наконец, при рассмотрении большого ряда задач могут возникать системы линейных уравнений с разрежёнными матрицами с порядком в миллионы строк и столбцов.

9.1. Постановка задачи

Линейное уравнение с n неизвестными $x_0, x_1, \ldots, x_{n-1}$ может быть определено при помощи выражения

$$a_0 x_0 + a_1 x_1 + \ldots + a_{n-1} x_{n-1} = b \qquad (9.1)$$

где величины $a_0, a_1, \ldots, a_{n-1}$ и b представляют собой постоянные значения.

Множество n линейных уравнений

$$\begin{aligned} a_{0,0} x_0 \quad &+ a_{0,1} x_1 \quad + \ldots + a_{0,n-1} x_{n-1} \quad = b_0 \\ a_{1,0} x_0 \quad &+ a_{1,1} x_1 \quad + \ldots + a_{1,n-1} x_{n-1} \quad = b_1 \\ &\ldots \\ a_{n-1,0} x_0 \quad &+ a_{n-1,1} x_1 \quad + \ldots + a_{n-1,n-1} x_{n-1} \quad = b_{n-1} \end{aligned} \qquad (9.2)$$

называется *системой линейных уравнений* или *линейной системой*. В более кратком (*матричном*) виде система может представлена как

$$Ax = b,$$

где $A=(a_{i,j})$ есть вещественная матрица размера $n \times n$, а вектора b и x состоят из n элементов.

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

9.2. Алгоритм Гаусса

Метод Гаусса является широко известным *прямым* алгоритмом решения систем линейных уравнений, для которых матрицы коэффициентов являются *плотными*. Если система линейных уравнений является *невырожденной*, то метод Гаусса гарантирует нахождение решения с погрешностью, определяемой точностью машинных вычислений. Основная идея метода состоит в приведении матрицы A посредством эквивалентных преобразований (не меняющих решение системы (9.2)) к треугольному виду, после чего значения искомых неизвестных может быть получено непосредственно в явном виде.

В подразделе дается общая характеристика метода Гаусса, достаточная для начального понимания алгоритма и позволяющая рассмотреть возможные способы параллельных вычислений при решении систем линейных уравнений. Более полное изложение алгоритма со строгим обсуждением вопросов точности получаемых решений может быть получено, например, в работах Березина и Жидкова (1966), Kahaner, Moler and Nash (1988), Bertsekas and Tsitsiklis (1989) и др.

9.2.1. Последовательный алгоритм

Метод Гаусса основывается на возможности выполнения преобразований линейных уравнений, которые не меняют при этом решение рассматриваемой 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 – *прямой ход* метода Гаусса – исходная система линейный уравнений при помощи последовательного исключения неизвестных приводится к верхнему треугольному виду

$$U x = c ,$$

где матрица коэффициентов получаемой системы имеет вид

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

На *обратном ходе* метода Гаусса (второй этап алгоритма) осуществляется определение значений неизвестных. Из последнего уравнения преобразованной системы может быть вычислено значение переменной x_{n-1}, после этого из предпоследнего уравнения становится возможным определение переменной x_{n-2} и т.д.

9.2.1.1 Прямой ход алгоритма Гаусса

Прямой ход метода Гаусса состоит в последовательном исключении неизвестных в уравнениях решаемой системы линейных уравнений. На итерации i, $0 \le i < n-1$, метода производится исключение неизвестной i для всех уравнений с номерами k, больших i (т.е. $i < k \le n-1$,). 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:

$$\begin{aligned} a'_{kj} &= a_{kj} - (a_{ki}/a_{ii}) \cdot a_{ij}, \\ b'_k &= b_k - (a_{ki}/a_{ii}) \cdot b_i, \end{aligned} \quad i \le j \le n-1, i < k \le n-1, 0 \le i < n-1$$

(следует отметить, что аналогичные вычисления выполняются и над вектором b).

Поясним выполнение прямого хода метода Гаусса на примере системы линейных уравнений вида:

$$\begin{aligned} x_0 &+ 3x_1 &+ 2x_2 &= 1 \\ 2x_0 &+ 7x_1 &+ 5x_2 &= 18 . \\ x_0 &+ 4x_1 &+ 6x_2 &= 26 \end{aligned}$$

На первой итерации производится исключение неизвестной x_0 из второй и третьей строки. Для этого из этих строк нужно вычесть первую строку, умноженную соответственно на 2 и 1. После этих преобразований система уравнений принимает вид:

$$
\begin{aligned}
x_0 \quad +3x_1 \quad +2x_2 \quad &= \quad 1 \\
x_1 \quad + \; x_2 \quad &= \quad 16\,. \\
x_1 \quad +4x_2 \quad &= \quad 25
\end{aligned}
$$

В результате остается выполнить последнюю итерацию и исключить неизвестную x_1 из третьего уравнения. Для этого необходимо вычесть вторую строку и в окончательной форме система имеет следующий вид:

$$
\begin{aligned}
x_0 \quad +3x_1 \quad +2x_2 \quad &= \quad 1 \\
x_1 \quad + x_2 \quad &= \quad 16\,. \\
3x_2 \quad &= \quad 9
\end{aligned}
$$

На рис. 9.1 представлена общая схема состояния данных на *i*-ой итерации прямого хода алгоритма Гаусса. Все коэффициенты при неизвестных, расположенные ниже главной диагонали и левее столбца *i*, уже являются нулевыми. На *i*-ой итерации прямого хода метода Гаусса осуществляется обнуление коэффициентов столбца *i*, расположенных ниже главной диагонали, путем вычитания строки *i*, умноженной на нужную ненулевую константу. После проведения *(n-1)* подобной итерации матрица, определяющая систему линейных уравнений, становится приведенной к верхнему треугольному виду.

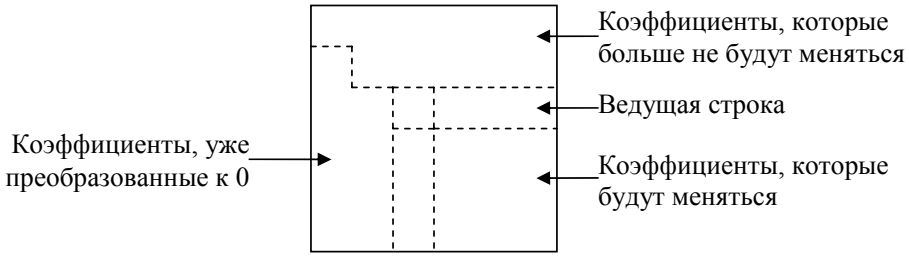

Рис. 9.1. Итерация прямого хода алгоритма Гаусса

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

Возможный способ избежать подобной проблемы может состоять в следующем – при выполнении каждой очередной итерации прямого хода метода Гаусса следует определить коэффициент с максимальным значением по абсолютной величине в столбце, соответствующем исключаемой неизвестной, т.е.

$$y = \max_{i \le k \le n-1} |a_{ki}|\,,$$

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

Вычислительная сложность прямого хода алгоритма Гаусса с выбором ведущей строки имеет порядок $O(n^3)$.

