НАУЧНОЕ ИЗДАНИЕ МГТУ ИМ. Н. Э. БАУМАНА
НАУКА и ОБРАЗОВАНИЕ
Эл № ФС77 - 48211. Государственная регистрация №0421200025. ISSN 1994-0408
электронный научно-технический журнал

Паралеллный алгоритм решения систем линейных алгебраических уравнений с многодиагональной матрицей коэффициентов

 # 07, июль 2013
DOI: 10.7463/0713.0590785

Желдаков А. В., Федорук В. Г.

УДК 004.942, 519.876.5

Россия, МГТУ им. Н.Э. Баумана
www.streeter@mail.ru
fedoruk.bmstu@mail.ru

Введение

При решении задач математической физики численными методами, такими как метод конечных элементов [1], метод конечных разностей [2], часто возникает необходимость нахождения решения систем линейных алгебраический уравнений (СЛАУ) [3] высоких порядков, матрица коэффициентов которых имеет выраженный диагональный (ленточный) характер. С распространением многопроцессорных вычислительных систем появилась возможность адаптации точных алгоритмов решения СЛАУ к данной архитектуре. Работы в этом направлении ведутся продолжительное время [4], разработано большое количество алгоритмов для вычислительных систем различной архитектуры [5-7]. Однако, предложенные алгоритмы рассматривают диагональную ленту матрицы коэффициентов как полностью заполненную ненулевыми элементами, хотя в практических задачах эта лента, как правило, содержит большое количество ненулевых диагоналей. Поэтому актуальной является задача разработки алгоритма, учитывающего внутреннюю разреженность диагональной ленты матрицы коэффициентов, что должно повысить его экономичность.

В работе предложен алгоритм параллельного решения СЛАУ с многодиагональной (многоленточной) матрицей коэффициентов на многопроцессорной вычислительной системе с общей памятью. Данный алгоритм реализует метод Гаусса [3] и состоит из трёх основных этапов: трансформация задачи, прямой ход алгоритма, обратный ход алгоритма.

1. Постановка задачи

Задача нахождения решения СЛАУ может быть представлена в виде

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{c}, \qquad (1)$$

где $\boldsymbol{A}$ - квадратная ленточная матрица коэффициентов с шириной ленты L, $\boldsymbol{x}$ - вектор-столбец неизвестных, $\boldsymbol{c}$ – вектор-столбец свободных членов. В общем случае $\boldsymbol{A}$, $\boldsymbol{x}$ и $\boldsymbol{c}$ имеют вид

$$\boldsymbol{A} = \begin{bmatrix} A_{11} & \dots & A_{1n} \\ \vdots & A_{ij} & \vdots \\ A_{n1} & \cdots & A_{nn} \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}. \tag{2}$$

Под решением задачи понимают такую совокупность чисел $u_1, u_2, \dots, u_n$, которая при подстановке на место неизвестных $x_1, x_2, \dots, x_n$ обращает все уравнения этой системы в тождества. Для большинства практических задач определитель матрицы $\boldsymbol{A}$ отличен от 0, следовательно, задача (1) имеет единственное решение.

Следует отметить следующую особенность решения СЛАУ с ленточной матрицей коэффициентов. Исходная матрица в практических задачах является сильно разреженной, однако, в ходе решения прямыми методами (таким, например, как метод Гаусса) лента полностью заполняется новыми ненулевыми элементами, что резко замедляет ход как последовательного, так и параллельного решения.

2. Трансформация задачи

На данном этапе производится изменение задачи (1) с целью уменьшения количества новых ненулевых элементов матрицы коэффициентов и упрощения процедуры распределения нагрузки между процессорами вычислительной системы.

Пусть N - число диагоналей, содержащих отличные от нуля элементы матрицы $\boldsymbol{A}$ (ненулевые диагонали), $\boldsymbol{d}_k, k \in [1:N]$ - вектор, содержащий элементы k-ой ненулевой диагонали. Размерность векторов $\boldsymbol{d_k}$ постоянна и равна n. Индексы компонентов векторов $\boldsymbol{d}_k, k \in [1:N]$ соответствуют индексам строк матрицы $\boldsymbol{A}$, в которых они находятся. Если какой-либо компонент векторов $\boldsymbol{d}_k, k \in [1:N]$ выходит за пределы матрицы $\boldsymbol{A}$, то данные компоненты тождественно равны 0. Будем считать, что диагональ матрицы $\boldsymbol{A}$, представленная вектором $\boldsymbol{d_1}$, не содержит нулевых элементов.

Введем смещение S – натуральное число, лежащее в интервале $[0, N-1]$. Значение S определяется по формуле

$$S = \max_{i \in [1:n]} (i) - 1, \quad A_{i,1} \neq 0.$$

Расширим вектор $\boldsymbol{x}$ задачи (1), добавив в его начало S переменных, а также введём в систему S дополнительных уравнений

$$\begin{cases} x_1 = 0, \\ x_2 = 0, \\ \dots \\ x_S = 0, \end{cases}$$

где $x_1, x_2, \dots, x_s$ - неизвестные, добавленные в (1). Введённым уравнениям присвоим номера $i, i \in [n+1:n+S]$. Тогда модифицированная задача примет вид

$$\boldsymbol{A}`\boldsymbol{x}` = \boldsymbol{c}`, \tag{3}$$

где $\boldsymbol{A}$` - матрица коэффициентов, $\boldsymbol{x}$` - вектор-столбец неизвестных и $\boldsymbol{c}$` - вектор-столбец свободных членов. Размерность матрицы $\boldsymbol{A}$` составит $(n + S)^2$, размерности векторов $\boldsymbol{x}$` и $\boldsymbol{c}$` будут равны $(n + S)$.

Далее к строкам матрицы $\boldsymbol{A}$` с индексом $i \in [1{:}S]$ прибавим строки матрицы $\boldsymbol{A}$`, полученные после модификации (1), по следующему правилу – к строке с индексом $i \in [1{:}S]$ прибавляется строка с индексом $j = i + n$.

Рассмотрим пример трансформации исходной задачи (1) размерности 5x5 со смещением $S = 2, L = 5$ и $N = 3$. Исходная задача имеет вид

x_1	x_2	x_3	x_4	x_5
$d_{2,1}$	0	$d_{3,1}$	0	0
0	$d_{2,2}$	0	$d_{3,2}$`	0
$d_{1,3}$	0	$d_{2,3}$	0	$d_{3,3}$
0	$d_{1,4}$	0	$d_{2,4}$	0
0	0	$d_{1,5}$	0	$d_{2,5}$

* | x_1 | x_2 | x_3 | x_4 | x_5 | = | c_1 | c_2 | c_3 | c_4 | c_5 | . (4)

Отметим, что в данной задаче $\boldsymbol{D}$ = ($\boldsymbol{d_1}$, $\boldsymbol{d_2}$, $\boldsymbol{d_3}$). Модифицируем задачу (4), используя рассмотренный выше алгоритм.

