ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ ЛИНЕЙНОЙ АЛГЕБРЫ

Вычислительные методы линейной алгебры включают в себя различные действия с матрицами и векторами (обращение матриц, вычисление собственных значений и собственных векторов матриц, умножение матриц и векторов, вычисление определителей и т.д.) и численные методы решения систем линейных алгебраических уравнений. В дальнейшем будем рассматриваются только системы линейных уравнений с квадратными невырожденными матрицами, имеющие единственное решение.

Некоторые сведения о матрицах

Система линейных алгебраических уравнений с квадратной матрицей записывается в виде

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, 2, \ldots, n .$$

Числа a_{ij} образуют таблицу, называемую матрицей, а сами эти числа называются элементами матрицы. Для обозначения матрицы используются заглавные буквы. Квадратные матрицы имеют структуру

$$A = \begin{pmatrix} a_{11}\, a_{12} \ldots a_{1n} \\ a_{21}\, a_{22} \ldots a_{2n} \\ \ldots\ldots\ldots\ldots \\ a_{n1}\, a_{n2} \ldots a_{nn} \end{pmatrix}.$$

Первый индекс у a_{ij} (общего элемента матрицы) соответствует номеру строки, второй – номеру столбца. Если x_j и b_i рассматривать как компоненты векторов $\vec{x}$ и $\vec{b}$ в конечномерном пространстве R_n, то система линейных уравнений запишется $A\vec{x} = \vec{b}$. Это означает, что вектор $\vec{x}$ под действием матрицы A преобразовался в вектор $\vec{b}$, т.е. A является оператором в линейном пространстве. Таким образом, любой оператор в линейном пространстве есть матрица и наоборот. Поэтому в линейном нормированном пространстве норму матрицы обычно определяют следующим образом: $\|A\vec{x}\| = \|\vec{b}\|$. Тогда $\|A\| = \underbrace{\sup}_{\vec{x}\neq 0} \dfrac{\|A\vec{x}\|}{\|\vec{x}\|}$ и называется согласованной нормой, т.е. $\|A\vec{x}\| \le \|A\| \cdot \|\vec{x}\|$.

Наиболее распространенными являются первая норма матрицы $\|A\|_1 = \underbrace{\max}_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|$ и вторая $\|A\|_2 \le \left(\sum_{i,j=1}^{n} a_{ij}^2 \right)^{1/2}$. Основные действия над матрицами:

1. A = B, если $a_{ij} = b_{ij}$; $A \pm B = \{a_{ij} \pm b_{ij}\}$; $\lambda A = \{\lambda a_{ij}\}$;

$A \cdot B = C = \{c_{ij}\} = \sum_{k=1}^{n} a_{ik} b_{kj}$. В общем случае $A \cdot B \neq B \cdot A$. Если же $A \cdot B = B \cdot A$, то говорят, что A и B
перестановочные (коммутативные).

2. Для любой матрицы $A = \{a_{ij}\}$ определена транспонированная к ней матрица $A^T = \{a_{ji}\}$. Имеет место равенство $(A \cdot B)^T = B^T \cdot A^T$.

3. Если $\{a_{ij}\} = \{a_{ji}\}$, то A - симметричная матрица. Очевидно, что для нее $A^T = A$. Чтобы из произвольной матрицы A получить симметричную, ее нужно умножить справа на транспонированную. Действительно:
$(A \cdot A^T)^T = (A^T)^T A^T = A \cdot A^T$, т.е. $B = A \cdot A^T$ – квадратная симметричная матрица.

4. Матрицей, обратной к A, называется матрица A^{-1}, удовлетворяющая условию $A \cdot A^{-1} = A^{-1}A = E$, где E – единичная матрица. Очевидно, что если $A\vec{x} = \vec{b}$, то $\vec{x} = A^{-1}\vec{b}$, т.е. для решения системы достаточно умножить ее слева на матрицу, обратную к A.

5. Если для матрицы A имеет место уравнение $A\vec{x} = \lambda\vec{x}$, то все возможные значения λ называются собственными числами матрицы A, а фундаментальные решения такой однородной системы уравнений при этих значениях λ называются собственными векторами $\vec{x}$ матрицы A. Значения λ находятся из условия существования нетривиального решения системы $(A - \lambda E)\vec{x} = 0$, т.е. из равенства нулю определителя $|A - \lambda E| = 0$, что приводит к алгебраическому уравнению n-го порядка, называемому характеристическим уравнением матрицы, решив которое находят все n собственных чисел матрицы A. Совокупность всех собственных чисел образует спектр матрицы A. При этом
$\|A\vec{x}\| = |\lambda| \cdot \|\vec{x}\|$, $\|A\| \geq \underbrace{\max}_{1 \leq k \leq n} |\lambda_k|$. Если матрица симметричная, то $\|A\|_2 = \underbrace{\max}_{1 \leq i \leq n} |\lambda_i|$,
$\left\|A^{-1}\right\|_2 = \dfrac{1}{\underbrace{\min |\lambda_i|}_{1 \leq i \leq n}}$. В случае, если $A > 0$, то есть матрица A положительно определенная, т.е. $(A\vec{x}, \vec{x}) > 0$, все $\lambda_i > 0$.

6. Мерой обусловленности матрицы называется число $\nu(A) = \|A\| \cdot \left\|A^{-1}\right\|$. Для симметричной матрицы $\nu(A) = \dfrac{\underbrace{\max |\lambda_i|}_{1 \leq i \leq n}}{\underbrace{\min |\lambda_i|}_{1 \leq i \leq n}}$. Если $\nu(A)$ велико, то матрица называется плохо обусловленной, если невелико - хорошо обусловленной. При плохой обусловленности матрицы A решение системы уравнений $A\vec{x} = \vec{b}$

затруднено в силу ее большой чувствительности к неустранимым ошибкам, которые могут содержать элементы матрицы и правые части системы, и даже к ошибкам округления.

Умножение матриц

Операция умножения двух матриц $A - m \times n$ и $B - n \times s$ определена только для матриц, левая из которых имеет столько столбцов, сколько строк имеет правая матрица. Результатом умножения является новая матрица $C = A \cdot B$, имеющая столько строк, сколько их имеет левая матрица и столько столбцов, сколько имеет правая. В терминах элементов матриц операция записывается следующим образом:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}, \; i = 1,\ldots,m; \; j = 1,\ldots,s$$

Методы решения этой задачи на параллельных ВС зависят от выбранной топологии вычислительной сети и от способа распределения матриц *A, B, C* по процессорам. Если сеть содержит *p* процессоров, то каждую из трех матриц можно распределить между ними одним из 3-х способов: на *p* горизонтальных полос, на *p* вертикальных полос, на сетку размера *p* на *p*. Рассмотрим сначала так называемый ленточный алгоритм. Учитывая формулу последовательного алгоритма, можно разбить матрицы *A* и C на *p* горизонтальных полос, матрицу *B* - на *p* вертикальных полос. Схема разбиения представлена на рис. 1.

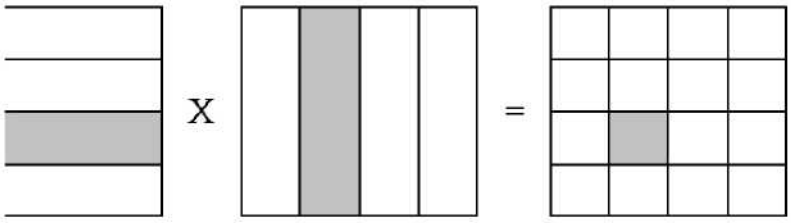

Рис.1. Ленточная схема умножения матриц

Сеть в этом случае будет иметь топологию «кольцо», при использовании которой предполагается, что горизонтальные полосы матрицы *A,* содержащие по *n/p* строчек, которые перед началом вычислений рассылаются в локальную память соответствующих процессоров и постоянно там находятся. Вертикальные полосы матрицы *B,* содержащие по *m/p* столбцов, рассылаются в локальную память соответствующих процессоров. В процессе вычислений каждый *i-й* процессор *(i* = 1, … , p) находит произведение элементов *i-й* горизонтальной полосы на элементы находящейся у него вертикальной полосы. В первом процессоре на этом же этапе в массиве *C* получается фрагмент, расположенный на одну полосу ниже и правее и т.д. На втором этапе вертикаль-

ные полосы массива *B* циклически сдвигаются на одну полосу, то есть перемещаются в локальную память соседних процессоров. После этого вычисляются элементы соответствующих фрагментов в массивах *C* каждого процессора. Затем снова выполняется сдвиг полос массива *B* и т.д. *p* раз. На последнем этапе производится сбор горизонтальных полос массива C из всех процессоров в корневой процессор.