9.2.1.2 Обратный ход алгоритма Гаусса

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-1}, после этого из предпоследнего уравнения становится возможным определение переменной x_{n-2} и т.д. В общем виде, выполняемые вычисления при обратном ходе метода Гаусса могут быть представлены при помощи соотношений:

$$x_{n-1} = b_{n-1} / a_{n-1,n-1},$$

$$x_i = (b_i - \sum_{j=i+1}^{n-1} a_{ij} x_j) / a_{ii}, \;\; i = n-2, n-3, ..., 0\,.$$

Поясним, как и ранее, выполнение обратного хода метода Гаусса на примере рассмотренной в пп. 9.2.1.1 системы линейных уравнений

$$
\begin{aligned}
x_0 \quad +3x_1 \quad +2x_2 \quad &= \quad 1 \\
x_1 \quad + x_2 \quad &= \quad 16\,. \\
3x_2 \quad &= \quad 9
\end{aligned}
$$

Из последнего уравнения системы можно определить, что неизвестная x_2 имеет значение 3. В результате становится возможным разрешение второго уравнения и определение значение неизвестной x_1=13, т.е.

$$\begin{aligned} x_0 \quad +3x_1 \quad +2x_2 \quad &= \quad 1 \\ x_1 \qquad\quad &= \quad 13 \\ x_2 \quad &= \quad 3 \end{aligned}.$$

На последней итерации обратного хода метода Гаусса определяется значение неизвестной x_0, равное -44.

С учетом последующего параллельного выполнения можно отметить, учет получаемых значений неизвестных может выполняться сразу во всех уравнениях системы (и эти действия могут выполняться в уравнениях одновременно и независимо друг от друга). Так, в рассматриваемом примере после определения значения неизвестной x_2 система уравнений может быть приведена к виду

$$\begin{aligned} x_0 \quad +3x_1 \qquad &= \quad -5 \\ x_1 \qquad\quad &= \quad 13 \\ x_2 \quad &= \quad 3 \end{aligned}$$

Вычислительная сложность обратного хода алгоритма Гаусса составляет $O(n^2)$.

9.2.2. Определение подзадач

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.

9.2.3. Выделение информационных зависимостей

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, $0 \le i < n-1$, 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. Поскольку строки матрицы A распределены по подзадачам, для поиска максимального значения подзадачи с номерами k, $k > i$, должны обменяться своими элементами при исключаемой переменной x_i. После сбора всех необходимых данных в каждой подзадаче может быть определено, какая из подзадач содержит ведущую строку и какое значение является ведущим элементом.

Далее для продолжения вычислений ведущая подзадача должна разослать свою строку матрицы A и соответствующий элемент вектора b всем остальным подзадачам с номерами k, $k > i$. Получив 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, $0 \le i < n-1$, определяет значение своей переменной x_i, это значение должно быть разослано всем подзадачам с номерами k, $k < i$. Далее подзадачи подставляют полученное значение новой неизвестной и выполняют корректировку значений для элементов вектора b.

9.2.4. Масштабирование и распределение подзадач по процессорам

Выделенные базовые подзадачи характеризуются одинаковой вычислительной трудоемкостью и сбалансированным объемом передаваемых данных. В случае, когда размер матрицы, описывающей систему линейных уравнений, оказывается большим, чем число доступных процессоров (т.е., $p < n$), базовые подзадачи можно укрупнить, объединив в рамках одной подзадачи несколько строк матрицы. Однако применение использованной в разделах 7 и 8 последовательной схемы разделения данных для параллельного решения систем линейных уравнений приведет к неравномерной вычислительной нагрузке процессоров – по мере исключения (на прямом ходе) или определения (на обратном ходе) неизвестных в методе Гаусса для все большей части процессоров все необходимые вычисления будут завершены и процессоры окажутся простаивающими. Возможное решение проблемы балансировки вычислений может состоять в использовании ленточной циклической схемы (см. раздел 7) для распределения данных между укрупненными подзадачами. В этом случае матрица A делится на наборы (полосы) строк вида (см. рис. 9.2):

$A = (A_0, A_2, ..., A_{p-1})^T$, $A_i = (a_{i_0}, a_{i_1}, ..., a_{i_{k-1}})$, $i_j = i + jp$, $0 \le j < k$, $k = m / p$.

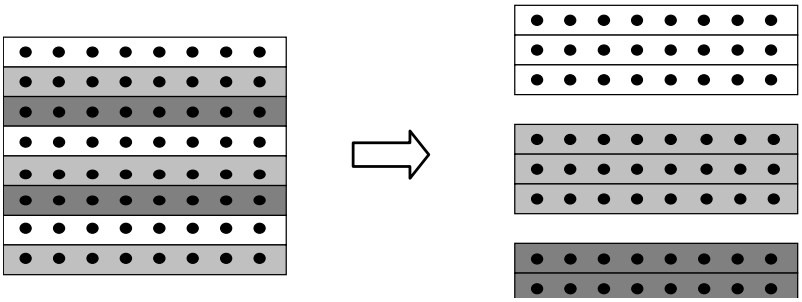

Рис. 9.2. Пример использования ленточной циклической схемы разделения строк матрицы между тремя процессорами

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.

Распределение подзадач между процессорами должно учитывать характер выполняемых в методе Гаусса коммуникационных операций. Основным видом информационного взаимодействия подзадач является операция передачи данных от одного процессора всем процессорам вычислительной системы. Как результат, для эффективной реализации требуемых информационных взаимодействий между базовыми подзадачами топология сети передачи данных должны иметь структуру гиперкуба или полного графа.

9.2.5. Анализ эффективности

Оценим трудоемкость рассмотренного параллельного варианта метода Гаусса. Пусть, как и ранее, n есть порядок решаемой системы линейных уравнений, а p, $p<n$, обозначает число используемых процессоров. Тем самым, матрица коэффициентов A имеет размер $n \times n$ и, соответственно, n/p есть размер полосы матрицы A на каждом процессоре.