Так как S = 2, добавим две дополнительные переменные и два уравнения в задачу (4):

x_1	x_2	x_3	x_4	x_5	x_6	x_7
0	0	$d_{2,1}$	0	$d_{3,1}$	0	0
0	0	0	$d_{2,2}$	0	$d_{3,2}$`	0
0	0	$d_{1,3}$	0	$d_{2,3}$	0	$d_{3,3}$
0	0	0	$d_{1,4}$	0	$d_{2,4}$	0
0	0	0	0	$d_{1,5}$	0	$d_{2,5}$
1	0	0	0	0	0	0
0	1	0	0	0	0	0

* | x_1 | x_2 | x_3 | x_4 | x_5 | x_6 | x_7 | = | c_1 | c_2 | c_3 | c_4 | c_5 | 0 | 0

Прибавим последние две строки матрицы $\boldsymbol{A}$` к первым двум по вышеуказанному правилу, получим

x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	$d_{2,1}$	0	$d_{3,1}$	0	0
0	1	0	$d_{2,2}$	0	$d_{3,2}$`	0
0	0	$d_{1,3}$	0	$d_{2,3}$	0	$d_{3,3}$
0	0	0	$d_{1,4}$	0	$d_{2,4}$	0
0	0	0	0	$d_{1,5}$	0	$d_{2,5}$
1	0	0	0	0	0	0
0	1	0	0	0	0	0

* | x_1 | x_2 | x_3 | x_4 | x_5 | x_6 | x_7 | = | c_1 | c_2 | c_3 | c_4 | c_5 | 0 | 0 (5)

Отметим, что в модифицированной задаче (5) векторы ненулевых диагоналей имеют вид $\boldsymbol{d}_1 = (1, 1, d_{1,3}, d_{1,4}, d_{1,5})$, $\boldsymbol{d}_2 = (d_{2,1}, d_{2,2}, d_{2,3}, d_{2,4}, d_{2,5})$, $\boldsymbol{d}_3 = (d_{3,1}, d_{3,2}, d_{3,3}, 0, 0)$.

На основе теоремы Кронекера – Капелли легко показать существование и единственность решения задачи (3) при условии существования и единственности решения задачи (1).

Необходимо отметить, что проведённая трансформация системы уравнений привела её матрицу коэффициентов к верхнетреугольному виду с нижним окаймлением. Это означает, что в прямом ходе метода Гаусса все подлежащие исключению ненулевые элементы матрицы (в том числе и вновь возникающие) располагаются только в её *S* последних строках.

3. Представление данных

В задаче (3), как правило, матрица $\boldsymbol{A}`$ является сильно разреженной, поэтому нецелесообразно хранить все элементы матрицы. Так уже при *n* = 10000 для хранения матрицы ***A,*** каждый элемент которой представлен числом с плавающей точкой двойной точности, требуется около 745 Мб.

Матрица $\boldsymbol{A}`$ полностью определяется следующими параметрами и векторами:

- *n* – размерность одной строки матрицы ***A***;
- *S* – смещение матрицы ***A***;
- *L* – ширина ленты матрицы ***A***;
- *N* – число ненулевых диагоналей матрицы ***A***;
- $\boldsymbol{d}_k,\ k \in [1{:}N]$ – векторы размерностью *n,* содержащие элементы k-ой ненулевой диагонали матрицы ***A***;
- $\boldsymbol{p}$ – вектор размерности N, где $\boldsymbol{p}_i, i \in [1{:}N]$ равен номеру *i*-ой ненулевой диагонали, отсчитанному от $\boldsymbol{d_1}$.

Вектора ***c*** и ***x*** сохраняют свое представление, указанное в (2).

Для матрицы $\boldsymbol{A}`$ дополнительно введем векторы $\boldsymbol{b_t}, t \in [1{:}S]$. Каждый из этих векторов имеет размерность *L* и содержит отличные от 0 элементы строки матрицы $\boldsymbol{A}`$ с номером $(n + t), t \in [1{:}S]$.

При таком подходе к представлению данных система из 10000 уравнений с $N = 3$ и $L = 100$ занимает около 0,3 Мб. Без введения вектора ***p*** эта же задача занимала бы в памяти 7,62 Мб (нет возможности учесть диагонали, состоящие из нулей внутри ленты).

4. Прямой ход алгоритма

Первоначально первые *S* уравнений задачи (3) вычитаются из последних *S* уравнений системы по следующему правилу: из уравнения с номером $j \in [n+1{:}n+S]$ вычитается уравнение с номером $i = j - n$. Для примера (4) получим систему

x_1	x_2	x_3	x_4	x_5	x_6	x_7				
1	0	$d_{2,1}$	0	$d_{3,1}$	0	0		x_1		c_1
0	1	0	$d_{2,2}$	0	$d_{3,2}$	0		x_2		c_2
0	0	$d_{1,3}$	0	$d_{2,3}$	0	$d_{3,3}$	*	x_3	=	c_3
0	0	0	$d_{1,4}$	0	$d_{2,4}$	0		x_4		c_4
0	0	0	0	$d_{1,5}$	0	$d_{2,5}$		x_5		c_5
0	0	$-d_{2,1}$	0	$-d_{3,1}$	0	0		x_6		$-c_1$
0	0	0	$-d_{2,2}$	0	$-d_{3,2}$	0		x_7		$-c_2$

Тогда векторы $\boldsymbol{b_t}, t \in [1:2]$ принимают вид $\boldsymbol{b_1} = (-d_{2,1}, 0, -d_{3,1}, 0, 0)$, $\boldsymbol{b_2} = (0, -d_{2,2}, 0, -d_{3,2}, 0)$.

