**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра вычислительной математики**

# РЕАЛИЗАЦИЯ НА OPENMP МЕТОДА ГАУССА-ЗЕЙДЕЛЯ ДЛЯ СИСТЕМ УРАВНЕНИЙ СПЕЦИАЛЬНОГО ВИДА

Курсовая работа

Бобовоза Владислава
Сергеевича
студента 4 курса,
специальность «прикладная
математика»

Научный руководитель:
кандидат физ.-мат. наук,
доцент
А.А. Толстиков

Минск, 2024

## РЕФЕРАТ

Курсовая работа, 37 с., 4 источника, 5 рис., 3 таблицы, 2 прил.

РЕАЛИЗАЦИЯ НА OPENMP МЕТОДА ГАУССА-ЗЕЙДЕЛЯ ДЛЯ СИСТЕМ УРАВНЕНИЙ СПЕЦИАЛЬНОГО ВИДА, OPENMP, МЕТОД ГАУССА-ЗЕЙДЕЛЯ, CPU

*Объект исследования* – метод Гаусса-Зейделя с использованием OpenMP.

*Цель работы* – разработать псевдокод последовательного и параллельного алгоритма Гаусса-Зейделя для численного решения блочно-трехдиагональных систем с трехдиагональными блоками, реализовать эти алгоритмы, сравнить и проанализировать результаты.

*Методы исследования* – технология OpenMP, метод Гаусса-Зейделя, анализ эффективности.

*Результаты работы:* разработаны псевдокоды последовательного и параллельного алгоритма Гаусса-Зейделя для численного решения блочно-трехдиагональных систем с трехдиагональными блоками, реализованы эти алгоритмы, а также проведено сравнение полученных результатов.

## РЭФЕРАТ

Курсавая праца, 37 с., 4 крыніцы, 5 мал., 3 табліцы, 2 дадаткі

РЭАЛІЗАЦЫЯ НА OPENMP МЕТАДУ ГАЎСА-ЗАЙДЭЛЯ ДЛЯ СІСТЭМ РАЎНАННЯЎ АДМЫСЛОВАГА ВЫГЛЯДУ, OPENMP, МЕТАД ГАЎСА-ЗАЙДЭЛЯ, CPU

*Аб'ект даследавання* – метад Гаўса-Зайдэля з выкарыстаннем OpenMP.

*Мэта працы* – распрацаваць псеўдакод паслядоўнага і паралельнага алгарытму Гаўса-Зайдэля для лікавага развязку блокава-трохдыяганальных сістэм з трохдыяганальнымі блокамі, рэалізаваць гэтыя алгарытмы, параўнаць і прааналізаваць вынікі.

*Метады даследавання* – тэхналогія OpenMP, метад Гаўса-Зайдэля, аналіз эфектыўнасці.

*Вынікі працы*: распрацаваны псеўдакоды паслядоўнага і паралельнага алгарытму Гаўса-Зайдэля для лікавага развязку блокава-трохдыяганальных сістэм з трохдыяганальнымі блокамі, рэалізаваны гэтыя алгарытмы, а таксама праведзена параўнанне атрыманых вынікаў.

## SUMMARY

Course work, 37 p., 4 sources, 5 fig., 3 tables, 2 app.

REALIZATION ON OPENMP OF GAUSS-SEIDEL METHOD FOR SYSTEMS OF EQUATIONS OF SPECIAL KIND, OPENMP, GAUSS-SEIDEL METHOD, CPU

*The object of study* – Gauss-Seidel method using OpenMP.

*The purpose of the work* – develop pseudo-code of sequential and parallel Gauss-Seidel algorithm for numerical solution of block-triple-diagonal systems with triple-diagonal blocks, implement these algorithms, compare and analyze the results.

*Research methods* – OpenMP technology, Gauss-Seidel method, efficiency analysis.

*Results of the work*: pseudo-codes of the sequential and parallel Gauss-Seidel algorithm for numerical solution of block-tri-diagonal systems with tridiagonal blocks have been developed, these algorithms have been implemented, and the results obtained have been compared and analyzed.

## ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

В настоящее время, численные методы и вычислительная математика играют важную роль в решении сложных задач физики, инженерии, экономики и других наук. Одной из фундаментальных проблем является решение систем линейных уравнений, которые возникают при моделировании различных процессов. Метод Гаусса-Зейделя является эффективным итерационным методом для нахождения приближенных решений.