Прежде всего, несложно показать, что общее время выполнения последовательного варианта метода Гаусса составляет:

$$T_1 = 2n^3 / 3 + n^2 . \qquad (9.3)$$

Определим теперь сложность параллельного варианта метода Гаусса. При выполнении **прямого хода** алгоритма на каждой итерации для выбора ведущей строки каждый процессор должен осуществить выбор максимального значения в столбце с исключаемой неизвестной в пределах своей полосы. Начальный размер полос на процессорах равен n/p, но по мере исключения неизвестных количество строк в полосах для обработки постепенно сокращается. Текущий размер полос приближенно можно оценить как $(n-i)/p$, где i, $0 \le i < n-1$, есть номер выполняемой итерации прямого хода метода Гаусса. Далее после сбора полученных максимальных значений, определения и рассылки ведущей строки, каждый процессор должен выполнить вычитание ведущей строки из каждой строки оставшейся части строк своей полосы матрицы A. Количество элементов строки, подлежащих обработке, также сокращается при исключении неизвестных, и текущее число элементов строки для вычислений оценивается величиной $(n-i)$. Тем самым, сложность процедуры вычитания строк оценивается как $2 \cdot (n-i)$ операций (перед вычитанием ведущая строка умножается на масштабирующую величину a_{ik}/a_{ii}). С учетом выполняемого количества итераций, общее число операций параллельного варианта прямого хода метода Гаусса определяется выражением:

$$T_p^1 = \sum_{i=0}^{n-2} \left[\frac{(n-i)}{p} + \frac{(n-i)}{p} \cdot 2(n-i) \right] = \frac{1}{p} \sum_{i=0}^{n-2} \left[(n-i) \cdot + 2(n-i)^2 \right].$$

На каждой итерации **обратного хода** алгоритма Гаусса после рассылки вычисленного значения очередной неизвестной каждый процессор должен обновить значения правых частей для всех строк, расположенных на этом процессоре. Отсюда следует, что трудоемкость параллельного варианта обратного хода алгоритма Гаусса оценивается как величина:

$$T_p^2 = \sum_{i=0}^{n-2} 2 \cdot (n-i) / p .$$

Просуммировав полученные выражения можно получить

$$T_p = \frac{1}{p}\sum_{i=0}^{n-2}\left[(n-i)\cdot +2(n-i)^2\right] + \frac{2}{p}\sum_{i=0}^{n-2}(n-i) = \frac{1}{p}\sum_{i=0}^{n-2}\left[3(n-i)+2(n-i)^2\right] = \frac{1}{p}\sum_{i=2}^{n}(3i+2i^2)\,.$$

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

$$S_p = \frac{(2n^3/3+n^2)}{\frac{1}{p}\sum_{i=2}^{n}(3i+2i^2)}, \qquad E_p = \frac{(2n^3/3+n^2)}{\sum_{i=2}^{n}(3i+2i^2)}\,. \tag{9.4}$$

Полученные соотношения имеют достаточно сложный вид для оценивания. Вместе с тем можно показать, что сложность параллельного алгоритма имеет порядок $\sim(2n^3/3)/p$, и, тем самым, балансировка вычислительной нагрузки между процессорами в целом является достаточно равномерной.

Дополним сформированные показатели вычислительной сложности метода Гаусса оценкой затрат на выполнение операций передачи данных между процессорами. При выполнении **прямого хода** на каждой итерации для определения ведущей строки процессоры обмениваются локально найденными максимальными значениями в столбце с исключаемой переменной. Выполнение данного действия одновременно с определением среди собираемых величин наибольшего значения может быть обеспечено при помощи операции обобщенной редукции (функция *MPI_Allreduce* библиотеки MPI). Всего для выполнения такой операции требуется $log_2 p$ шагов, что с учетом количества итераций позволяет оценить время, необходимое для проведения операций редукции, при помощи следующего выражения:

$$T_p^1(comm) = (n-1)\cdot \log_2 p \cdot (\alpha + w/\beta)\,,$$

где, как и ранее, α – латентность сети передачи данных, β – пропускная способность сети, w – размер пересылаемого элемента данных.

Далее также на каждой итерации прямого хода метода Гаусса выполняется рассылка выбранной ведущей строки. Сложность данной операции передачи данных является равной:

$$T_p^2(comm) = (n-1)\cdot \log_2 p \cdot (\alpha + wn/\beta)\,.$$

При выполнении **обратного хода** алгоритма Гаусса на каждой итерации осуществляется рассылка между всеми процессорами вычисленного значения очередной неизвестной. Общее время, необходимое для выполнения подобных действий, можно оценить как:

$$T_p^3(comm) = (n-1)\cdot \log_2 p \cdot (\alpha + w/\beta)\,.$$

Подводя итог, с учетом всех полученных выражений, трудоемкость параллельного варианта метода Гаусса составляет:

$$T_p = \frac{1}{p}\sum_{i=2}^{n}(3i+2i^2)\,\tau + (n-1)\cdot \log_2 p \cdot (3\alpha + w(n+2)/\beta)\,, \tag{9.5}$$

где τ есть время выполнения базовой вычислительной операции.

9.2.6. Программная реализация

Рассмотрим возможный вариант параллельной реализации метода Гаусса для решения систем линейных уравнений. При этом реализация отдельных модулей не приводится, если их отсутствие не оказывает влияние на понимании общей схемы параллельных вычислений.

1. Главная функция программы. Реализует логику работы алгоритма, последовательно вызывает необходимые подпрограммы.

```
// Программа 9.1 - Алгоритм Гаусса решения систем линейных уравнений
int ProcNum;            // The number of the available processes
int ProcRank;           // The rank of the current process

int *pParallelPivotPos; // The number of rows selected as the pivot ones
int *pProcPivotIter;    // The number of iterations, at which the processor
                        // rows were used as the pivot ones
void main(int argc, char* argv[]) {
  double* pMatrix;        // The matrix of the linear system
  double* pVector;        // The right parts of the linear system
  double* pResult;        // The result vector
  double *pProcRows;      // The Rows of matrix A on the process
  double *pProcVector;    // The Elements of vector b on the process
```

```
  double *pProcResult;    // The Elements of vector x on the process
  int     Size;           // The Sizes of the initial matrix and vector
  int     RowNum;         // The Number of  the matrix rows on the  current
                          // process
  MPI_Init ( &argc, &argv );
  MPI_Comm_rank ( MPI_COMM_WORLD, &ProcRank );
  MPI_Comm_size ( MPI_COMM_WORLD, &ProcNum );

  if (ProcRank == 0)
        printf("Parallel Gauss algorithm for solving linear systems\n");

  // Memory allocation and definition of object elements
  ProcessInitialization(pMatrix, pVector, pResult,
    pProcRows, pProcVector, pProcResult, Size, RowNum);

  // The execution of the parallel Gauss algorithm
  DataDistribution(pMatrix, pProcRows, pVector, pProcVector, Size, RowNum);
  ParallelResultCalculation(pProcRows, pProcVector, pProcResult, Size,
   RowNum);
  ResultCollection(pProcResult, pResult);

  // Computational process termination
  ProcessTermination(pMatrix, pVector, pResult, pProcRows, pProcVector,
    pProcResult);

  MPI_Finalize();
}
```

Следует пояснить использование дополнительных массивов. Элементы массива *pParallelPivotPos* определяют номера строк матрицы, выбираемых в качестве ведущих, по итерациям прямого хода метода Гаусса. Именно в этом порядке должны выполняться затем итерации обратного хода для определения значений неизвестных системы линейных уравнений. Массив *pParallelPivotPos* является глобальным и любое его изменение в одном из процессов требует выполнения операции рассылки измененных данных всем остальным процессам программы.