Далее начинается итерационный процесс, состоящий из $(n - S)$ итераций. Каждая итерация состоит из двух шагов. На первом шаге компоненты векторов $\boldsymbol{b_t}, t \in [1:S]$ и $\boldsymbol{c}$` вычисляются по формулам

$$b_{t,j}^{i} = b_{t,j}^{i-1} - K_i \cdot d_{i+s,j},$$

$$c`_{n+t}^{i} = c`_{n+t}^{i-1} - K_i \cdot c`_i ,$$

$$K_i = \frac{b_{t,1}^{i-1}}{d_{1,i+s}},$$

$$j = p_m, m \in [1:N], i \in [1:n-S], t \in [1:S] \quad (6)$$

В выражении (6) индекс *i* - номер итерации, j - номер ненулевой диагонали в векторе $\boldsymbol{p}$, стоящий на позиции *m*, $b_{t,j}^{i}$ – значение *j*-го компонента вектора $\boldsymbol{b_t}$ на итерации с номером *i*, $c`_{n+t}^{i}$ – значение компоненты вектора $\boldsymbol{c}$` с индексом $(n+t)$ на итерации с номером *i*.

На втором шаге итерации выполняется циклический сдвиг векторов $\boldsymbol{b_t}, t \in [1:S]$ на один элемент влево.

Из выражения (6) легко видеть возможность независимого вычисления компонентов векторов $\boldsymbol{b_t}, t \in [1:S+1]$, что позволяет параллельно производить вычисления в общем случае на $(S \cdot N)$ независимых узлах вычислительной системы.

После выполнения $(n - S)$ итераций в правом нижнем углу матрицы коэффициентов образуется (возможно, полностью заполненная) подматрица $\boldsymbol{Q}$ размерности *S*x*S*. Для примера (4) получаем

x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	$d_{2,1}$	0	$d_{3,1}$	0	0
0	1	0	$d_{2,2}$	0	$d_{3,2}$	0
0	0	$d_{1,3}$	0	$d_{2,3}$	0	$d_{3,3}$
0	0	0	$d_{1,4}$	0	$d_{2,4}$	0
0	0	0	0	$d_{1,5}$	0	$d_{2,5}$
0	0	0	0	0	$q_{1,1}$	$q_{1,2}$
0	0	0	0	0	$q_{2,1}$	$q_{2,2}$

*

x_1
x_2
x_3
x_4
x_5
x_6
x_7

=

c_1
c_2
c_3
c_4
c_5
c_6
c_7

При этом векторы $\boldsymbol{b_t}, t \in [1:2]$ имеют вид $\boldsymbol{b_1} = (0,0,0,q_{1,1},q_{1,2})$, $\boldsymbol{b_2} = (0,0,0,q_{2,1},q_{2,2})$. В частном случае некоторые диагональные элементы $q_{i,i}, i \in [1:S]$ могут быть равны нулю, что потребует переупорядочивания последних *S* строк матрицы $\boldsymbol{A}$`.

Перед выполнением следующего этапа требуется синхронизировать работу программы. Для последних *S* уравнений задачи (3) выполняется прямой ход метода Гаусса. Для примера (4) имеем

x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	$d_{2,1}$	0	$d_{3,1}$	0	0
0	1	0	$d_{2,2}$	0	$d_{3,2}$	0
0	0	$d_{1,3}$	0	$d_{2,3}$	0	$d_{3,3}$
0	0	0	$d_{1,4}$	0	$d_{2,4}$	0
0	0	0	0	$d_{1,5}$	0	$d_{2,5}$
0	0	0	0	0	1	$q`_{1,2}$
0	0	0	0	0	0	1

*

x_1
x_2
x_3
x_4
x_5
x_6
x_7

=

c_1
c_2
c_3
c_4
c_5
$c`_6$
$c`_7$

Тогда вектора $\boldsymbol{b_t}, t \in [1:2]$ примут вид $\boldsymbol{b_1} = (0,0,0,1,q`_{1,2})$, $\boldsymbol{b_2} = (0,0,0,0,1)$.

5. Обратный ход алгоритма

Обратный ход алгоритма полностью совпадает с аналогичным этапом канонического метода Гаусса.

6. Схема метода для многопроцессорной вычислительной системы

Рассматриваем вычислительную систему с общей памятью, включающую *p* процессоров. Схема метода решения задачи на вычислительной системе данной архитектуры выглядит следующим образом.

1) Какой-либо из процессоров назначается главным (*master*-процессор), а все остальные - вспомогательными (*slave*-процессоры). На master-процессоре запускается основной процесс алгоритма.

2) В оперативную память системы загружаются исходные данные решаемой задачи.

3) На *master*–процессоре выполняем модификацию задачи (1) согласно изложенному выше алгоритму.

4) Запускается p потоков управления, каждому из которых передаётся на обработку $z=\left\lceil\frac{S}{p}\right\rceil$ векторов $\boldsymbol{b_t}, t \in [1{:}S]$. Если S не кратно p, то последний поток управления получит меньше векторов $\boldsymbol{b_t}, t \in [1{:}S]$ чем остальные. Распределение потоков управления по процессорам выполняется планировщиком операционной системы. При обработке данных участвуют как *master*-процессор, так и *slave*-процессоры.

5) Каждый поток управления, используя (6), производит модификацию векторов $\boldsymbol{b_t}, t \in [i \cdot z + 1{:}(i+1) \cdot z]$, где i – номер потока управления.

6) На *master*-процессоре выполняется прямой ход стандартного метода Гаусса для последних S уравнений модифицированной задачи.

7) На *master*-процессоре выполняется обратный ход метода Гаусса.

7. Аналитическая оценка эффективности алгоритма

Для начала оценим время выполнения итерации алгоритма на одном процессоре. На каждой итерации требуется выполнить не более чем S операций деления, $S \cdot N$ операции умножения и $S \cdot N$ операций вычитания.

Пусть среднее время выполнения одной операции деления над числами с плавающей точкой двойной точности задаётся константой $t_{\text{дел}}$, а время выполнения прочих арифметических операций - константой $t_{\text{арф}}$. Тогда аналитическое выражение для времени выполнения прямого хода для всей системы имеет вид

$$T(n,N,S) = (n-S) \cdot (S) \cdot (2 \cdot N \cdot t_{\text{арф}} + t_{\text{дел}}) + S^3 \cdot t_{\text{арф}}, \quad (7)$$

где $S^3 \cdot t_{\text{арф}}$ – слагаемое, введенное для учета верхней границы числа арифметических операций, требуемых для решения СЛАУ из S уравнений в конце выполнения прямого хода алгоритма. Константы $t_{\text{арф}}, t_{\text{дел}}$ определяются экспериментально.