Если матрицу *B* не разрезать на вертикальные полосы, то сдвиг полос не потребуется. Однако в этом случае понадобится в *p* раз больше локальной памяти для каждого процессора.

Другой прием для организации параллельных вычислений называется геометрическим способом распараллеливания. В данных методах матрицы *A, B, C* разбиваются на прямоугольные блоки, как показано в правой части рис. 1. Каждый из процессоров, которые можно объединить по топологии тора или иным удобным способом, отвечает за вычисление только своего прямоугольного фрагмента матрицы C. В процессе вычислений происходит обмен фрагментами в горизонтальном направлении для матрицы *A* и в вертикальном направлении для матрицы *B*.

Вычислительную трудоемкость рассмотренного блочного метода можно оценить как $T_h = 2n^3 / p$.

Представление матриц в виде решетки, с одной стороны, увеличивает объем информации, пересылаемой между процессорами, однако, с другой стороны, обмен может быть совмещен с обработкой данных. Ленточный метод умножения позволяет снизить объем пересылаемых данных. Но разница в объемах пересылаемых данных уменьшается при увеличении числа процессоров в соответствии с выражением $(1+1/\sqrt{p})$. Кроме того, использование геометрического способа распараллеливания позволяет помещать прямоугольные блоки данных в процессорах с учетом близости их физического расположения. Это также позволяет ускорить пересылку данных. Разбиение матриц по принципу прямоугольной решетки дает возможность строить эффективные алгоритмы для обработки разреженных матриц.

Решение систем линейных алгебраических уравнений

Методы решения систем линейных алгебраических уравнений разделяют на два больших класса методов: прямые методы (их часто называют также точными) и итерационные (приближенные).

Прямые методы позволяют получить решение за конечное число арифметических операций (метод Крамера, метода Гаусса и его модификации и т.д.). Однако число операций в этих методах существенно зависит от размерности системы уравнений, причем эта зависимость нелинейная. Так, метод Крамера требует порядка $n \cdot n!$ арифметических операций и по этой причине уже при n =

30 неприменим, метод Гаусса требует $\approx n^3$ операций, что также ограничивает область его применения.

Итерационные методы позволяют получить решение лишь с заданной точностью, но у них число операций менее жестко связано с размерностью системы уравнений. В настоящее время прямые метода применяют для решения систем до $n \approx 10^4$, итерационные - до $n \approx 10^7$.

3.1.Прямые методы

Наиболее распространенным среди прямых методов является метод Гаусса, основанный на элементарных преобразованиях расширенной матрицы и последовательном исключении неизвестных из системы уравнений. Напомним, что под элементарными преобразованиями понимаются следующие операции со строками (или столбцами) матрицы: 1) перестановка строк; 2) умножение любой строки на число, отличное от нуля; 3) сложение строк.

3.1.1. Метод Гаусса

Пусть в системе уравнений

$$\begin{aligned}
a_{11}^{(0)}x_1 + a_{12}^{(0)}x_2 + \ldots, + a_{1n}^{(0)}x_n &= a_{1,n+1}^{(0)} \\
a_{21}^{(0)}x_1 + a_{22}^{(0)}x_2 + \ldots, + a_{2n}^{(0)}x_n &= a_{2,n+1}^{(0)} \\
\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots & \\
a_{n1}^{(0)}x_1 + a_{n2}^{(0)}x_2 + \ldots, + a_{nn}^{(0)}x_n &= a_{n,n+1}^{(0)}
\end{aligned}$$

первый элемент $a_{11}^{(0)} \neq 0$ и назовем его ведущим элементом первой строки. Поделим все элементы этой строки на $a_{11}^{(0)}$:

$$\begin{aligned}
x_1 + a_{12}^{(1)}x_2 + \ldots, + a_{1n}^{(1)}x_n &= a_{1,n+1}^{(1)} \qquad \times(-a_{21}^{(0)}) \\
a_{21}^{(0)}x_1 + a_{22}^{(0)}x_2 + \ldots, + a_{2n}^{(0)}x_n &= a_{2,n+1}^{(0)} \qquad +
\end{aligned}$$

Затем умножим всю строку на $\times(-a_{21}^{(0)})$ и сложим с элементами второй строки. Получим:

$$\begin{aligned}
x_1 + a_{12}^{(1)}x_2 + \ldots, + a_{1n}^{(1)}x_n &= a_{1,n+1}^{(1)} \\
0 \; + a_{22}^{(1)}x_2 + \ldots, + a_{2n}^{(1)}x_n &= a_{2,n+1}^{(1)}
\end{aligned}$$

Продолжая этот процесс для всех остальных строк, получим систему уравнений, эквивалентную исходной:

$$
\begin{aligned}
&x_1 + a_{12}^{(1)} x_2 + \ldots, + a_{1n}^{(1)} x_n = a_{1,n+1}^{(1)} \\
&0 \; + a_{22}^{(1)} x_2 + \ldots, + a_{2n}^{(1)} x_n = a_{2,n+1}^{(1)} \\
&\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\
&0 \; + a_{n2}^{(1)} x_2 + \ldots, + a_{nn}^{(1)} x_n = a_{n,n+1}^{(1)}.
\end{aligned}
$$

Если ведущий элемент второй строки $a_{22}^{(1)} \neq 0$, то, продолжая аналогичное исключение, начиная со второй строки, приходим к системе уравнений с верхней треугольной матрицей (прямой ход метода Гаусса)

$$
\begin{aligned}
&x_1 + a_{12}^{(1)} x_2 + \ldots, \quad + a_{1n}^{(1)} x_n = a_{1,n+1}^{(1)} \\
&0 \; + \quad x_2 + \ldots, \quad + a_{2n}^{(2)} x_n = a_{2,n+1}^{(2)} \\
&0 \; + \quad 0 \; + x_3 + \ldots, + a_{3n}^{(3)} x_n = a_{3,n+1}^{(3)} \\
&\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\
&0 \; + \quad 0 + \ldots \quad \ldots, \quad + \; x_n = a_{n,n+1}^{(n)} = X_n .
\end{aligned}
$$

Если ведущий элемент какой-нибудь строки равен нулю, то для невырожденной матрицы путем перестановки ее строк всегда ведущий элемент можно сделать ненулевым. Далее, аналогичным образом организуется обратный ход метода: последняя строка умножается на $-a_{n-1,n}^{(n-1)}$ и складывается с предпоследней.

$$
\begin{aligned}
&0 \; + \quad 0 + \ldots \quad + x_{n-1} \; + \; 0 = X_{n-1} \\
&0 \; + \quad 0 + \ldots \quad \ldots, \quad + \; x_n = X_n
\end{aligned}
$$

Продолжая этот процесс, получим систему с диагональной матрицей:

$$
\begin{aligned}
&x_1 + \quad 0 + \ldots, \quad + \quad 0 = X_1 \\
&0 \; + \quad x_2 + \ldots, \quad + \quad 0 = X_2 \\
&0 \; + \quad 0 \; + x_3 + \ldots, + 0 = X_3 \\
&\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\
&0 \; + \quad 0 + \ldots \quad \ldots, + \; x_n = X_n .
\end{aligned}
$$

В результате на месте правого вектора-столбца будет находиться вектор-столбец искомого решения.