Элементы массива *pProcPivotIter* определяют номера итераций прямого хода метода Гаусса, на которых строки процесса использовались в качестве ведущих (т.е., строка *i* процесса выбиралась ведущей на итерации *pProcPivotIter[i]*). Начальное значение элементов массива устанавливается нулевым и, тем самым, нулевое значение элемента массива *pProcPivotIter[i]* является признаком того, что строка *i* процесса все еще подлежит обработке. Кроме того, важно отметить, что запоминаемый в элементах массива *pProcPivotIter* номера итераций дополнительно означают и номера неизвестных, для определения которых будут использованы соответствующие строкам уравнения. Массив *pProcPivotIter* является локальным для каждого процесса.

Функция *ProcessInitialization* определяет исходные данные решаемой задачи (число неизвестных), выделяет память для хранения данных, осуществляет ввод матрицы коэффициентов системы линейных уравнений и вектора правых частей (или формирует эти данные при помощи какого-либо датчика случайных чисел).

Функция *DataDistribution* реализует распределение матрицы линейной системы и вектора правых частей между процессорами вычислительной системы.

Функция *ResultCollection* осуществляет сбор со всех процессов отдельных частей вектора неизвестных.

Функция *ProcessTermination* выполняет необходимый вывод результатов решения задачи и освобождает всю ранее выделенную память для хранения данных.

Реализация всех перечисленных функций может быть выполнена по аналогии с ранее рассмотренными примерами и предоставляется читателю в качестве самостоятельного упражнения.

2. Функция ParallelResultCalculation. Реализует логику работы параллельного алгоритма Гаусса, последовательно вызывает функции, выполняющие прямой и обратный ход метода.

```
// Function for the execution of the parallel Gauss algorithm
void ParallelResultCalculation(double* pProcRows, double* pProcVector,
  double* pProcResult, int Size, int RowNum) {
  ParallelGaussianElimination (pProcRows, pProcVector, Size, RowNum);
  ParallelBackSubstitution (pProcRows, pProcVector, pProcResult,
```

```
    Size, RowNum);
}
```

3. Функция ParallelGaussianElimination. Функция выполняет параллельный вариант прямого хода алгоритма Гаусса.

```
void ParallelGaussianElimination (double* pProcRows, double* pProcVector,
  int Size, int RowNum) {
  double MaxValue;   // The value of the pivot element of the process
  int    PivotPos;   // The Position of the pivot row in the stripe
                     // of the process
  // Structure for the pivot row selection
  struct { double MaxValue; int ProcRank; } ProcPivot, Pivot;

  // pPivotRow is used for storing the pivot row and the corresponding
  // element of the vector b
  pPivotRow = new double [Size+1];

  // The iterations of the Gaussian elimination stage
  for (int i=0; i<Size; i++) {

    // Calculating the local pivot row
    double MaxValue = 0;
      for (int j=0; j<RowNum; j++) {
      if ((pProcPivotIter[j] == -1) &&
        (MaxValue < fabs(pProcRows[j*Size+i]))) {
        MaxValue = fabs(pProcRows[j*Size+i]);
        PivotPos = j;
        }
    }
    ProcPivot.MaxValue = MaxValue;
    ProcPivot.ProcRank = ProcRank;

    // Finding the pivot process
    // (process with the maximum value of MaxValue)
    MPI_Allreduce(&ProcPivot, &Pivot, 1, MPI_DOUBLE_INT, MPI_MAXLOC,
                  MPI_COMM_WORLD);

    // Broadcasting the pivot row
    if ( ProcRank == Pivot.ProcRank ){
      pProcPivotIter[PivotPos]= i; //iteration number
      pParallelPivotPos[i]= pProcInd[ProcRank] + PivotPos;
      }
    MPI_Bcast(&pParallelPivotPos[i], 1, MPI_INT, Pivot.ProcRank,
      MPI_COMM_WORLD);

    if ( ProcRank == Pivot.ProcRank ){
      // Fill the pivot row
      for (int j=0; j<Size; j++) {
        pPivotRow[j] = pProcRows[PivotPos*Size + j];
      }
      pPivotRow[Size] = pProcVector[PivotPos];
    }
    MPI_Bcast(pPivotRow, Size+1, MPI_DOUBLE, Pivot.ProcRank,
      MPI_COMM_WORLD);

    ParallelEliminateColumns(pProcRows, pProcVector, pPivotRow, Size,
      RowNum, i);
  }
}
```

Функция *ParallelEliminateColumns* проводит вычитание ведущей строки из строк процесса, которые еще не использовались в качестве ведущих (т.е. для которых элементы массива $pProcPivotIter$ равны нулю).

4. Функция ParallelBackSubstitution. Функция реализует параллельный вариант обратного хода Гаусса.

```
void ParallelBackSubstitution (double* pProcRows, double* pProcVector,
  double* pProcResult, int Size, int RowNum) {
  int IterProcRank;   // The rank of the process with the current pivot row
  int IterPivotPos;   // The position of the pivot row of the process
  double IterResult;  // The calculated value of the current unknown
  double val;

  // Iterations of the back substitution stage
  for (int i=Size-1; i>=0; i--) {

    // Calculating the rank of the process, which holds the pivot row
    FindBackPivotRow(pParallelPivotPos[i], Size, IterProcRank,
      IterPivotPos);

    // Calculating the unknown
    if (ProcRank == IterProcRank) {
      IterResult =
        pProcVector[IterPivotPos]/pProcRows[IterPivotPos*Size+i];
      pProcResult[IterPivotPos] = IterResult;
    }
    // Broadcasting the value of the current unknown
    MPI_Bcast(&IterResult, 1, MPI_DOUBLE, IterProcRank, MPI_COMM_WORLD);

    // Updating the values of the vector b
    for (int j=0; j<RowNum; j++)
      if ( pProcPivotIter[j] < i ) {
        val = pProcRows[j*Size + i] * IterResult;
        pProcVector[j]=pProcVector[j] - val;
    }
  }
}
```

Функция *FindBackPivotRow* определяет строку, из которой можно вычислить значение очередного элемента результирующего вектора. Номер этой строки хранится в массиве *pParallelPivotIter*. По номеру функция *FindBackPivotRow* определяет номер процесса, на котором эта строка хранится, и номер этой строки в полосе *pProcRows* этого процесса.

9.2.7. Результаты вычислительных экспериментов

Вычислительные эксперименты для оценки эффективности параллельного варианта метода Гаусса для решения систем линейных уравнений проводились при условиях, указанных в п. 7.6.5 и состоят в следующем.

Эксперименты проводились на вычислительном кластере на базе процессоров Pentium III Xeon 1000 Мгц и сетью Gigabit Ethernet под управлением операционной системы Microsoft Windows 2000 (см. п. 1.2.3).

Для оценки длительности τ базовой скалярной операции проводилось решение системы линейных уравнений при помощи последовательного алгоритма и полученное таким образом время вычислений делилось на общее количество выполненных операций – в результате подобных экспериментов для величины τ было получено значение 1,5 нсек. Эксперименты, выполненные для определения параметров сети передачи данных, показали значения латентности α и пропускной способности β соответственно 107 мкс и 43.29 Мбайт/с. Все вычисления производились над числовыми значениями типа double, т. е. величина w равна 8 байт.