Оценим время выполнения данного алгоритма на многопроцессорной системе с общей оперативной памятью, используя рассмотренную выше схему метода:

$$T_{\text{мп}}(n,N,S,p) = T_{орг} + T_{\text{выч}}(n,N,S,p). \quad (8)$$

В (8) время $T_{орг}$ - накладные расходы на модификацию задачи (1), запуск p потоков и распределения их по процессорам, $T_{\text{выч}}$ – время выполнения вычислительных операций всеми процессорами:

$$T_{\text{выч}}(n,N,S,p) = \max_{i=1;i\le p}(T_i(n,N,Z)) + S^3 \cdot t_{\text{арф}}. \quad (9)$$

В формуле (9) величина $T_i, i \in [1{:}p]$ – время решения части задачи i-ым процессором. В данную величину следует включить время обращения процессора к общей оперативной памяти.

В случае, если все процессоры системы имеют одинаковую производительность и не имеют дополнительной вычислительно нагрузки, выражение (8) принимает вид

$$T_{\text{мп}}(n,N,S,p) = T_{орг} + T_{\max}(n,N,Z) + S^3 \cdot t_{\text{арф.}} \quad (10)$$

Отсюда получаем выражение для ускорения вычислительного процесса

$$S_{\text{мп}}(n,N,S,p) = \frac{T(n,N,S)}{T_{\text{мп}}(n,N,S,p)}. \quad (11)$$

Ниже приведены результаты аналитического исследования алгоритма.

Пример 1. Отобразим зависимость $S_{\text{мп}}(p), \text{p} \in [1{:}64]$ для матрицы с n = 25000000, N = 5, S = 2500. Количество вспомогательных векторов $\boldsymbol{b_t}, t \in [1{:}S]$ при модификации задачи составляет *2500*, объем требуемой памяти - 1,02 Гб. Организационное время на создание одного потока принимается постоянным и составляет 0,0003 сек, $t_{\text{арф}} \approx 6{,}4 \cdot 10^{-10}$сек/оп. Полученная зависимость приведена на рисунке 1.

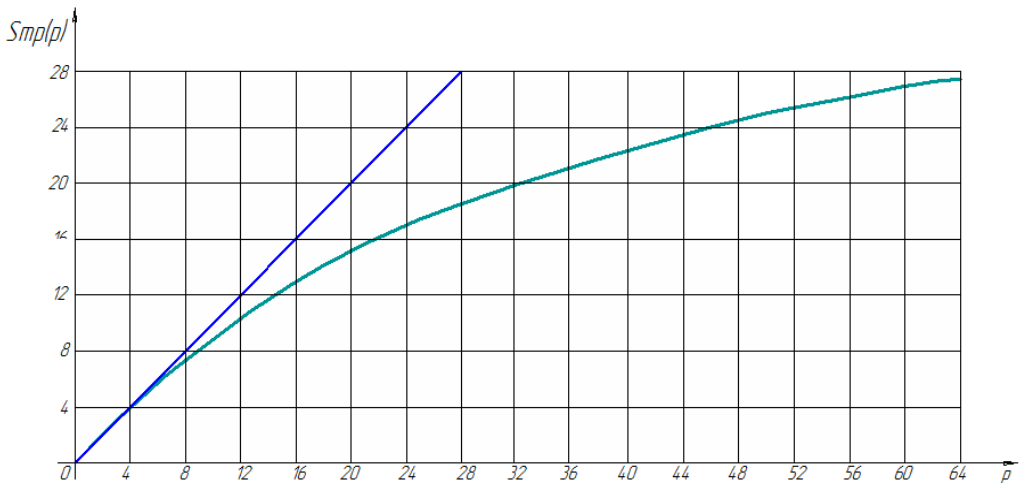

Рисунок 1. Зависимость $S_{\text{mp}}(n,N,S,p)$ от числа процессоров p для примера 1

На рисунке 2 дан график зависимости ускорения от смещения S ленты матрицы коэффициентов при фиксированном количестве процессов p=8.

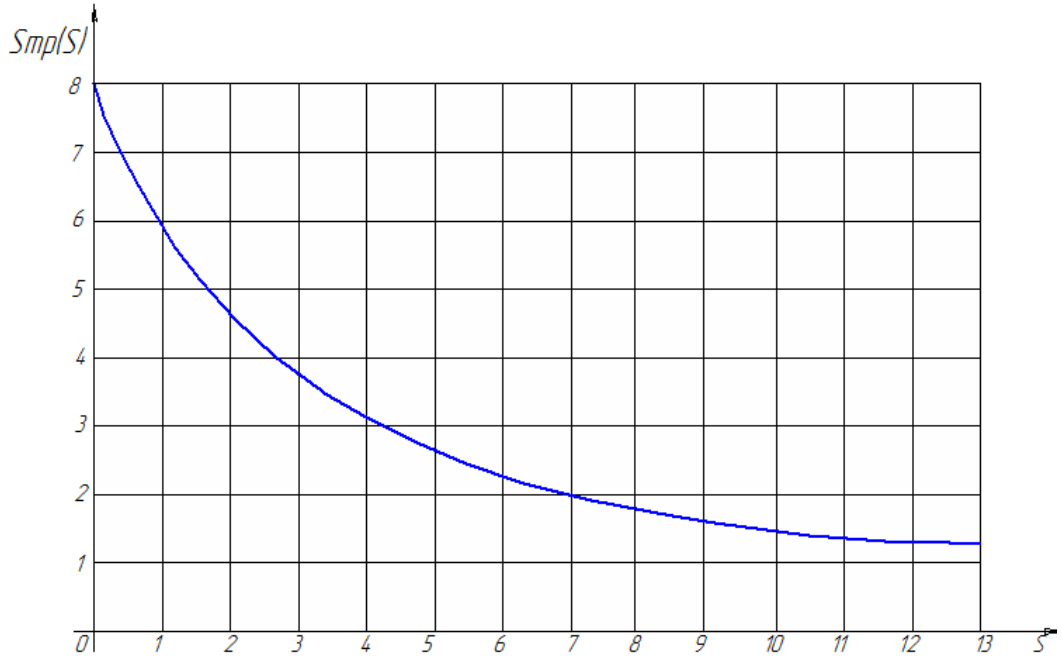

Рисунок 2. Зависимость $S_{мп}(n,N,S,p)$ от смещения для примера 1

Пример 2. Здесь приводятся зависимости ускорения $S_{мп}(p), p \in [1:64]$ при $S = 5000$ и $S_{мп}(S), S \in [1:\frac{n}{2}-1]$ для задачи с n = 100000000, N = 10000, остальные исходные данные аналогичны примеру 1. Ниже на рисунках 3 и 4 приведены зависимости аналогичные зависимостям на рисунках 1 и 2.