3.1.2. Параллельный алгоритм метода Гаусса

Рассмотрим возможный алгоритм распараллеливания метода Гаусса на N процессорах. Расширенная матрица

$$\overline{A} = \begin{pmatrix} a_{11}\ a_{12} \ldots a_{1n}\ a_{1,n+1} \\ a_{21}\ a_{22} \ldots a_{21}\ a_{2,n+1} \\ \ldots \quad \ldots \quad \ldots \quad \ldots \\ a_{n1}\ a_{n2} \ldots a_{nn}\ a_{n.n+1} \end{pmatrix}$$

системы уравнений разрезается на N полос приблизительно равной ширины по числу выбранных процессоров, каждая из которых загружается в свой процессор. Далее в нулевом (активном) процессоре первая (ведущая) строка делится на первый (ведущий) элемент и она передается всем остальным процессорам. Во всех процессорах с помощью этой строки элементарными преобразованиями все элементы первого столбца матрицы $\overline{A}$ (за исключением первой строки активного процессора) преобразуются в нулевые. Затем в активном процессоре ведущей становится следующая строка, ведущим – ее первый ненулевой элемент и процесс продолжается до исчерпания строк в активном процессоре. После этого активным становится следующий (первый) процессор и все повторяется снова до тех пор, пока не исчерпаются все ведущие строки во всех процессорах. В результате матрица A приведется к верхней треугольной матрице, распределенной по всем процессорам.

На этом прямой ход метода Гаусса заканчивается. Далее активным становится последний (N-1)- ый процессор и аналогичным образом организуется обратный ход, в результате которого на месте первых n столбцов расширенной матрицы $\overline{A}$ будут находиться элементы единичной матрицы, а на месте последнего столбца оказываются значения вектор- столбца искомого решения $\vec{x}$.

Вид матрицы *A* размера 16 × 16, получившейся после преобразования к треугольному виду, приведен на рис. 2 (для 4-х узлов). Символом **V** на рисунке обозначены вещественные числа.

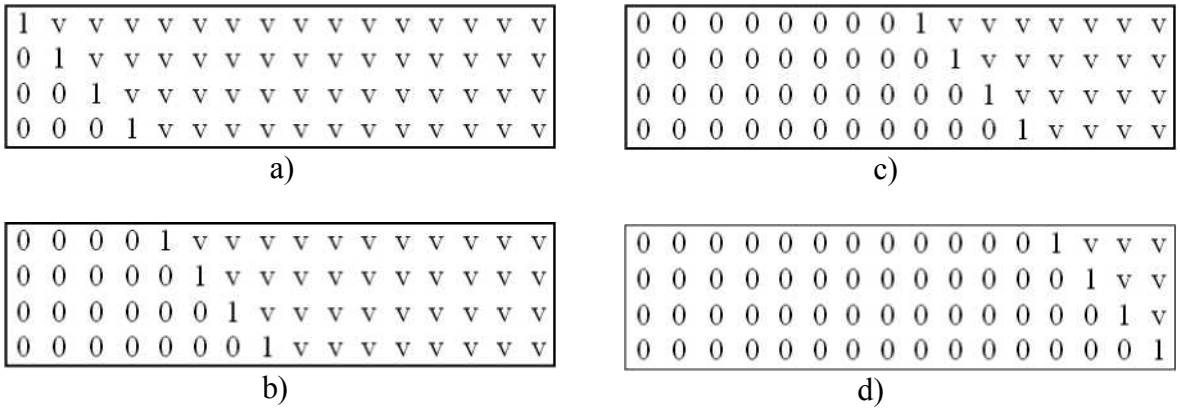

Рис.2. Прямой ход Гаусса (вариант 1)

Обратный ход метода Гаусса выполняется аналогично, но начинается с нижней строки последнего компьютера.

Недостатком рассмотренной модификации метода Гаусса является то, что по мере увеличения номера рассылаемой строки верхние относительно нее компьютеры при прямом ходе метода и нижние при обратном начинают простаивать, пока остальные не завершат обработку своих строк. То есть вычислительная работа между компьютерами оказывается распределенной неравномерно.

Во второй модификации метода Гаусса матрица A и вектор F распределяются между узлами следующим образом: 0-я строка заносится в 0-й компьютер, 1-я - в 1-й, ..., $(p-1)$-я - в $(p-1)$-й. Затем *p-я* трока снова располагается под 0-ой в 0-ом компьютере, $(p+1)$-я - в 1-ом и т.д.

Обработка строк при таком распределении выполняется следующим образом. Сначала текущей для рассылки и обработки является 0-я строка в 0-ом узле, затем 0-я в 1-ом узле и т.д. вплоть до 0-ой в $(p-1)$-м узле. Затем текущими являются первые строки последовательно из 0-го, 1-го, ... $(p-1)$-го узлов и т.д. Вид матрицы A после прямого хода метода для данной модификации представлен на рис. 3.

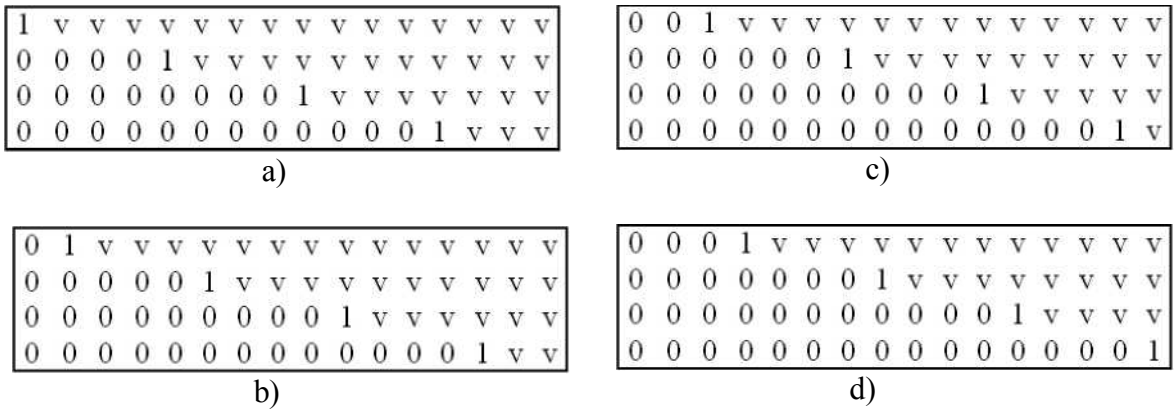

Рис.3. Схема прямого хода Гаусса (вариант 2)

Для обратного хода метода порядок выбора текущих строк выполняется аналогично, но в направлении снизу вверх.

Достоинством второй модификации метода Гаусса является то, что как при прямом, так и при обратном ходе метода процессоры более равномерно загружены вычислениями. Время простоя узла, разослaвшего свою строку остальным узлам, во второй модификации равно времени обработки одной строки, а не целой полосы.

При сравнении двух модификаций метода Гаусса следует учитывать, что преимущество второй модификации начинает сказываться лишь при сравнительно больших матрицах - не менее нескольких сотен тысяч элементов. Это объясняется тем, что время на рассылку строк во второй модификации пре-

вышает аналогичную величину первой модификации. Во втором случае на каждом шаге строка рассылается всем узлам кластера, а в первом случае - только оставшимся, число которых с каждым шагом уменьшается (от $p-1$ до нуля).

4.3. Вычисление определителей

Для вычисления определителей квадратных матриц можно использовать метод исключения (метод Гаусса), приводя матрицу к верхней треугольной с помощью элементарных преобразований подобно тому, как это было описано в п. 2.2.1. для прямого хода метода Гаусса. Однако есть принципиальное отличие – нельзя делить ведущую строку на ее главный элемент. Итак, пусть дана матрица A

$$A = \begin{pmatrix} a_{11}\, a_{12} \ldots a_{1n} \\ a_{21}\, a_{22} \ldots a_{2n} \\ \ldots\ldots\ldots\ldots \\ a_{n1}\, a_{n2} \ldots a_{nn} \end{pmatrix}$$

и нужно вычислить значение ее определителя. Пусть $a_{11} \neq 0$. Этого всегда можно добиться путем перестановки строк, учитывая при этом, что нечетное число таких перестановок меняет знак определителя. Далее, умножая последовательно первую строку на $-a_{11}/a_{i1}$ и складывая ее с i – той $(i = 1, 2, \ldots, n)$, получим преобразованную матрицу

$$A_1 = \begin{pmatrix} a_{11}\, a_{12} \ldots a_{1n} \\ 0 \quad a_{22}^{(1)} \ldots a_{2n}^{(1)} \\ \ldots\ldots\ldots\ldots \\ 0 \quad a_{n2}^{(1)} \ldots a_{nn}^{(1)} \end{pmatrix},$$

определитель которой равен определителю исходной матрицы, поскольку умножение любой строки на число и сложение ее с другой строкой не меняет значение определителя. Продолжая этот процесс дальше, приведем ее к верхней треугольной матрице

$$A_{n-1} = \begin{pmatrix} a_{11}\, a_{12} \ldots a_{1n} \\ 0 \quad a_{22}^{(1)} \ldots a_{2n}^{(1)} \\ \ldots\ldots\ldots\ldots \\ 0 \quad 0 \ \ldots \ a_{nn}^{(n-1)} \end{pmatrix},$$

определитель которой равен произведению элементов ее главной строки: $\det(A) = (-1)^m a_{11} \cdot a_{22}^{(1)} \cdot \ldots \cdot a_{nn}^{(n-1)}$, где m – число перестановок строк в процессе приведения матрицы к верхней треугольной.