Однако, зачастую приходится решать системы уравнений огромных размеров и довольно сложных структур. Именно в таких случаях помогают различные технологии параллельного программирования. OpenMP является одним из наиболее распространенных инструментов для создания параллельных приложений на многоядерных процессорах, все благодаря своей простоте и эффективности.

Актуальность данной работы обусловлена потребностью в ускорении вычислений при решении больших и сложных систем уравнений специального вида. В частности, параллельная реализация метода Гаусса-Зейделя с использованием OpenMP позволяет значительно сократить время вычислений и повысить производительность.

Целью данной курсовой работы является разработка и реализация последовательного и параллельного алгоритма метода Гаусса-Зейделя для систем уравнений специального вида с использованием технологии OpenMP, а также анализ эффективности параллельного подхода по сравнению с последовательной реализацией.

В рамках работы будут рассмотрены теоретические аспекты метода Гаусса-Зейделя, особенности его применения к системам уравнений специального вида и принципы параллельного программирования с OpenMP.

## 1 Представление блочно-трехдиагональных систем уравнений с трехдиагональными блоками

Пусть $N_1$, $N_2$ – некоторые натуральные числа, а $A_1, \dots, A_{N_1}$, $B_0, B_1, \dots, B_{N_1-1}$, $C_0, C_1, \dots, C_{N_1}$ – вещественные матрицы порядка $N_2 - 1$, а $Y_0, Y_1, \dots, Y_{N_1}$, $F_0, F_1, \dots, F_{N_1}$ – $(N_2 - 1)$-мерные векторы. Тогда, рассмотрим блочно-трехдиагональную систему линейных алгебраических уравнений вида [1]:

$$\begin{cases} C_0 Y_0 + B_0 Y_1 = F_0 \\ A_i Y_{i-1} + C_i Y_i + B_i Y_{i+1} = F_i, \qquad i = \overline{1, N_1 - 1}, \\ A_{N_1} Y_{N_1-1} + C_{N_1} Y_{N_1} = F_{N_1} \end{cases} \tag{1.1}$$

Будем рассматривать случай трехдиагональных матриц-блоков:

$$C_i = \begin{pmatrix} c_1^{C_i} & b_1^{C_i} & 0 & 0 & \cdots \\ a_2^{C_i} & c_2^{C_i} & b_2^{C_i} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & a_{N_2-2}^{C_i} & b_{N_2-2}^{C_i} & c_{N_2-2}^{C_i} \\ \cdots & 0 & 0 & a_{N_2-1}^{C_i} & c_{N_2-1}^{C_i} \end{pmatrix}, \qquad i = \overline{0, N_1}$$

$$A_i = \begin{pmatrix} c_1^{A_i} & b_1^{A_i} & 0 & 0 & \cdots \\ a_2^{A_i} & c_2^{A_i} & b_2^{A_i} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & a_{N_2-2}^{A_i} & b_{N_2-2}^{A_i} & c_{N_2-2}^{A_i} \\ \cdots & 0 & 0 & a_{N_2-1}^{A_i} & c_{N_2-1}^{A_i} \end{pmatrix}, \qquad i = \overline{1, N_1}$$

$$B_i = \begin{pmatrix} c_1^{B_i} & b_1^{B_i} & 0 & 0 & \cdots \\ a_2^{B_i} & c_2^{B_i} & b_2^{B_i} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & a_{N_2-2}^{B_i} & b_{N_2-2}^{B_i} & c_{N_2-2}^{B_i} \\ \cdots & 0 & 0 & a_{N_2-1}^{B_i} & c_{N_2-1}^{B_i} \end{pmatrix}, \qquad i = \overline{0, N_1 - 1}$$

Таким образом, матрица системы имеет порядок $(N_1 + 1)(N_2 - 1)$, т.е. имеет размеры $\big((N_1 + 1)(N_2 - 1)\big) \times \big((N_1 + 1)(N_2 - 1)\big)$. Также видно, что матрица имеет блочно-трехдиагональную структуру, где на главной диагонали – $N_1 + 1$ трехдиагональных матриц-блоков размером $(N_2 - 1) \times (N_2 - 1)$, а на поддиагонали и на наддиагонали – $N_1$ трехдиагональных матриц-блоков размером $(N_2 - 1) \times (N_2 - 1)$.