Результаты вычислительных экспериментов приведены в таблице в таблице 9.1. Эксперименты выполнялись с использованием двух, четырех и восьми процессоров.

Таблица 9.1. Результаты вычислительных экспериментов для параллельного алгоритма Гаусса

Размер системы	Последовательный алгоритм	Параллельный алгоритм					
		2 процессора		4 процессора		8 процессоров	
		Время	Ускорение	Время	Ускорение	Время	Ускорение
500	0,36	0,3302	1,0901	0,5170	0,6963	0,7504	0,4796
1000	3,313	1,5950	2,0770	1,6152	2,0511	1,8715	1,7701

1500	11,437	4,1788	2,7368	3,8802	2,9474	3,7567	3,0443
2000	26,688	9,3432	2,8563	7,2590	3,6765	7,3713	3,6204
2500	50,125	16,9860	2,9509	11,9957	4,1785	11,6530	4,3014
3000	85,485	28,4948	3,0000	19,1255	4,4696	17,6864	4,8333

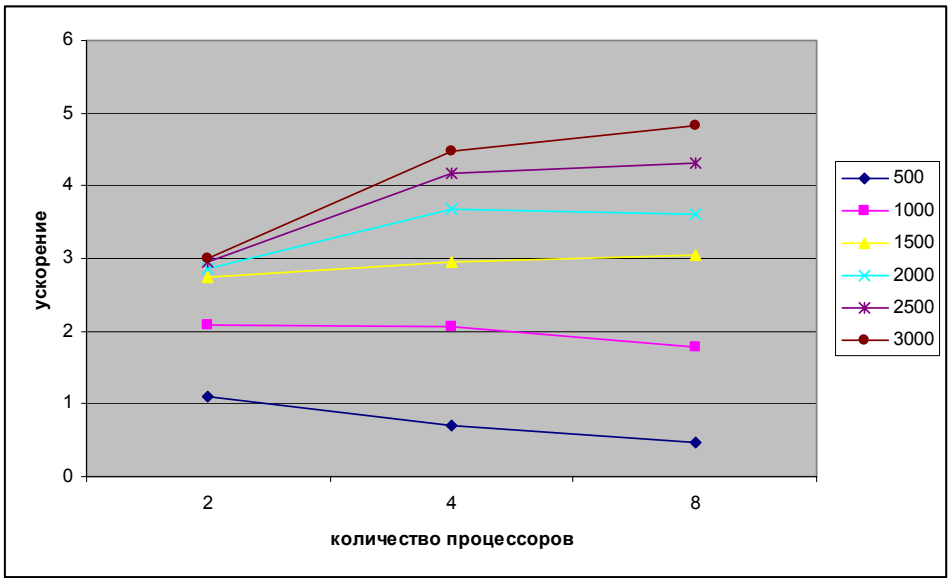

Рис. 9.3. Зависимость ускорения от количества процессоров при выполнении параллельного алгоритма Гаусса для разных размеров систем линейных уравнений

Сравнение времени выполнения эксперимента T_p^* и теоретической оценки T_p из (9.5) приведено в таблице 9.2 и на рис. 9.4.

Таблица 9.2. Сравнение экспериментального и теоретического времени выполнения параллельного алгоритма Гаусса

Размер системы	2 процессора		4 процессора		8 процессоров	
	T_p	T_p^*	T_p	T_p^*	T_p	T_p^*
500	0,4683	0,3302	0,7400	0,5170	1,0444	0,7504
1000	1,7959	1,5950	2,0236	1,6152	2,5128	1,8715
1500	4,7631	4,1788	4,2404	3,8802	4,5987	3,7567
2000	10,1511	9,3432	7,7808	7,2590	7,4973	7,3713
2500	18,7411	16,9860	13,0353	11,9957	11,4040	11,6530
3000	31,3142	28,4948	20,3946	19,1255	16,5140	17,6864

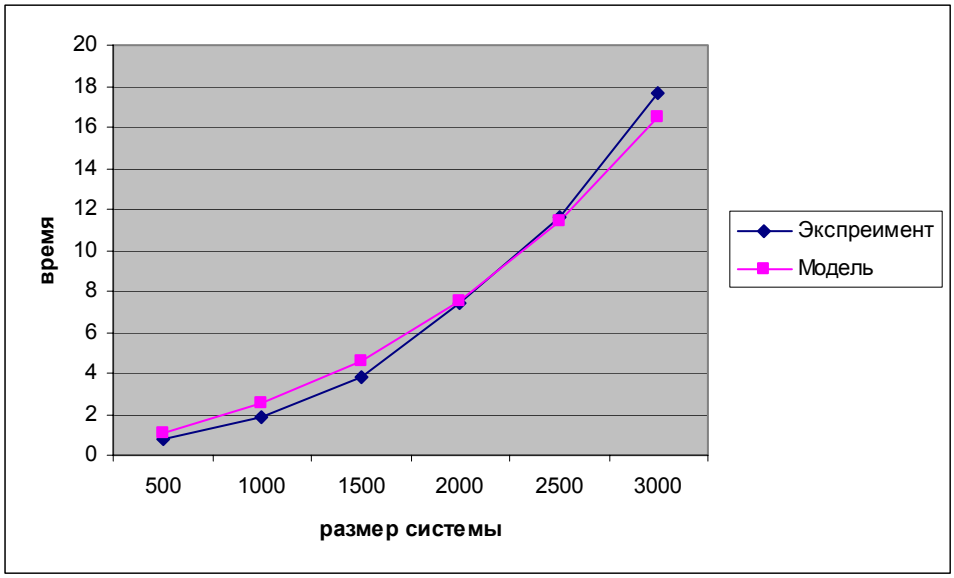

Рис. 9.4. График зависимости экспериментального и теоретического времени проведения эксперимента на восьми процессорах от объема исходных данных

9.3. Метод сопряженных градиентов

Рассмотрим теперь совершенно иной подход к решению систем линейных уравнений, при котором к искомому точному решению $x*$ системы $Ax=b$ строится последовательность приближенных решений x^0, x^1,…, x^k,… При этом процесс вычислений организуется таким способом, что каждое очередное приближение дает оценку точного решения со все уменьшающейся погрешностью, и при продолжении расчетов оценка точного решения может быть получена с любой требуемой точностью. Подобные *итерационные методы* решения систем линейных уравнений широко используются в практике вычислений. К преимуществам итерационных методов можно отнести меньший объем (по сравнению, например, с методом Гаусса) необходимых вычислений для решения разреженных систем линейных уравнений, возможность более быстрого получения начальных приближений искомого решения, наличие эффективных способов организации параллельных вычислений. Дополнительная информация с описанием таких методов, рассмотрение вопросов сходимости и точности получаемых решений может быть получена, например, в Bertsekas and Tsitsiklis (1989), Бахвалов и др. (1987).