4.4. Вычисление ранга произвольной матрицы

Вычисление ранга матрицы основано на элементарных преобразованиях и последовательных исключениях (метод Гаусса), когда матрица приводится к так называемому «ступенчатому» виду, в ходе которого получающиеся нулевые строки вычеркиваются. При этом допускается перестановка строк, поскольку такие операции не меняют ранга матрицы.

Матрицу будем называть «ступенчатой», если она удовлетворяет следующим условиям:

1) первый элемент ее первой строки является ненулевым;
2) в каждой последующей ненулевой строке число нулевых элементов, предшествующих первому ненулевому, больше, чем у предыдущей строки.

После приведения матрицы к «ступенчатому» виду, ее ранг будет равен числу строк в матрице. Заметим, что структуры исходной и «ступенчатой» матриц различны, но их ранги одинаковы.

Примеры «ступенчатых» матриц:

$$A = \begin{pmatrix} 1 & 2 & 5 & 0 & 9 \\ 0 & 3 & 2 & 6 & 0 \\ 0 & 0 & 0 & 2 & 1 \end{pmatrix}, \quad \operatorname{rang}(A) = 3; \quad B = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 \end{pmatrix}, \quad \operatorname{rang}(B) = 2.$$

4.5. Вычисление обратной матрицы.

Для вычисления обратной матрицы можно также использовать метод Гаусса, обобщив его на решение матричных уравнений. Действительно, согласно определению, матрица A^{-1}, обратная к матрице A, должна удовлетворять следующему матричному уравнению $A \cdot A^{-1} = E$, если рассматривать элементы матрицы A^{-1} как неизвестные. Расширенная матрица этой системы размерности $n \times 2n$ имеет вид

$$\overline{A} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} & 1 & 0 & 0 & \ldots & 0 \\ a_{21} & a_{22} & \ldots & a_{2n} & 0 & 1 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & & & & \\ a_{n1} & a_{n2} & \ldots & a_{nn} & 0 & 0 & 0 & \ldots & 1 \end{pmatrix}$$

и после прямого и обратного ходов метода Гаусса, как это описано в п. 3.1.1, на месте элементов исходной матрицы будут находиться элементы единичной матрицы, а на месте элементов единичной матрицы – элементы обратной $A^{-1} = \{b_{ij}\}$:

$$\overline{A} = \begin{pmatrix} 1\ 0 \dots 0\ b_{11}\ b_{12} \dots b_{1n} \\ 0\ 1 \dots 0\ b_{21}\ b_{22} \dots b_{2n} \\ \dots\ \dots\ \dots\ \dots\ \dots \\ 0\ 0 \dots 1\ b_{n1}\ b_{n2} \dots b_{nn} \end{pmatrix}.$$

Метод прогонки

Этот метод применяется для решения систем линейных уравнений с матрицей специального вида - трех- или пятидиагональной. Такие системы уравнений обычно возникают при численном решении уравнений (обыкновенных или в частных производных) с помощью конечно-разностных методов. В случае трехдиагональной матрицы система уравнений имеет вид

$$\begin{cases} A_j x_{j-1} + C_j x_j + B_j x_{j+1} = F_j, \quad j = 1, 2, \dots, N-1 \\ x_0 = \alpha_0 x_1 + \beta_0, \quad x_N = \alpha_N x_{N-1} + \beta_N \end{cases}$$

или

$$\begin{pmatrix} 1 & -\alpha_0 & 0 \dots & & & & \\ A_1 & C_1 & B_1\ 0 \dots & & & & \\ 0 & 0 & 0\ A_2\ C_2\ B_2\ 0 \dots & & & 0 & \\ 0 & 0 & \dots & & A_{N-1}\ C_{N-1}\ B_{N-1} & & \\ 0 & 0 & \dots & & & -\alpha_{N-1} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} \beta_0 \\ F_1 \\ \dots \\ F_{N-1} \\ \beta_N \end{pmatrix}$$

Если при этом выполняются условия

$$|C_j| \geq |A_j| + |B_j| > 0, \quad |\alpha_0| < 1, \quad |\alpha_N| \leq 1,$$

то говорят, что матрица A этой системы имеет диагональное преобладание.

Подставляя $x_0 = \alpha_0 x_1 + \beta_0$ в первое уравнение, получим

$A_1(\alpha_0 x_1 + \beta_0) + C_1 x_1 + B_1 x_2 = F_1$ или $x_1 = \alpha_1 x_2 + \beta_1$, где

$$\alpha_1 = \frac{-B_1}{C_1 + A_1\alpha_0}, \quad \beta_1 = \frac{F_1 - A_1\beta_0}{C_1 + A_1\alpha_0}.$$

Найденное x_1 подставим в следующее уравнение и получим уравнение, связывающее x_2 и x_3, и т.д. Допустим, что уже найдено соотношение $x_{k-1} = \alpha_{k-1}x_k + \beta_{k-1}, \quad k < N-1$. Подставив его в k – тое уравнение, получим $A_k(\alpha_{k-1}x_k + \beta_{k-1}) + C_k x_k + B_k x_{k+1} = F_k$ или $x_k = \alpha_k x_{k+1} + \beta_k$, где

$$\alpha_k = \frac{-B_k}{C_k + A_k\alpha_{k-1}}, \quad \beta_k = \frac{F_k - A_k\beta_{k-1}}{C_k + A_k\alpha_{k-1}}, \; k = 1,2,\ldots,N-1 \qquad (1)$$

При $k = N-1$ имеем $x_{N-1} = \alpha_{N-1}x_N + \beta_{N-1}, \quad x_N = \alpha_N x_{N-1} + \beta_N$, откуда

$$x_N = \frac{\beta_N + \alpha_N\beta_{N-1}}{1 - \alpha_N\alpha_{N-1}}.$$

Затем по формуле

$$x_k = \alpha_k x_{k+1} + \beta_k, \quad k = N-1, N-2, \ldots, 0 \qquad (2)$$

в обратном порядке определяем все x_k. Вычисление α_k, β_k по (1) называют прямой прогонкой (перегоняется левое граничное условие), вычисление x_k по формуле (2) - обратной прогонкой. Покажем, что если выполняются условия диагонального преобладания, то все вычисления корректны, т.е. знаменатель в (1) не обращается в нуль.

При k = 1 выполнено условие $|\alpha_{k-1}| < 1$. Поскольку $|C_k| \geq |A_k| > 0$, получим $|C_k + A_k\alpha_{k-1}| \geq |C_k| - |A_k|\cdot|\alpha_{k-1}| > |C_k| - |A_k| \geq 0$, т.е. знаменатель в нуль не обращается.

Далее, $|\alpha_k| = \dfrac{|B_k|}{|C_k + A_k\alpha_{k-1}|} \leq \dfrac{|C_k| - |A_k|}{|C_k| - |A_k|\cdot|\alpha_{k-1}|} < 1$. Отсюда по индукции $|\alpha_k| < 1, \; (k = 1,2,\ldots,N-1)$, и знаменатель в (1.4) в нуль не обращается. Так как $|\alpha_N| \leq 1$ и по доказанному выше $|\alpha_{N-1}| < 1$, то $1 - \alpha_N\alpha_{N-1} > 0$ и знаменатель в (1) тоже не обращается в нуль.