Тогда, можем построить схему, которая бы наглядно иллюстрировала схему матрицы. Рассмотрим случай $N_1 + 1 = 4,\ N_2 - 1 = 4$. В результате получим схему, где значком «×» отмечены ненулевые элементы:

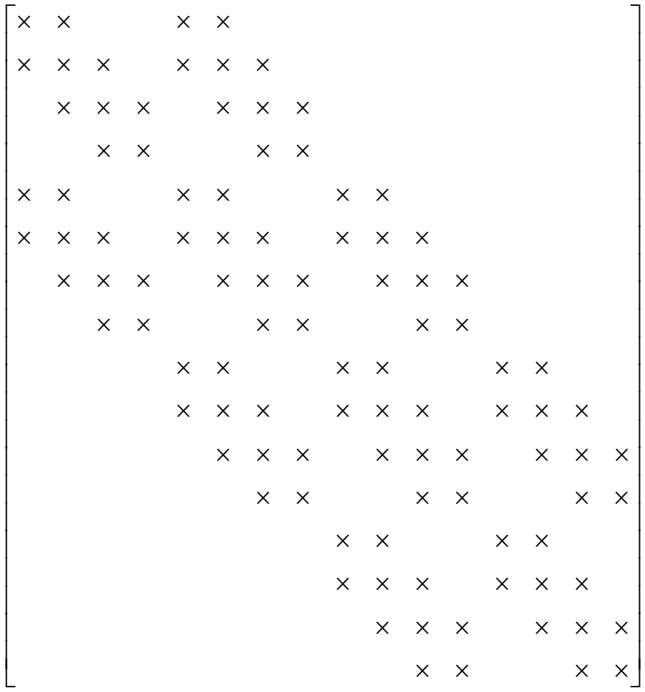

Рисунок 1.1 – Структура рассматриваемой матрицы (общий вид)

Ограничимся следующим частым для приложений частным случаем: $B_0, A_{N_1}$ – нулевые, а $C_0, C_{N_1}$ – единичные матрицы размеров $(N_2 - 1) \times (N_2 - 1)$. В этом случае, решения $Y_0$ и $Y_{N_1}$ первого и последнего блочных уравнений известны: $Y_0 = F_0, Y_{N_1} = F_{N_1}$.Тогда схема примет следующий вид:

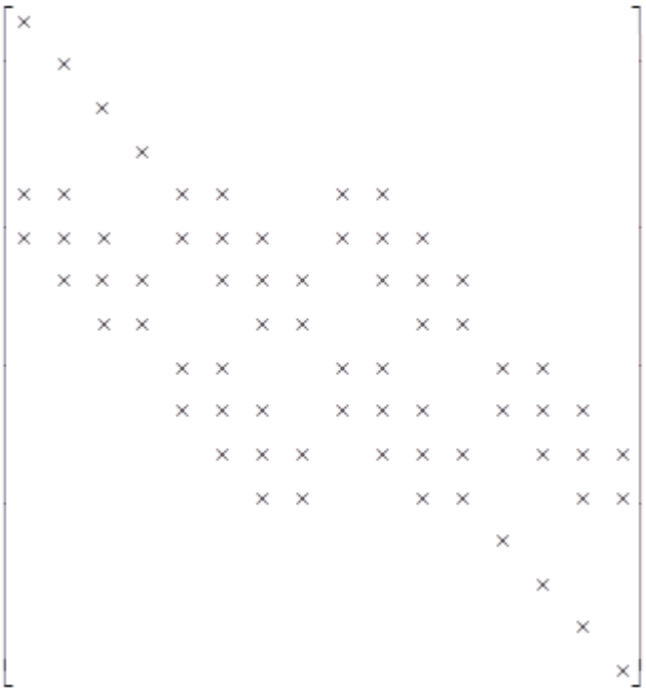

Рисунок 1.2 – Структура рассматриваемой матрицы (частный случай)

Блочные строки и столбцы имеют номера $0,1,\dots,N_1$. Вектор неизвестных:
$(Y_0, Y_1, \dots, Y_{N_1}) = (y_{0,1}, \dots, y_{0,N_2-1}, y_{1,1}, \dots, y_{1,N_2-1}, \dots, y_{N_1,1}, \dots, y_{N_1,N_2-1})$.