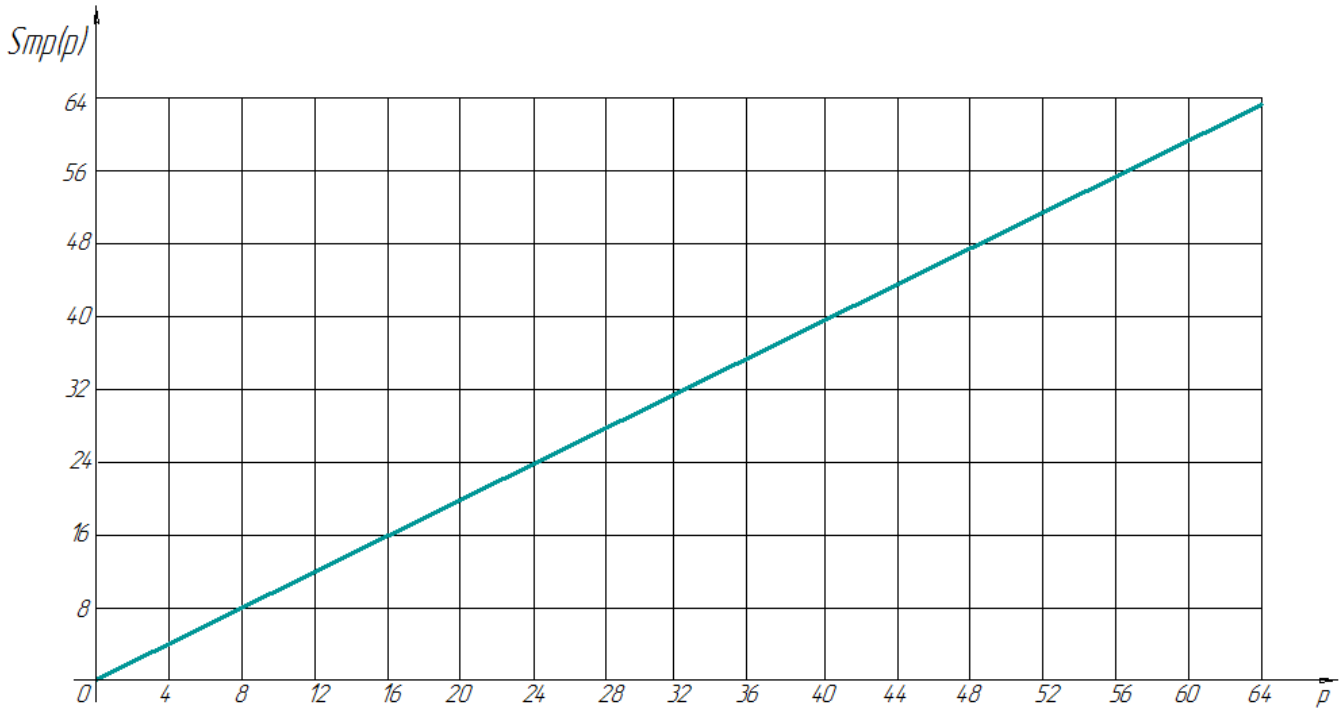

Рисунок 3. Зависимость $S_{мп}(n,N,S,p)$ от числа процессоров P для примера 2

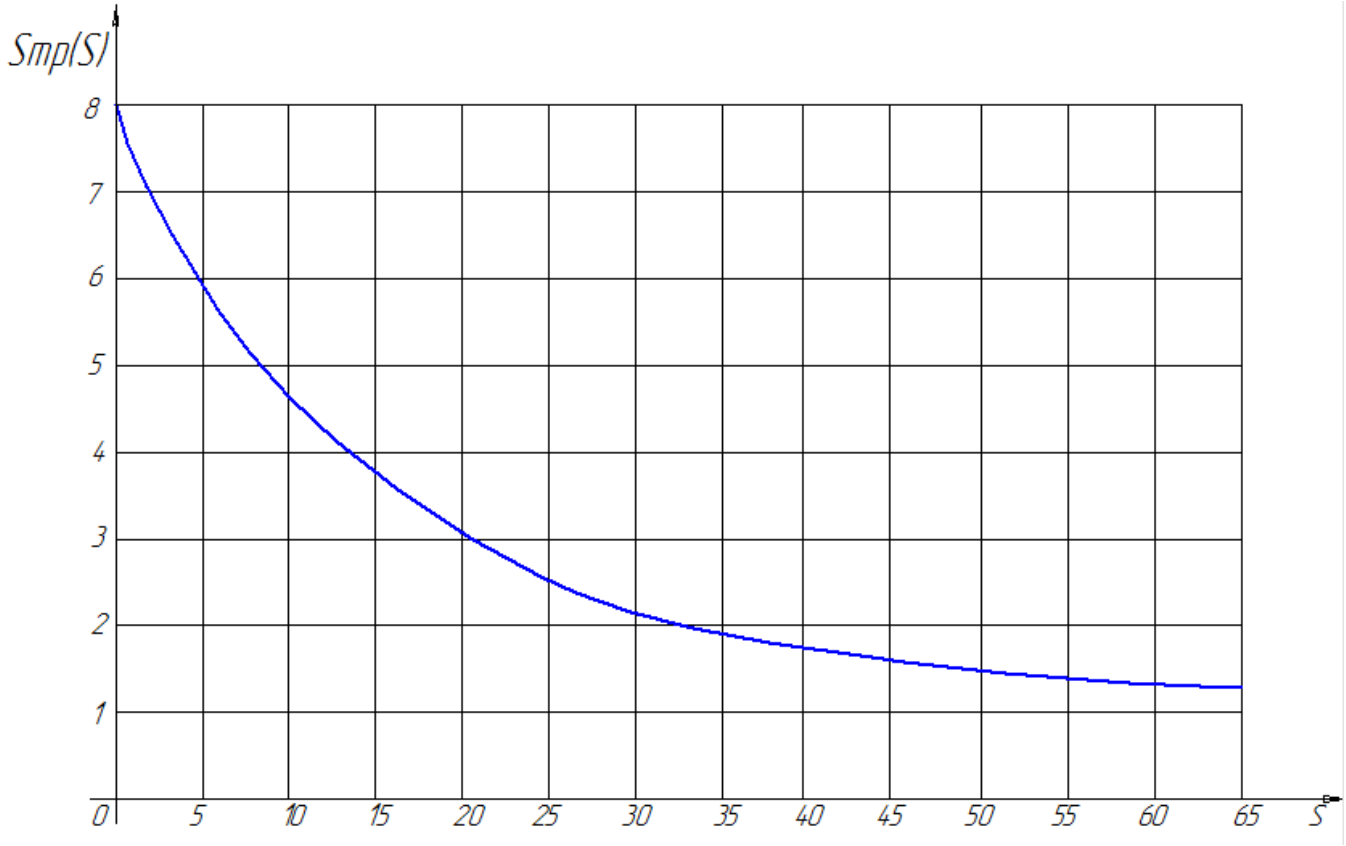

Рисунок 4. Зависимость $S_{мп}(n, N, S, p)$ от смещения для примера 2

Пример 3. Здесь даны зависимости ускорения $S_{мп}(p), p \in [1:64]$ при $S = 700$ и $S_{мп}(S), S \in [1:\frac{n}{2} - 1]$ для матрицы $c\, n = 3000000, N = 5,$. Ниже на рисунках 5 и 6 приведены зависимости аналогичные зависимостям на рисунках 1 и 2.