Одним из наиболее известных итерационных алгоритмов является *метод сопряженных градиентов*, который может быть применен для решения симметричной положительно определенной системы линейных уравнений большой размерности.

Напомним, что матрица A является $симметричной$, если она совпадает со своей транспонированной матрицей, т.е. $A=A^T$. Матрица A называется $положительно\ определенной$, если для любого вектора x справедливо: $x^T Ax > 0$.

Известно (см., например, Bertsekas and Tsitsiklis (1989), Бахвалов и др. (1987)), что после выполнения n итераций алгоритма (n есть порядок решаемой системы линейных уравнений), очередное приближение x^n совпадает с точным решением.

9.3.1. Последовательный алгоритм

Если матрица A симметричная и положительно определенная, то функция

$$q(x)=\frac{1}{2}x^T\cdot A\cdot x-x^T b+c \qquad (9.6)$$

имеет единственный минимум, который достигается в точке $x*$, совпадающей с решением системы линейных уравнений (9.2). *Метод сопряженных градиентов* является одним из широкого класса итерационных алгоритмов, которые позволяют найти решение (9.2) путем минимизации функции $q(x)$.

Итерация метода сопряженных градиентов состоит в вычислении очередного приближения к точному решению в соответствии с правилом:

$$x^k = x^{k-1} + s^k d^k . \qquad (9.7)$$

Тем самым, новое значение приближения x^k вычисляется с учетом приближения, построенного на предыдущем шаге x^{k-1}, скалярного шага s^k и вектора направления d^k.

Перед выполнением первой итерации вектора x^0 и d^0 полагаются равными нулю, а для вектора g^0 устанавливается значение равное $-b$. Далее каждая итерация для вычисления очередного значения приближения x^k включает выполнение четырех шагов:

Шаг 1: Вычисление градиента:

$$g^k = A \cdot x^{k-1} - b; \tag{9.8}$$

Шаг 2: Вычисление вектора направления:

$$d^k = -g^k + \frac{\left((g^k)^T, g^k\right)}{\left((g^{k-1})^T, g^{k-1}\right)} d^{k-1}, \tag{9.9}$$

где (g^T,g) есть скалярное произведение векторов;

Шаг 3: Вычисление величины смещения по выбранному направлению:

$$s^k = \frac{\left(d^k, g^k\right)}{(d^k)^T \cdot A \cdot d^k}; \tag{9.10}$$

Шаг 4: Вычисление нового приближения:

$$x^k := x^{k-1} + s^k d^k. \tag{9.11}$$

Как можно заметить, данные выражения включают две операции умножения матрицы на вектор, четыре операции скалярного произведения и пять операций над векторами. Как результат, что общее количество числа операций, выполняемых на одной итерации, составляет

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

Как уже отмечалось ранее, для нахождения точного решения системы линейных уравнений с положительно определенной симметричной матрицей необходимо выполнить n итераций. Таким образом, для нахождения решения системы необходимо выполнить

$$T_1=2n^3+13n^2, \tag{9.12}$$

и, тем самым, сложность алгоритма имеет порядок $O(n^3)$.

Поясним выполнение метода сопряженных градиентов на примере решения системы линейных уравнений вида:

$$\begin{aligned} 3x_0 - x_1 &= 3 \\ -x_0 + 3x_1 &= 7 \end{aligned}.$$

Эта система уравнений второго порядка обладает симметричной положительно определенной матрицей, для нахождения точного решения этой системы достаточно провести всего две итерации метода.

На первой итерации было получено значение градиента $g^1=(-3, -7)$, значение вектора направления $d^1=(3, 7)$, значение величины смещения $s^1=0.439$. Соответственно, очередное приближение к точному решению системы $x^1=(1.318, 3.076)$.

На второй итерации было получено значение градиента $g^2=(-2.121, 0.909)$, значение вектора направления $d^2=(2.397, -0.266)$, а величина смещения – $s^2=0.284$. Очередное приближение совпадает с точным решением системы $x^2=(2, 3)$.

На рис. 9.3 представлена последовательность приближений к точному решению, построенная методом сопряженных градиентов (в качестве начального приближения x^0 выбрана точка (0,0)).

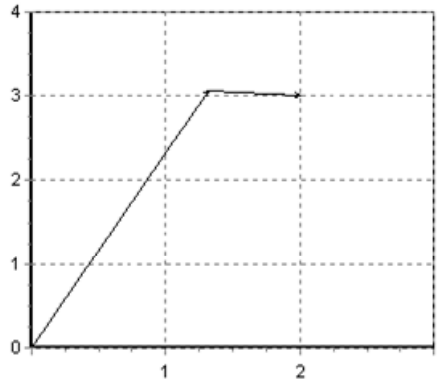

Рис. 9.5. Итерации метода сопряженных градиентов при решении системы второго порядка

9.3.2. 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 соотношений (9.8)-(9.11) показывает, что основные вычисления, выполняемые в соответствии с методом, состоят в умножении матрицы A на вектора x и d, и, как результат, при организации параллельных вычислений может быть полностью использован материал, изложенный в разделе 7.

Дополнительные вычисления в (9.8)-(9.11), 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. Общие же рекомендации могут состоять в следующем – при малом размере векторов можно применить дублирование векторов между процессорами, при большом порядке решаемой системы линейных уравнений более целесообразно осуществлять блочное разделение векторов.

9.3.3. Анализ эффективности

Выберем для дальнейшего анализа эффективности получаемых параллельных вычислений параллельный алгоритм матрично-векторного умножения при ленточном горизонтальном разделении матрицы и при полном дублировании всех обрабатываемых векторов.

Трудоемкость последовательного метода сопряженных градиентов была уже определена ранее в (9.12).

Определим время выполнения параллельной реализации метода сопряженных градиентов. Вычислительная сложность параллельных операций умножения матрицы на вектор при использовании схемы ленточного горизонтального разделения матрицы составляет:

$T_p^1(calc) = 2n\lceil n/p \rceil \cdot (2n-1)$ (см. раздел 7).

Как результат, при условии дублирования всех вычислений над векторами общая вычислительная сложность параллельного варианта метода сопряженных градиентов является равной:

$$T_p(calc) = n\left(2 \cdot \lceil n/p \rceil \cdot (2n-1) + 13n\right).$$

С учетом полученных оценок, показатели ускорения и эффективности параллельного алгоритма могут быть выражены при помощи соотношений:

$$S_p = \frac{2n^3 + 13n^2}{n\left(2\lceil n/p \rceil \cdot (2n-1) + 13n\right)}, \quad E_p = \frac{2n^3 + 13n^2}{p \cdot n\left(2\lceil n/p \rceil \cdot (2n-1) + 13n\right)}.$$

Рассмотрев построенные показатели, можно отметить, что балансировка вычислительной нагрузки между процессорами в целом является достаточно равномерной.