Описанная выше прогонка называется монотонной, так как все вычисления проводятся при монотонно изменяющихся значениях индекса k. В этом смысле она является аналогом схемы Гаусса без выбора главного элемента. Заметим, что это было возможным лишь благодаря выполнению условий диагонального преобладания. Если эти условия не выполняются, тогда можно, аналогично схеме Гаусса, организовать выбор главного элемента, и тогда прогонка будет

немонотонной, но всегда корректной. Алгоритм такой немонотонной прогонки можно найти в книге А.А. Самарского, Е.В. Николаева "Методы решения сеточных уравнений".
Можно оценить трудоемкость метода прогонки. При выполнении прямого хода по формулам (1) потребуется 8(n-2)+2 операций. Для выполнения обратного хода по формулам (2) потребуется еще 2(n-1)+5 операций. Таким образом, общее число операций можно оценить величиной 10n+O(1), а время решения системы уравнений методом прогонки при больших n будет определяться как

$$T = 10n \cdot \tau,$$

где τ - время выполнения одной операции.

3.1.4. Метод встречных прогонок

Совершенно аналогично изложенному выше, организацию вычисления прогоночных коэффициентов можно начинать с использования правого граничного условия $x_N = \alpha_N x_{N-1} + \beta_N$, подставляя его в предыдущее $N-1$ – ое уравнение исходной системы и т.д. В этом случае выражения для прогоночных коэффициентов имеют вид

$$\begin{aligned} &\xi_k = \frac{-A_k}{C_k + B_k \xi_{k+1}}, \ \eta_k = \frac{F_k - B_k \eta_{k+1}}{C_k + B_k \xi_{k+1}}, \quad k = N-1, N-2, \ldots, 1, \\ &\xi_N = \beta_N, \ \eta_N = \alpha_N. \end{aligned} \tag{3}$$

т.е. их вычисление производится «справа - налево». Значение компонент вектора решения вычисляется «слева – направо»

$$\begin{aligned} &x_k = \xi_k x_{k-1} + \eta_k, \quad k = 1, 2, \ldots, N, \\ &x_0 = \frac{\beta_0 + \alpha_0 \eta_1}{1 - \alpha_0 \xi_1} \end{aligned} \tag{4}$$

Поэтому часто вычисление компонент решения по формуле (2) называют *правой прогонкой*, а по формуле (4) – *левой прогонкой*. Комбинируя эти варианты метода прогонки, можно построить алгоритм *встречной прогонки*, когда вычисления прогоночных коэффициентов одновременно начинаются с левой и правой границ и «встречаются» в некоторой внутренней точке $k = m$. Значение решения x_m в этой точке определяется из решения системы уравнений

$$\begin{aligned} &x_m = \xi_m x_{m-1} + \mu_m, \\ &x_{m-1} = \alpha_{m-1} x_m + \beta_{m-1}, \end{aligned}$$

в которой значения всех прогоночных коэффициентов уже вычислены. Тогда

$$x_m = \frac{\eta_m + \xi_m \beta_{m-1}}{1 - \xi_m \alpha_{m-1}}$$

и по формулам

$$x_k = \xi_k x_{k-1} + \eta_k, \quad k = m+1, m+2, \ldots, N,$$
$$x_k = \alpha_k x_{k+1} + \beta_k, \quad k = m-1, m-2, \ldots, 0,$$

вычисляются все остальные компоненты вектора решения.

Нетрудно видеть, что вычисления при $k < m$ и при $k > m$ независимы и могут быть выполнены на двух параллельных процессорах.

Трудоемкость метода встречных прогонок можно оценить как 5n+O(1). Поскольку вычисления в этом методе при прямом и обратном ходах производятся независимо, то теоретическое значение коэффициента ускорения вычислений равно двум.

Параллельная прогонка

Изложим один из вариантов алгоритма параллельной прогонки, предложенный в работе Коновалова-Яненко. Исходная трехдиагональная матрица разбивается на блоки, каждый из которых назначается отдельному процессу. В каждом блоке решение представляется в виде линейной комбинации трех компонент с неизвестными коэффициентами. Три компоненты решения вычисляются независимо в каждом блоке тремя прогонками, однако за счет того, что системы уравнений отличаются лишь правыми частями, объем вычислений удается существенно оптимизировать. Затем в отдельном процессе вычисляются коэффициенты линейной комбинации для каждого блока. В итоге, платой в удвоенное количество числа арифметических операций, мы получаем на системе с числом процессоров N_p параллельную версию прогонки с обменом данных между процессорами порядка $O(N_p)$.

Сущность распараллеливания данного алгоритма параллельной прогонки заключается в следующем. Имеется система $N+1$ линейных алгебраических уравнений с трехдиагональной матрицей

$$a_i x_{i-1} + c_i x_i + b_i x_{i+1} = f_i; \quad i = 1, \ldots, N-1;$$
$$x_0 = \mu_0 x_1 + \nu_0; \; x_N = \mu_N x_{N-1} + \nu_N, \qquad (5)$$

решение которой можно представить в виде одномерного вектора, состоящий из N+1 элементов $X = \{x_0, \ldots, x_N\}$. Разобьем этот вектор точками M на $M+1$ произвольных интервалов размером $N_M + 1$, каждому из которых назначим свой номер процессора. Значения решения системы (1) в точках разбиения обозначим $\{\alpha_j\}, j = 1, \ldots, M$. Тогда вектор X можно представить в виде

$$X=(x_0^1,\ldots,x_{N_1-1}^1,\alpha_1;\,x_0^P,\ldots,x_{N_P-1}^P,\alpha_P;\,x_0^S,\ldots,x_{N_S-1}^S,\alpha_S)=(X^1,\ldots,X^P,\ldots,X^S).$$

Здесь $S=M+1$ – число процессоров, P – индекс процессора, $\alpha_S=x_N$. В силу линейности системы (5) представим вектор X на каждом интервале в виде

$$X^P=\alpha_{P-1}L^P+\alpha_P R^P+Q^P\ , \tag{6}$$

где L^P, R^P, Q^P есть решения краевых задач

$$\begin{aligned}
&a_i L_{i-1}^P+c_i L_i^P+b_i L_{i+1}^P=0,\quad L_0^P=1,\quad L_{N_P}^P=0;\\
&a_i R_{i-1}^P+c_i R_i^P+b_i R_{i+1}^P=0,\quad R_0^P=0,\quad R_{N_P}^P=1;\\
&a_i Q_{i-1}^P+c_i Q_i^P+b_i Q_{i+1}^P=f_i,\quad Q_0^P=0,\quad Q_{N_P}^P=0;
\end{aligned} \tag{7}$$

Задачи (7) решаются методом прогонки независимо друг от друга и результаты их решения запоминаются в памяти ЭВМ. Затем на границах каждого из интервалов (в каждой из M точек разбиения) находим

$$X_1^{P+1}=\alpha_P L_1{}^{P+1}+\alpha_{P+1}R_1{}^{P+1}+Q_1{}^{P+1} \tag{8}$$

$$X_{N_P-1}^P=\alpha_{P-1}L^P{}_{N_P-1}+\alpha_P R_{N_P-1}^P+Q_{N_P-1}^P, \tag{9}$$

где нижние индексы обозначают компоненту соответствующего вектора в данной точке его интервала. Подставляя (6), (8) в (5), получим систему уравнений для определения $\{\alpha_j\}$.

$$a_k L_{N_P-1}^P\alpha_{j-1}+[c_k+a_k R_{N_P-1}^P+b_k L_1^{P+1}]\alpha_j+b_k R_1^{P+1}\alpha_{j+1}=f_k-a_k Q_{N_P-1}^P-b_k Q_1^{P+1},$$

$$j\in[1,\ldots,S-1],\ k=\sum_{P=1}^{j}N_P. \tag{10}$$

Краевые условия для (10) определяются из краевых условий системы (5) путем подстановки в них выражений (8), (9):

$$\alpha_0=\xi_0\alpha_1+\eta_0,\quad \alpha_S=\xi_S\alpha_{S-1}+\eta_S.$$

Граничные условия слева

$$\xi_0=\frac{\mu_0 R_1^1}{1-\mu_0 L_1^1}\ \text{ и }\ \eta_0=\frac{\mu_0 Q_1^1+\nu_0}{1-\mu_0 L_1^1}$$