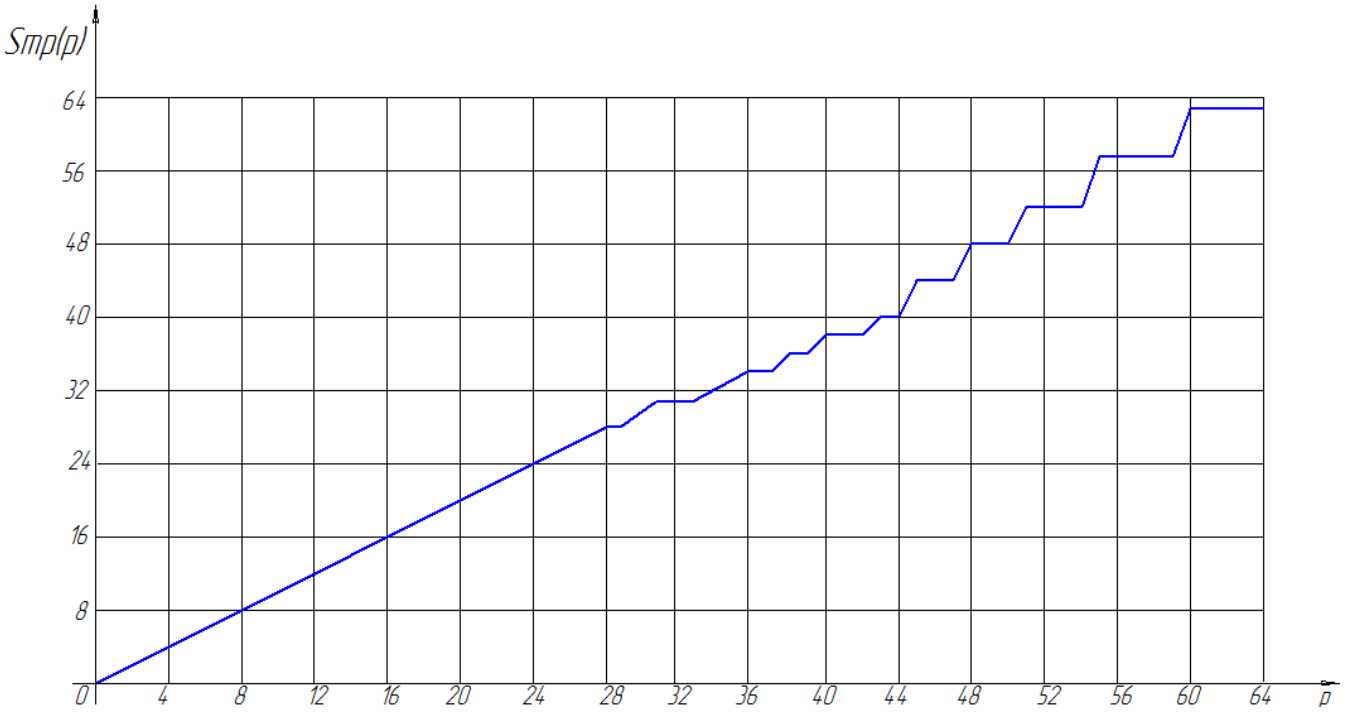

Рисунок 5. Зависимость $S_{мп}(n, N, S, p)$ от числа процессоров *p* для примера 3