Уточним теперь приведенные выражения – учтем длительность выполняемых вычислительных операций и оценим трудоемкость операции передачи данных между процессорами. Как можно заметить, информационное взаимодействие процессоров возникает только при выполнении операций умножения матрицы на вектор. С учетом результатов раздела 7 коммуникационная сложность рассматриваемых параллельных вычислений является равной:

$$T_p(comm) = 2n\left(\alpha \cdot \lceil \log p \rceil + w(n/p)(p-1)/\beta\right),$$

где α – латентность, β – пропускная способность сети передачи данных, а w есть размер элемента упорядочиваемых данных в байтах.

В окончательном виде, время выполнения параллельного варианта метода сопряженных градиентов для решения систем линейных уравнений принимает вид:

$$T_p = n \cdot \left[\left(2\lceil n/p \rceil \cdot (2n-1) + 13n\right) \cdot \tau + 2\left(\alpha \cdot \lceil \log p \rceil + w(n/p)(p-1)/\beta\right)\right], \qquad (9.13)$$

где τ есть время выполнения базовой вычислительной операции.

9.3.4. Результаты вычислительных экспериментов

Вычислительные эксперименты для оценки эффективности параллельного варианта метода сопряженных градиентов для решения систем линейных уравнений проводились при условиях, указанных в п. 7.6.5.

Результаты вычислительных экспериментов приведены в таблице 9.3. Эксперименты проводились на вычислительных системах, состоящих из двух, четырех и восьми процессоров.

Таблица 9.3. Результаты вычислительных экспериментов для параллельного метода сопряженных градиентов для решения систем линейных уравнений

Размер системы	Последовательный алгоритм	Параллельный алгоритм					
		2 процессора		4 процессора		8 процессоров	
		Время	Ускорение	Время	Ускорение	Время	Ускорение
500	0,5	0,4634	1,0787	0,4706	1,0623	1,3020	0,3840
1000	8,14	3,9207	2,0761	3,6354	2,2390	3,5092	2,3195
1500	31,391	17,9505	1,7487	14,4102	2,1783	20,2001	1,5539
2000	92,36	51,3204	1,7996	40,7451	2,2667	37,9319	2,4348
2500	170,549	125,3005	1,3611	85,0761	2,0046	87,2626	1,9544
3000	363,476	223,3364	1,6274	146,1308	2,4873	134,1359	2,7097

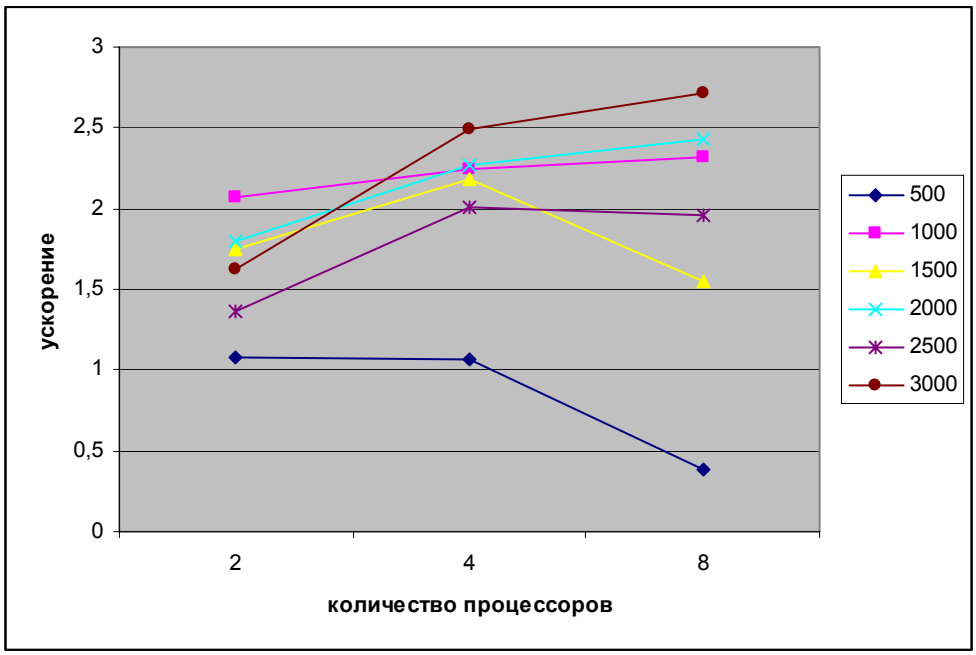

Рис. 9.6. Зависимость ускорения от количества процессоров при выполнении параллельного метода сопряженных градиентов для решения систем линейных уравнений

Сравнение времени выполнения эксперимента T_p^* и теоретической оценки T_p из (9.13) приведено в таблице 9.4 и на рис. 9.7.

Таблица 9.4. Сравнение экспериментального и теоретического времени выполнения параллельного метода сопряженных градиентов решения системы линейных уравнений

Размер системы	2 процессора		4 процессора		8 процессоров	
	T_p	T_p^*	T_p	T_p^*	T_p	T_p^*
500	1,3042	0,4634	0,6607	0,4706	0,3390	1,3020
1000	10,3713	3,9207	5,2194	3,6354	2,6435	3,5092
1500	34,9333	17,9505	17,5424	14,4102	8,8470	20,2001
2000	82,7220	51,3204	41,4954	40,7451	20,8822	37,9319
2500	161,4695	125,3005	80,9446	85,0761	40,6823	87,2626
3000	278,9077	223,3364	139,7560	146,1308	70,1801	134,1359

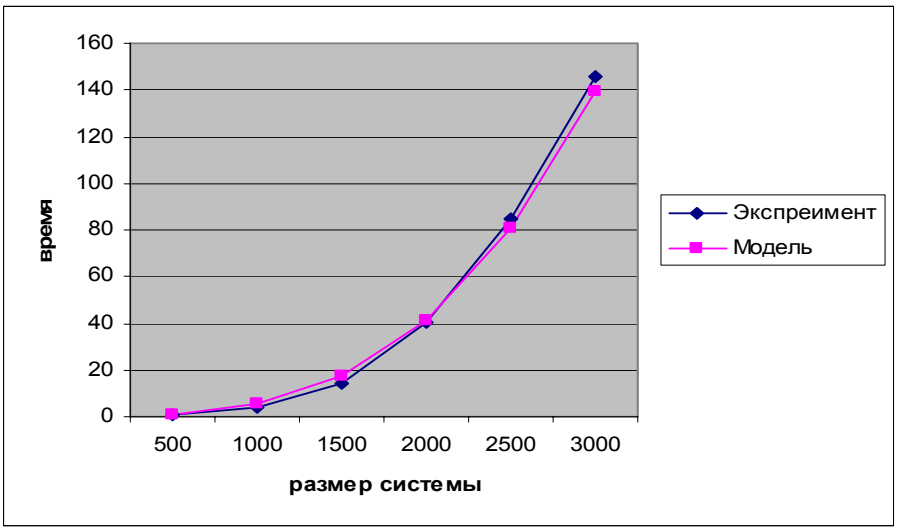

Рис. 9.7. График зависимости экспериментального и теоретического времени проведения эксперимента на четырех процессорах от объема исходных данных

9.4. Краткий обзор раздела

Данный раздел посвящен проблеме параллельных вычислений при решении систем линейных уравнений. Изложение учебного материала проводится с использованием двух широко известных алгоритма – *метод Гаусса* как пример прямого алгоритма решения задачи и итерационный *метод сопряженных градиентов*.