Граничные условия справа, в случае, если это для (5) есть условие первого рода, т.е. $x_N=\gamma$, то $\alpha_S=\gamma$. Если граничное условие для (5) задано в виде соотношения $x_N=\mu_N x_{N-1}+\nu_N$, то значение α_S получается из решения системы уравнений

$$\begin{cases}
\alpha_{S-1}=\xi'_{S-1}\alpha_S+\eta'_{S-1},\\
\alpha_S=\dfrac{\mu_N L_{N_S-1}^S}{1-\mu_N R_{N_S-1}^S}\alpha_{S-1}+\dfrac{\nu_N+\mu_N Q_{N_S-1}^S}{1-\mu_N R_{N_S-1}^S},
\end{cases} \tag{11}$$

где ξ'_{S-1} и η'_{S-1} получены из прямого хода прогонки при решении системы (3.17) для начальных значений ξ_0 и η_0 из (3.18). Решение системы (11) дает следующее выражение для α_S

$$\alpha_S = \frac{\delta_S + \sigma_S \eta'_{S-1}}{1 - \sigma_S \xi'_{S-1}}, \quad \text{где: } \sigma_S = \frac{\mu_N L^S_{N_S-1}}{1 - \mu_N R^S_{N_S-1}}, \quad \delta_S = \frac{\nu_N + \mu_N Q^S_{N_S-1}}{1 - \mu_N R^S_{N_S-1}}.$$

Выполняемое число операций в этом алгоритме при расчете трех матриц в прямом и обратном ходе в два раза меньше трехкратного выполнения расчетов для одной матрицы за счет повторяемости элементов вычисления. Однако число операций для переопределения элементов матрицы может быть значительно больше числа операций для прогонки. Оценочное выражение для ускорения s параллельного алгоритма (закон Амдала) имеет следующий вид

$$S = \frac{N_M}{\frac{aN_M}{N_P} + b(N_P) + c}.$$

Здесь a – коэффициент, пропорциональный числу операций на один элемент сетки размером N_M; N_p – число используемых процессоров, $b(N_P)$ – функция, характеризующая задержки сети, зависящая от числа процессоров;
c – коэффициент, учитывающий неускоряемую часть программы.

Для тестов производительности данного параллельного метода прогонки вычислялось ускорение счета по сравнению с последовательной версией. Оказалось, что производительность алгоритма ограничивается во многом пропускной способностью контроллера памяти. По результатам измерения производительности были сделаны следующие выводы: 1). В системе, где на каждый процесс приходится один контроллер памяти наблюдалось близкое к линейному ускорение вплоть до 32 процессов, которое лежит в интервале 0.667 N_p и 0.556 N_p. 2). В системе, использующей процессоры с тремя уровнями кэш-памяти большого объема, наблюдалось значительное улучшение производительности по мере того, как при распараллеливании размер блока данных, выделенных каждому процессорному ядру приближался к размеру кэш-памяти.

Итерационные методы

Из всего множества итерационных методов решения систем линейных алгебраических уравнений рассмотрим только методы Якоби, Зейделя и метод простых итераций.

Метод Якоби

Пусть в системе из N уравнений

$$A\vec{x} = \vec{b} \qquad (1)$$

все диагональные элементы матрицы $a_{ii} \neq 0$. Тогда, поделив ее строки на a_{ii}, приведем ее к виду $\vec{x} = B\vec{x} + \vec{d}$, где $\vec{d} = \{b_i / a_{ii}\}^T$, $b_{ij} = \begin{cases} -a_{ij}/a_{ii}, i \neq j, \\ 0, i = j \end{cases}$.

Итерации в методе Якоби проводятся по формуле

$$\vec{x}^{n+1} = B\vec{x}^{n} + \vec{d}, \qquad (2)$$

или в покомпонентной записи

$$x_i^{n+1} = \sum_{j=1}^{i-1} b_{ij} x_j^n + \sum_{j=i+1}^{N} b_{ij} x_j^n + d_i, \quad n = 0,1,2,\ldots,$$

где $i = 1,2,\ldots,N$, n – номер итерации, $\vec{x}^0$ - произвольный вектор.

Теорема. *Для существования единственного решения системы* (1*) и сходимости метода Якоби необходимо выполнение условия:* $\|B\| < 1$. *При выполнении этого условия метод Якоби сходится со скоростью геометрической прогрессии при любом ненулевом векторе* $\vec{x}^0$

$$\left\|\vec{x}^{n+1} - \vec{x}_*\right\| = q^n \left\|\vec{x}^{n} - \vec{x}^{0}\right\|, \text{ где } q = \|B\|, \ \vec{x}_* \text{- решение системы (1).}$$

Итерации прекращаются при достижении заданной абсолютной $\left\|\vec{x}^{n+1} - \vec{x}^{n}\right\| \leq \varepsilon$ или относительной $\left\|\vec{x}^{n+1} - \vec{x}^{n}\right\| / \left\|\vec{x}^{n}\right\| \leq \varepsilon$ точности.

Алгоритм метода Якоби хорошо распараллеливается. Действительно, нетрудно видеть, что алгоритм (2) состоит из умножения матрицы на вектор и сложения полученного вектора с вектором правых частей.

Метод Зейделя

Отличается от метода Якоби только способом вычисления компонент вектора решения

$$x_i^{n+1} = \sum_{j=1}^{i-1} c_{ij} x_j^{n+1} + \sum_{j=i+1}^{N} c_{ij} x_j^n + d_i, \quad n = 0,1,2,\ldots, \qquad (3)$$

где $i = 1,2,\ldots,N$, n – номер итерации, $\vec{x}^0$ - произвольный ненулевой вектор.

Теорема. *Для существования единственного решения системы* (3*) и сходимости метода Зейделя достаточно выполнения хотя бы одного из двух условий:*

1) $\sum_{i \neq j} |a_{ij}| < |a_{ii}|, \quad i = 1, 2, \ldots, N,$

2) *Матрица A - симметричная и положительно определенная.*

Заметим, что первое из этих условий есть не что иное, как условия диагонального преобладания. Отличие метода Зейделя от метода Якоби заключается в том, что новые значения вектора вычисляются не только на основании значений предыдущей итерации, но и с использованием значений уже вычисленных на данной итерации. Вычисления компонент вектора решения в (3) являются рекуррентными и параллельный алгоритм, аналогичный параллельному алгоритму для метода Якоби, построить не удается. Однако можно поступить следующим образом.

Каждая компонента x_i^{n+1} вектора $\vec{x}^{n+1}$ последовательно для каждого значения индекса i рассчитывается по формуле (3) и распараллеливание здесь возможно только при вычислении сумм в правой части формулы (3). Здесь можно использовать алгоритмы сдваивания при вычислении произведений в суммах и при вычислении самих сумм. Такие алгоритмы требуются достаточно интенсивных информационных обменов и, по-видимому, наиболее оптимальным для реализации такого параллельного алгоритма является использование OpenMP.

Метод простых итераций

В этом методе система линейных алгебраических уравнений (1)

$$A\vec{x} = \vec{b}$$

приводится к виду

$$\vec{x} = B \cdot \vec{x} + \vec{c} \quad , \tag{4}$$

где B - некоторая матрица, и решение системы находится как предел последовательности

$$\vec{x}^{n+1} = B \cdot \vec{x}^{n} + \vec{c} \tag{5}$$

Теорема *(о достаточном условии сходимости метода простой итерации). Если* $\|B\| < 1$*, то система уравнений* (1) *имеет единственное решение и итерационный .процесс* (5) *сходится к решению со скоростью геометрической прогрессии.*

Доказательство. Для всякого решения (4) имеем

$\|\vec{x}\| \le \|B\| \cdot \|\vec{x}\| + \|\vec{c}\|, \quad \|\vec{x}\|(1 - \|B\|) \le \|\vec{c}\|$ или $\|\vec{x}\| \le \dfrac{\|\vec{c}\|}{(1 - \|B\|)}$. Отсюда вытекает единственность решения однородной системы $\vec{x} = B \cdot \vec{x}$, т.е. при $\vec{c} = 0$, а, следовательно, существование и единственность решения системы (4.4) при любой правой ее части.