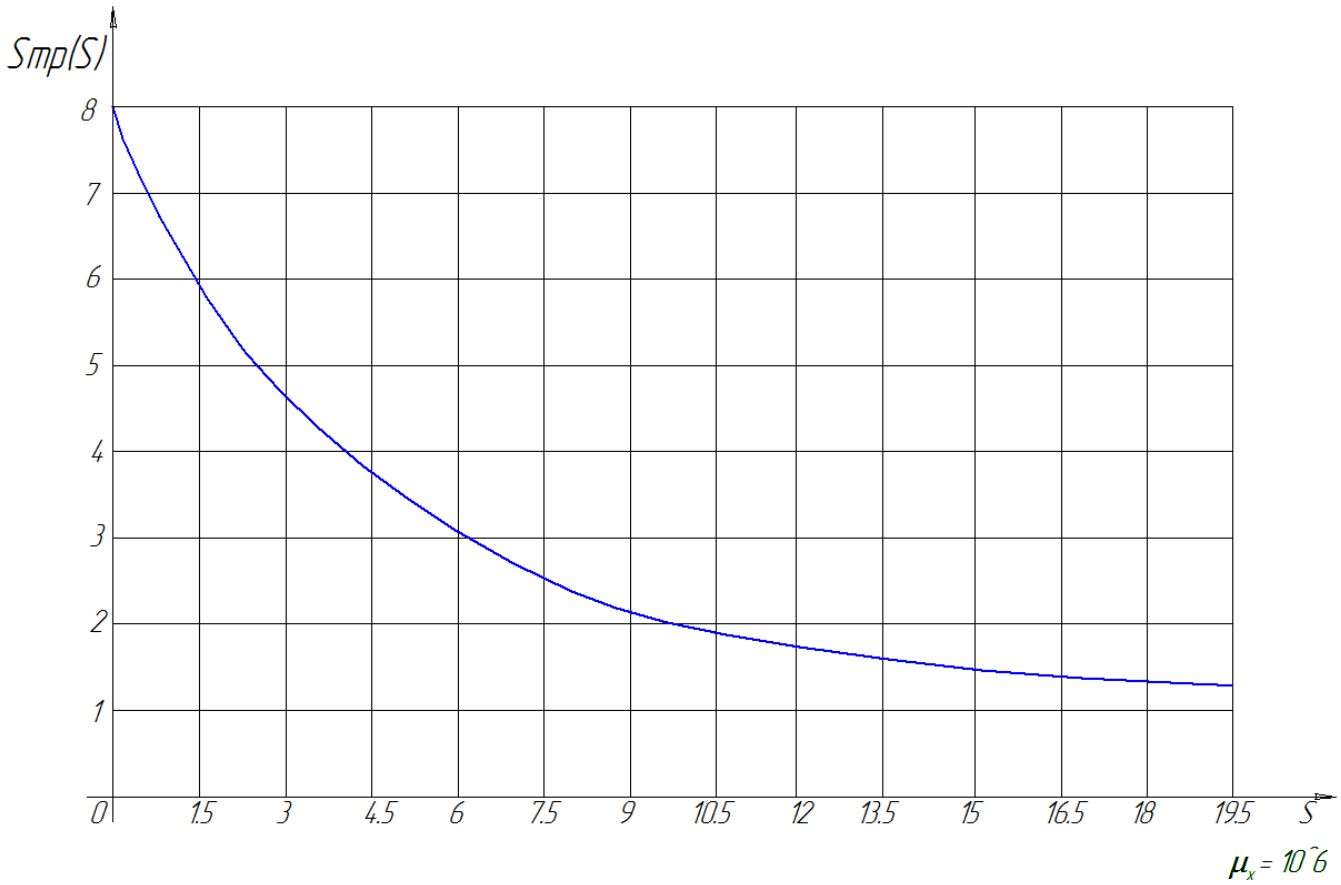

Рисунок 6. Зависимость $S_{мп}(n, N, S, p)$ от смещения для примера 2

Пример 4. Здесь исследуется зависимость ускорения $S_{мп}(p), p \in [1:64]$ для задачи небольшой размерности $n = 10000, N = 100, S = 50$, результат представлен на рисунке 7.

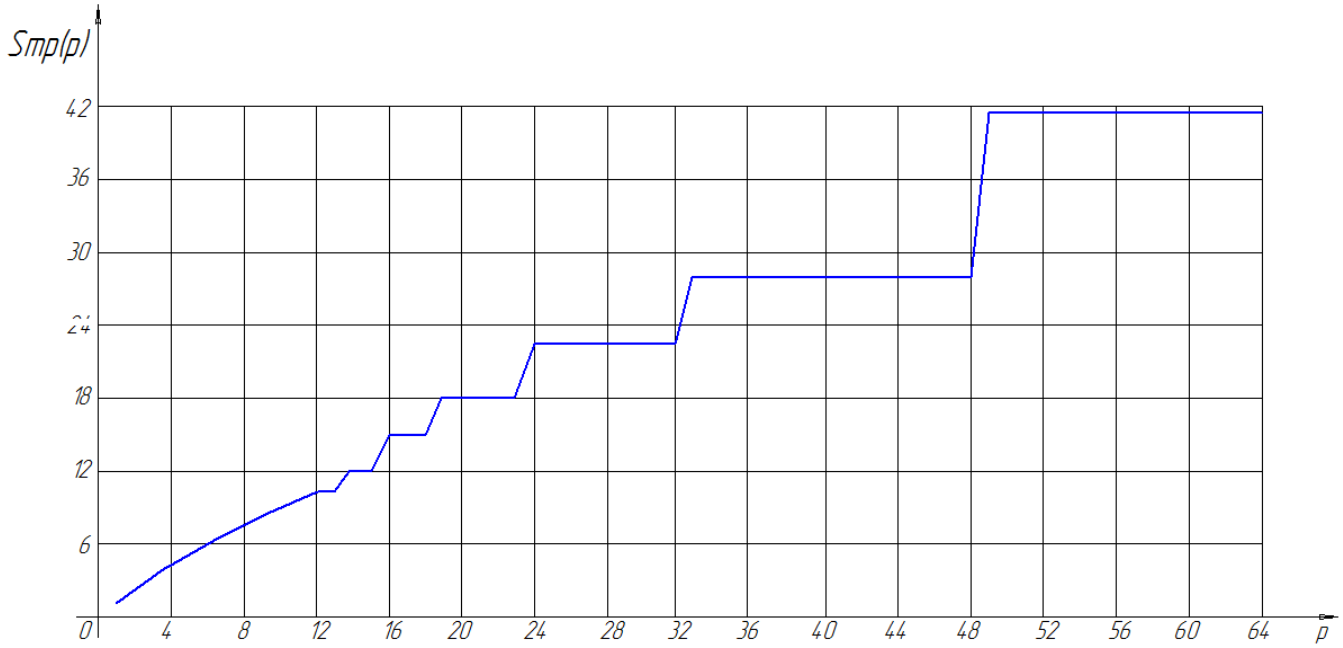

Рисунок 7. Зависимость $S_{мп}(n, N, S, p)$ от числа процессоров *p* для примера 4

Из рассмотренных выше примеров очевидно, что ускорение слабо зависит от размерности задачи, а зависит лишь от смещения. Ступеньки на рисунках 1, 3, 5, 7 появились из-за несбалансированности загрузки системы, полученной в результате некратности чисел *S* и *p*. Так при увеличении *p* не всегда стоит ожидать кратного увеличения ускорения. Из рисунков 2, 4, 6 видно, как с ростом смещения падает ускорение, это связано с преобладанием слагаемого $S^3 t_{арф}$ в (9). Наибольшая эффективность алгоритма достигается при равенстве количества процессоров системы *p* и смещения *S*.

8. Программная реализация

Алгоритм реализован на языке высокого уровня C++ [8] с использованием стандартных библиотек многопоточного программирования операционной системы UNIX [9].

Ниже приведены результаты численных экспериментов для некоторых СЛАУ.

n = 100000, *S* = 400, *N* = 5.

Кол-во ядер	1	2	3	4
$T_{мп}$ (сек.)	8,12	4,42	3,03	2,09
$S_{мп}$	1,00	1,83	2,67	3.87

n = 100000, S = 700, N =5.

Кол-во ядер	1	2	3	4
$T_{мп}$ (сек.)	31,40	16,22	11,17	8,19
$S_{мп}$	1,00	1,93	2,81	3,83

n = 3000000, S = 400, N = 5.

Кол-во ядер	1	2	3	4
$T_{мп}$ (сек.)	243	127	86	64
$S_{мп}$	1,00	1,910	2,820	3,790

n = 3000000, S = 700, N = 5.

Кол-во ядер	1	2	3	4
$T_{мп}$ (сек.)	957	492	322	254
$S_{мп}$	1,00	1,94	2,97	3,76

Далее для n = 3000000, N = 5 приведена зависимость ускорения $S_{мп}$от S при использовании четырёхядерной вычислительной системы.