Параллельный вариант метода Гаусса (подраздел 9.2) основывается на ленточном разделении матрицы между процессорами с использованием циклической схемы распределения строк, что позволяет сбалансировать вычислительную нагрузку. Для определения параллельного варианта метода проводится полный цикл проектирования – определяются базовые подзадачи, выделяются информационные взаимодействия, обсуждаются вопросы масштабирования, выводятся оценки показателей эффективности, предлагается схема программной реализации и приводятся результаты вычислительных экспериментов. В целом, как показывает проведенный анализ эффективности, использование параллельного варианта метода Гаусса не позволяет получить ускорения вычислений (см. рис. 9.7) в силу большого числа выполняемых коммуникационных операций.

Важный момент при рассмотрении *параллельного варианта метода сопряженных градиентов* (подраздел 9.3) состоит в том, что способ параллельных вычислений для этого метода может быть получен через параллельные алгоритмы выполняемых вычислительных действий – операций умножения матрицы на вектор, скалярного произведения векторов, сложения и вычитания векторов. Выбранный для учебного изучения подход состоит в распараллеливании вычислительно наиболее трудоемкой операции умножения матрицы на вектор, в то время как все вычисления над векторами дублируются на каждом процессоре для уменьшения числа выполняемых операций передачи данных. Такой подход позволил организовать параллельные вычисления с достаточно высокими показателями эффективности см. рис. 9.7).

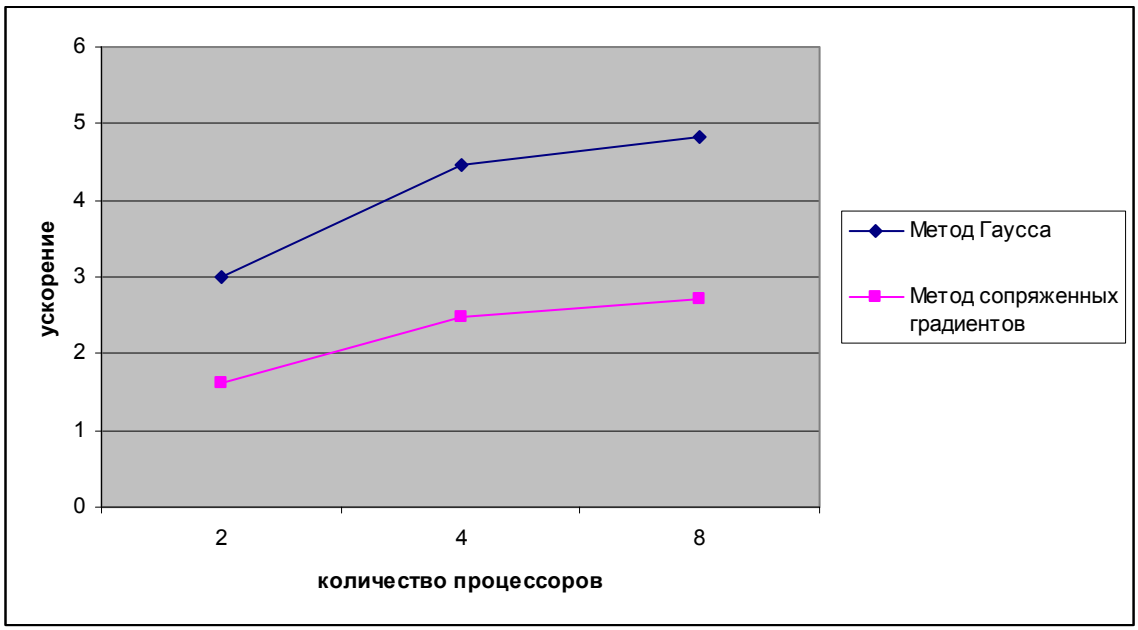

Рис. 9.8. Ускорение параллельных алгоритмов решения системы линейных уравнений с размером матрицы 3000×3000

9.5. Обзор литературы

Проблема численного решения систем линейных уравнений широко рассматривается в литературе. В учебного рассмотрения могут быть рекомендованы работы Березина и Жидкова (1966), Самарского и Гулина (1989), Бахвалова и др. (1987), Kumar (1994) и Quinn (2003). Широкое обсуждение вопросов параллельных вычислений для решения данного типа задач выполнено в работе Bertsekas and Tsitsiklis (1989), Dongarra, et al. (1999).

При рассмотрении вопросов программной реализации параллельных методов может быть рекомендована работа Blackford, et al. (1997).. В данной работе рассматривается хорошо известная и широко используемая в практике параллельных вычислений программная библиотека численных методов ScaLAPACK.

9.6. Контрольные вопросы

1. Что представляет собой система линейных уравнений? Какие типы систем вам известны? Какие методы могут быть использованы для решения систем разных типов?

2. В чем состоит постановка задачи решения системы линейных уравнений?

3. В чем идея параллельной реализации метода Гаусса?

4. Какие информационные взаимодействия имеются между базовыми подзадачами для параллельного варианта метода Гаусса?

5. Какие показатели эффективности для параллельного варианта метода Гаусса?

6. В чем состоит схема программной реализации параллельного варианта метода Гаусса?

7. В чем состоит идея параллельной реализации метода сопряженных градиентов?

8. Какой из алгоритмов обладает большей коммуникационной сложностью?

9.7. Задачи и упражнения

1. Выполните анализ эффективности параллельных вычислений в отдельности для прямого и обратного этапов метода Гаусса. Оцените, на каком этапе происходит большее снижение показателей.

2. Выполните разработку параллельного варианта метода Гаусса при вертикальном разбиении матрицы по столбцам. Постройте теоретические оценки времени работы этого алгоритма с учетом параметров используемой вычислительной системы. Проведите вычислительные эксперименты. Сравните результаты выполненных экспериментов с ранее полученными теоретическими оценками.

3. Выполните реализацию параллельного метода сопряженных градиентов. Постройте теоретические оценки времени работы этого алгоритма с учетом параметров используемой вычислительной системы. Проведите вычислительные эксперименты. Сравните результаты выполненных экспериментов с ранее полученными теоретическими оценками.

2. Выполните разработку параллельных вариантов методов Якоби и Зейделя решения систем линейных уравнений (см. например Kumar (1994)). Постройте теоретические оценки времени работы этого алгоритма с учетом параметров используемой вычислительной системы. Проведите вычислительные эксперименты. Сравните результаты выполненных экспериментов с ранее полученными теоретическими оценками.