Пусть $\vec{x}_*$ - решение системы (1). Обозначим через $\vec{r}^n = \vec{x}^n - \vec{x}_*$ вектор погрешности на n- ой итерации. Тогда, вычитая (5) из (4), получаем $\vec{r}^{n+1} = B \cdot \vec{r}^n$ или $\vec{r}^{n+1} = (B)^n \cdot \vec{r}^0$. Отсюда следует, что $\left\|\vec{r}^n\right\| \le \|B\|^n \cdot \left\|\vec{r}^0\right\| \to 0$, так как $\|B\| < 1$. Теорема доказана.

Оценим погрешность итераций. Из равенств (4) и (5) имеем

$$\vec{x}_* - \vec{x}^n = \vec{x}^{n+1} - \vec{x}^n + B(\vec{x}_* - \vec{x}^n);$$

$$\left\|\vec{x}_* - \vec{x}^n\right\| \le \left\|\vec{x}^{n+1} - \vec{x}^n\right\| + \|B\| \cdot \left\|\vec{x}_* - \vec{x}^n\right\| \text{ или } \left\|\vec{x}_* - \vec{x}^{n+1}\right\| \le \frac{\|B\|}{1-\|B\|}\left\|\vec{x}^{n+1} - \vec{x}^n\right\|.$$

Эта оценка широко используется на практике.

К виду (4) систему (1) можно привести различными способами, используя конкретную структуру матрицы A. В самом общем случае, когда о матрице A ничего не известно, поступают следующим образом. Умножают обе части уравнения $A\vec{x} = \vec{b}$ на транспонированную матрицу A^T и записывают систему в виде (4):

$$\vec{x} = (E - D)\vec{x} + \vec{c},$$

где $D = A^T A, \vec{c} = A^T\vec{b}$. При этом $B = E - D$ - симметричная положительно определенная матрица. Если $b_{ii} \ne 0$, то для ее решения применим метод Зейделя или метод простой итерации при $\|B\| < 1$. Заметим, что указанная симметризация матрицы ухудшает обусловленность системы уравнений.

Выполнение условий сходимости метода простой итерации для систем уравнений общего вида $A\vec{x} = \vec{b}$ обеспечивает следующий итерационный процесс:

$$\vec{x}^{n+1} = \vec{x}^n - \alpha(A\vec{x}^n - \vec{b}), \text{ или } \vec{x}^{n+1} = B\vec{x}^n + \vec{c},$$

где $B = E - \alpha A, \vec{c} = \alpha\vec{b}$.

Если $\lambda_1, \lambda_2, \ldots, \lambda_N$ - собственные значения матрицы A, то

$$\|B\|_2 = \underbrace{\max}_{i} |\lambda_i(B)| = \underbrace{\max}_{i} |1 - \alpha\lambda_i|.$$

Пусть все $\lambda_i > 0$. Это справедливо для симметричной положительно определенной матрицы, один из способов построения указан выше. Тогда для нахождения максимального собственного числа такой матрицы можно использовать следующий алгоритм, который приведем без доказательства. В декартовом

(ортонормированном) n – мерном базисе возьмем произвольный ненулевой вектор $\vec{x}_0 = (x_1^{(0)}, x_2^{(0)}, \ldots, x_2^{(0)})$, где $x_1^{(0)}, x_2^{(0)}, \ldots, x_2^{(0)}$ – произвольные числа и организуем следующий итерационный процесс. Вычислим $\vec{z}_0 = \dfrac{\vec{x}_0}{\|\vec{x}_0\|}$ и затем по рекуррентным соотношениям $\vec{x}_k = A\vec{z}_{k-1}, \quad \lambda_{max}^{(k)} = (\vec{x}_k, \vec{z}_{k-1}), \quad \vec{z}_k = \dfrac{\vec{x}_k}{\|\vec{x}_k\|}$, где $k = 1, 2, \ldots,$.

Условием прекращения итерация является выполнение условия $\dfrac{|\lambda_{max}^{(k+1)} - \lambda_{max}^{(k)}|}{\lambda_{max}^{(k)}} \le \varepsilon$, где ε – заданная относительная точность вычисления λ_{max}.

Для нахождения минимального собственного значения поступим следующим образом. По изложенному выше алгоритму находим максимальное по модулю собственное значение $\lambda_{max}(A)$. Затем для матрицы $B = A - \lambda_{max}(A)E$ находим тем же способом $\lambda_{max}(B)$. Тогда

$$\lambda_{min}(A) = \lambda_{max}(A) + \lambda_{max}(B).$$

Действительно, $\lambda_i(B) = \lambda_i(A) - \lambda_{max}(A) \le 0, \ i = 1, 2, \ldots, n$, поэтому $\lambda_{max}(B) = \lambda_{min}(A) - \lambda_{max}(A)$.

Значения λ_i обычно неизвестны, однако оценка их границ вида $0 < \mu \le \lambda_i \le M < \infty$ не составляет труда. В частности, можно положить $\mu = \min \lambda(A)$, а $M = \max \lambda(A)$, алгоритм вычисления которых для такой матрицы изложен выше. Скорость сходимости итерационного процесса можно характеризовать величиной $\rho(\alpha) = \underbrace{\max}_{\mu \le \lambda \le M} |1 - \alpha\lambda| = \|B\|_2$. Рассмотрим задачу минимизации $\rho(\alpha)$ за счет выбора α. При $0 < \alpha \le 1/M$ функция $\rho(\alpha)$ неотрицательна и монотонно убывает на отрезке $[\mu, M]$. При некотором $\alpha = \alpha_0$ $1 - \alpha_0\mu = -(1 - \alpha_0)M$, откуда $\alpha_0 = \dfrac{2}{\mu + M}, \quad \rho(\alpha_0) = \dfrac{M - \mu}{M + \mu}$. Поэтому итерационный процесс

$$\vec{x}^{n+1} = \vec{x}^n - \frac{2}{\mu + M}(A\vec{x}^n - \vec{b}), \qquad (6)$$

где A - симметричная, положительно определенная матрица, всегда сходится и является оптимальным. Скорость сходимости оценивается

$$\left\|\vec{x}^n - \vec{x}_*\right\|_2 \le \left(\frac{M - \mu}{M + \mu}\right)^n \left\|\vec{x}^0 - \vec{x}_*\right\|_2$$

Погрешность:

$$\left\|\vec{x}^{n+1}-\vec{x}_*\right\|_2 \le \frac{M-\mu}{2\mu}\left\|\vec{x}^{n+1}-\vec{x}^{n}\right\|_2 .$$

Если выбирать α переменным на каждом итерационном шаге, то есть $\alpha=\alpha_n$, можно существенно повысить скорость сходимости. Способов построения α_n существует довольно много. Наиболее распространенным является способ, предложенный Чебышевым (Чебышевское ускорение)

$$\vec{x}^{n}=\vec{x}^{n-1}-\alpha_n^k(A\vec{x}^{n-1}-\vec{b}),\quad n=1,2,\dots,k, \qquad (7)$$

$$\alpha_n^k=\frac{2}{\mu+M+(M-\mu)\cos\dfrac{\pi(2n-1)}{2k}}$$

Погрешность:

$$\left\|\vec{x}^{n}-\vec{x}_*\right\|_2 \le 2\left(\frac{\sqrt{M}-\sqrt{\mu}}{\sqrt{M}+\sqrt{\mu}}\right)^n\left\|\vec{x}^{0}-\vec{x}_*\right\|_2$$

Нетрудно видеть, что по сравнению с методом простой итерации (стационарный итерационный процесс) этот метод дает выигрыш в числе итераций примерно в $\sqrt{M/\mu}$ раз, что может быть весьма существенным.

Распараллеливание алгоритма простой итерации достаточно очевидно – поскольку в (5) распараллеливается умножение матрицы на вектор и сложение двух получившихся векторов.

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 *n-й* итерации — порядка $O(n^2)$ операций - состоят в умножении матрицы *A* на вектор текущего приближения $\vec{x}^{(n+1)}$. 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, которые рассмотрим ниже.

Метод верхней релаксации

Рассмотрим систему (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

$$A=L+D+R. \tag{8}$$

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

$$\frac{(D + \omega L)(\vec{x}^{(n+1)} - \vec{x}^{(n)})}{\omega} + A\vec{x}^{(n)} = \vec{b}$$

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

Необходимым условием сходимости метода релаксации с любого начального приближения $\vec{x}^{(0)}$ к точному решению задачи $\vec{x}^{*}$ 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 $\omega = 1$ метод релаксации будет совпадать с описанным ранее *методом Зейделя.*

Скорость сходимости метода верхней релаксации определяется выбором параметра ω. Известно, 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 диффе-

ренциальных 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 является выражение

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

где ρ — спектральный радиус матрицы, а *R, D, L* — из (8). Возникающий при этом вопрос об оценке спектрального радиуса может быть решен численно, например, при помощи степенного метода.