S	16	32	64	128	256	512	1024	2048
$T_{мп}$ (сек.)	0,001	0,095	3,15	7,39	75,11	93	1192	11260
T_1 (сек.)	0,0037	0,254	11,28	26,17	266,93	328	2250	18353
$S_{мп}$	3,70	3,58	3,58	3,55	3,55	3,52	1,88	1,63

Данные результаты полностью согласуются с аналитической оценкой эффективности алгоритма. Так с ростом ширины ленты эффективность алгоритма падает, а при одинаковой ширине ленты ускорение практически не зависит от размерности исходной задачи n.

Заключение

В статье предложен новый параллельный алгоритм решения систем линейных алгебраических уравнений с диагонально-ленточной матрицей коэффициентов, учитывающий наличие нулевых элементов в отдельных диагоналях внутри ленты.

Основными достоинствами алгоритма являются следующие.

- Простота программной реализации.
- Высокие показатели ускорения для задач с малым смещением ленты матрицы коэффициентов.
- Экономия памяти вычислительной системы.
- Возможность простой адаптации под различные вычислительные системы с общей памятью.
- Отсутствие множественного доступа по записи к исходным данным.

К недостаткам алгоритма относятся следующие.

- Падение ускорения с увеличением смещения S ленты матрицы $\boldsymbol{A}$.

- Необходимость выделения дополнительной памяти на этапе модификации исходной задачи.

- Возможная несбалансированность нагрузки процессоров вычислительной системы.

Список литературы

1. Ильин В.А., Позняк Э.Г. Линейная алгебра. М.: Физматлит, 1999. 280 с.
2. Ильин В.А., Ким Г.Д. Линейная алгебра и аналитическая геометрия. М.: Проспект, 2007. 400 с.
3. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем: пер. с англ. М.: Мир, 1991. 367 с. [Ortega J.M. Introduction to Parallel and Vector Solution of Linear Systems. NY.: Plenum Press, 1988. 305 p.]
4. Tewarson R.P. Solution of linear equations with coefficient matrix in band form // BIT Numerical Mathematics. 1968. Vol. 8, iss. 1. P. 53-58. DOI: 10.1007/BF01939979
5. Saad Y., Schultz H. Parallel direct methods for solving banded linear systems // Linear Algebra Appl. 1987. Vol. 88-89. P. 623-650. http://dx.doi.org/10.1016/0024-3795(87)90128-5
6. Polizzi E., Sameh A. SPIKE: A parallel environment for solving banded linear systems // Computers & Fluids. 2007. Vol. 36, no. 1. P. 113-141. http://dx.doi.org/10.1016/j.compfluid.2005.07.005
7. Krüger J., Westermann R. GPU Gems 2. Chapter 44. A GPU Framework for Solving Systems of Linear Equations. Режим доступа: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter44.html (дата обращения 19.06.2013).
8. Эккель Б. Философия C++. Введение в стандартный C++. СПб.: Питер, 2004. 572 с. [Eckel B. Thinking in C++: Introduction to Standard C++. NY.: Prentice Hall, 2000. 814 p.]
9. Чан Т. Системное программирование на C++ для Unix. М.: BHV, 1997. 490 с. [Chan T. UNIX System Programming using C++. Prentice Hall, 1997. 598 p.]

SCIENTIFIC PERIODICAL OF THE BAUMAN MSTU

SCIENCE and EDUCATION

EL № FS77 - 48211. №0421200025. ISSN 1994-0408

electronic scientific and technical journal

Parallel algorithm for solving systems of linear algebraic equations with a multi-diagonal coefficient matrix

 # 07, July 2013

DOI: 10.7463/0713.0590785

Jeldakov A.V., Федорук В. Г.

Bauman Moscow State Technical University, 105005, Moscow, Russian Federation

www.streeter@mail.ru

fedoruk.bmstu@mail.ru

This article describes a parallel algorithm for solving systems of linear algebraic equations with a multi-diagonal (band) coefficient matrix. A scheme of the algorithm for multi-processor shared-memory computing systems is also presented in the article. Data structures for a compact storage of sparse band matrices were implemented. Theoretical and experimental investigations of efficiency of the algorithm were conducted using various problems. Dependences of the program's speedup on various parameters of the problem and the number of processors were given in this work. The conclusion presents pros and cons of the algorithm.

Publications with keywords: parallel algoritm, system of linear equations, multidiagonal matrix of coefficients, band matrix of coefficients, multiprocessor system

Publications with words: parallel algoritm, system of linear equations, multidiagonal matrix of coefficients, band matrix of coefficients, multiprocessor system

References

1. Il'in V.A., Poznyak E.G. *Lineynaya algebra* [Linear algebra]. Moscow, Fizmatlit, 1999. 280 p.

2. Il'in V.A., Kim G.D. *Lineynaya algebra i analiticheskaya geometriya* [Linear algebra and analytical geometry]. Moscow, Prospekt, 2007. 400 p.

3. Ortega J.M. *Introduction to Parallel and Vector Solution of Linear Systems*. NY, Plenum Press, 1988. 305 p. (Russ. ed.: Ortega Dzh. *Vvedenie v parallel'nye i vektornye metody resheniya lineynykh system*. Moscow, Mir, 1991. 367 p.).

4. Tewarson R.P. Solution of linear equations with coefficient matrix in band form. *BIT Numerical Mathematics,* 1968, vol. 8, iss. 1, pp. 53-58. DOI: 10.1007/BF01939979

5. Saad Y., Schultz H. Parallel direct methods for solving banded linear systems. *Linear Algebra Appl*., 1987, vol. 88-89, pp. 623-650. http://dx.doi.org/10.1016/0024-3795(87)90128-5

6. Polizzi E., Sameh A. SPIKE: A parallel environment for solving banded linear systems. *Computers & Fluids*, 2007, vol. 36, no. 1, pp. 113-141. http://dx.doi.org/10.1016/j.compfluid.2005.07.005

7. Krüger J., Westermann R. *GPU Gems 2. Chapter 44. A GPU Framework for Solving Systems of Linear Equations*. Available at: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter44.html , accessed 19.06.2013.

8. Eckel B. *Thinking in C++: Introduction to Standard C++*. NY, Prentice Hall, 2000. 814 p. (Russ. ed.: Ekkel' B. *Filosofiya C++. Vvedenie v standartnyy C++*. St. Petersburg, Piter, 2004. 572 p.).

9. Chan T. *UNIX System Programming using C++*. Prentice Hall, 1997. 598 p. (Russ. ed.: Chan T. *Sistemnoe programmirovanie na C++ dlya Unix*. Moscow, BHV, 1997. 490 p.).