Также следует отметить, что для ряда задач, возникающих при решении задач математической физики методом сеток, оптимальные параметры метода верхней релаксации могут быть найдены аналитически.

Последовательный алгоритм

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

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

С учетом того, что *A-L=R+D,* получаем

$$D\vec{x}^{(n+1)} = -\omega L\vec{x}^{(n+1)} + (1 - \omega)D\vec{x}^{(n)} - \omega R\vec{x}^{(n)} + \omega\vec{b}$$

Далее нетрудно записать явные формулы для отыскания компонент нового вектора $\vec{x}^{(n+1)}$

$$a_{ii}x_i^{(n+1)} = -\omega\sum_{j=1}^{i-1} a_{il}x_j + (1 - \omega)a_{ii}x_i^{(n)} - \omega\sum_{j=1+1}^{N} a_{il}x_j + \omega\vec{b}$$

Как следует из формулы (3), при подсчете *i*-й компоненты нового приближения все компоненты, индекс которых меньше *i,* берутся из нового приближения $\vec{x}^{(n+1)}$, а все компоненты, индекс которых больше либо равен *i* — из старого приближения $\vec{x}^{(n)}$. 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 $\vec{x}^{(n+1)}$ компоненты с индексом, меньшим или равным *i,* будут использо-

ваться «в новой версии». В силу этого обстоятельства для реализации метода достаточно хранить только одно (текущее) приближение $\vec{x}^{(n)}$, а при расчете следующего приближения $\vec{x}^{(n+1)}$ использовать формулу (3) для всех компонент по порядку и постепенно обновлять вектор $\vec{x}^{(n)}$.

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 *n-й* итерации - порядка $O(n^2)$ операций - заключаются в операциях матричного умножения $L\vec{x}^{(n+1)}$ и $R\vec{x}^{(n+1)}$. 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 $\vec{x}$ 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 (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 $\left\|\vec{x}^{n+1}-\vec{x}^{n}\right\|\le\varepsilon$ с параметром $\varepsilon=10^{-6}$, а итерационный параметр ω=1.1. Во всех экспериментах метод нашел решение с требуемой точностью за 11 итераций. Как и для предыдущих экспериментов, ускорение фиксировалось по сравнению с параллельной программой, запущенной в один поток.

Табл. 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

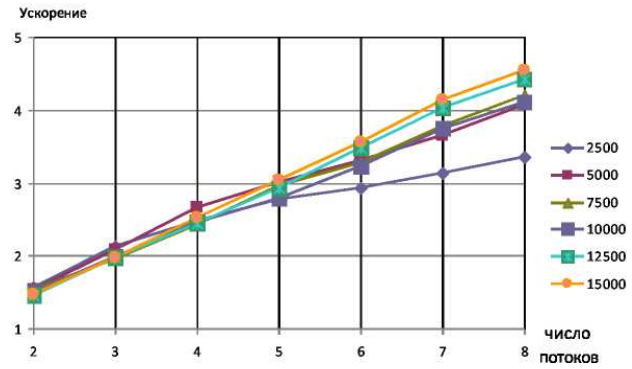

Рис. 4. Ускорение параллельного метода верхней релаксации

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

Метод сопряженных градиентов

Рассмотрим систему линейных уравнений (1) с симметричной, положительно определенной матрицей A размера $n \times n$. Основой *метода сопряженных градиентов* является следующее свойство: решение системы линейных уравнений (1) с симметричной положительно определенной матрицей A эквивалентно решению задачи минимизации функции

$$F(\vec{x}) = \frac{1}{2}(A\vec{x},\vec{x}) - (\vec{b},\vec{x})$$

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

$$\nabla F(\vec{x}) = A\vec{x} - \vec{b} \qquad (9)$$

обращается в ноль. Таким образом, решение системы (1) можно искать как решение задачи безусловной минимизации (9).

Последовательный алгоритм

С целью решения задачи минимизации (9) организуется следующий итерационный процесс.

Подготовительный шаг (s=0) определяется формулами

$$\vec{x}^{(1)} = \vec{x}^{(0)} + \alpha_0 \vec{h}^{(0)}, \quad \vec{r}^{(0)} = \vec{h}^{(0)} = \vec{b} - A\vec{x}^{(0)}$$

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

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

Основные шаги (n=1, 2,..., N—1) определяются формулами

$$\alpha_n = \frac{(\vec{r}^{(n)}, \vec{r}^{(n)})}{(A\vec{h}^{(n)}, \vec{h}^{(n)})}, \quad \vec{r}^{(n+1)} = \vec{r}^{(n)} - \alpha_n A\vec{h}^{(n)},$$

$$\beta_n = \frac{(\vec{r}^{(n+1)}, \vec{r}^{(n+1)})}{(\vec{r}^{(n)}, \vec{r}^{(n)})}, \quad \vec{h}^{(n+1)} = \vec{r}^{(n+1)} + \beta_n \vec{h}^{(n)},$$

$$\vec{x}^{(n+1)} = \vec{x}^{(n)} + \alpha_n \vec{h}^{(n)}$$

Здесь $\vec{r}^{(n)} = \vec{b} - A\vec{x}^{(n)}$ - невязка *n*-го приближения, коэффициент β_n находят из условия сопряженности направлений $\vec{h}^{(n)}$ и $\vec{h}^{(n-1)}$

$$(A\vec{h}^{(n)}, \vec{h}^{(n-1)}) = 0,$$

а α_n является решением задачи минимизации функции *F* по направлению $\vec{h}^{(n)}$

$$F(\vec{x}^{(n)} - \alpha_n \vec{h}^{(n)}) \to \min$$

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 $A\vec{h}^{(n)}$ 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

$$T_1 = L(2n^2 + 13n) \qquad (10)$$

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 при выполнении условия малости относительной нормы невязки

$$\left\|\vec{r}^{(n)}\right\| \Big/ \left\|\vec{b}\right\| \le \varepsilon \qquad (11)$$

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-той 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 (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

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

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

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

где L - число итераций метода.

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 (11) с параметром $\varepsilon = 10^{-6}$.
Результаты вычислительных экспериментов приведены в таблице 2 (время работы алгоритмов указано в секундах).

Табл. 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

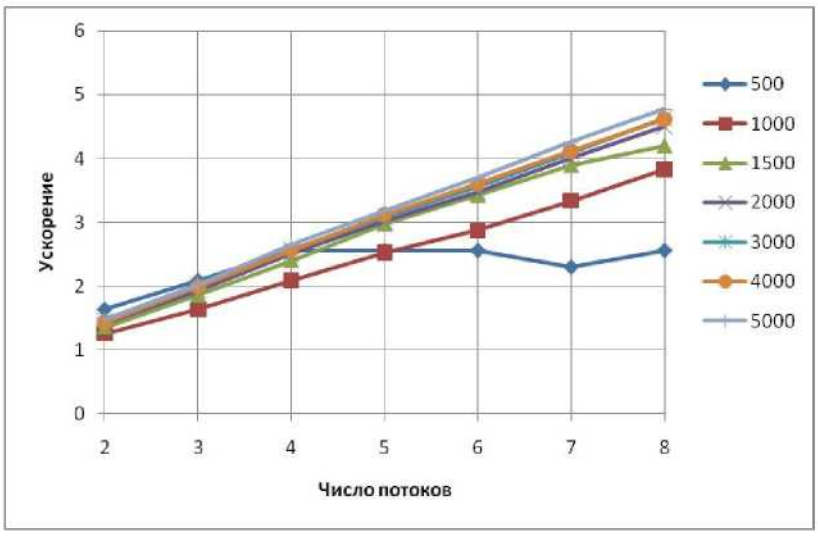

Рис. 5. Ускорение параллельного метода сопряженных градиентов

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. Использовать более чем 4 потока для решения данной задачи при $N<500$ - нецелесообразно.