В дальнейшем нам потребуется знать расположение ненулевых коэффициентов матрицы системы (1.1) относительно неизвестных при $i = \overline{1, N_1 - 1}$. Укажем это расположение.

Рассмотрим в системе слагаемое $A_i Y_{i-1}$:

$$\begin{pmatrix} c_1^{A_i} & b_1^{A_i} & 0 & 0 & \cdots \\ a_2^{A_i} & c_2^{A_i} & b_2^{A_i} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & a_{N_2-2}^{A_i} & b_{N_2-2}^{A_i} & c_{N_2-2}^{A_i} \\ \cdots & 0 & 0 & a_{N_2-1}^{A_i} & c_{N_2-1}^{A_i} \end{pmatrix} \begin{pmatrix} y_{i-1,1} \\ y_{i-1,2} \\ \cdots \\ y_{i-1,N_2-2} \\ y_{i-1,N_2-1} \end{pmatrix}.$$

Здесь перед неизвестным $y_{i-1,j}$ расположен коэффициент $c_j^{A_i}$, перед неизвестным $y_{i-1,j-1}$ $(j \neq 1)$ расположен коэффициент $a_j^{A_i}$, а перед неизвестным $y_{i-1,j+1}$ $(j \neq N_2 - 1)$ расположен коэффициент $b_j^{A_i}$.

Рассмотрим слагаемое $C_i Y_i$:

$$\begin{pmatrix} c_1^{C_i} & b_1^{C_i} & 0 & 0 & \cdots \\ a_2^{C_i} & c_2^{C_i} & b_2^{C_i} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & a_{N_2-2}^{C_i} & b_{N_2-2}^{C_i} & c_{N_2-2}^{C_i} \\ \cdots & 0 & 0 & a_{N_2-1}^{C_i} & c_{N_2-1}^{C_i} \end{pmatrix} \begin{pmatrix} y_{i,1} \\ y_{i,2} \\ \cdots \\ y_{i,N_2-2} \\ y_{i,N_2-1} \end{pmatrix}$$

Здесь перед неизвестным $y_{i,j}$ расположен коэффициент $c_j^{C_i}$, перед неизвестным $y_{i,j-1}$ $(j \neq 1)$ расположен коэффициент $a_j^{C_i}$, а перед неизвестным $y_{i,j+1}$ $(j \neq N_2 - 1)$ расположен коэффициент $b_j^{C_i}$.

Рассмотрим слагаемое $B_i Y_{i+1}$:

$$\begin{pmatrix} c_1^{B_i} & b_1^{B_i} & 0 & 0 & \cdots \\ a_2^{B_i} & c_2^{B_i} & b_2^{B_i} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & a_{N_2-2}^{B_i} & b_{N_2-2}^{B_i} & c_{N_2-2}^{B_i} \\ \cdots & 0 & 0 & a_{N_2-1}^{B_i} & c_{N_2-1}^{B_i} \end{pmatrix} \begin{pmatrix} y_{i+1,1} \\ y_{i+1,2} \\ \cdots \\ y_{i+1,N_2-2} \\ y_{i+1,N_2-1} \end{pmatrix}$$

Здесь перед неизвестным $y_{i+1,j}$ расположен коэффициент $c_j^{B_i}$, перед неизвестным $y_{i+1,j-1}$ $(j \neq 1)$ расположен коэффициент $a_j^{B_i}$, а перед неизвестным $y_{i+1,j+1}$ $(j \neq N_2 - 1)$ расположен коэффициент $b_j^{B_i}$.

Таким образом строки матрицы системы (1.1) с номерами

$$i(N_2 - 1) + j, \qquad 1 \leq i \leq N_1 - 1, \qquad 2 \leq j \leq N_2 - 2,$$

имеют 9 ненулевых коэффициентов, которые стоят перед неизвестными

$y_{i-1,j-1}, y_{i-1,j}, y_{i-1,j+1}, \quad y_{i,j-1}, y_{i,j}, y_{i,j+1}, \quad y_{i+1,j-1}, y_{i+1,j}, y_{i+1,j+1}.$

Строки матрицы системы (1.1) с номерами

$$i(N_2 - 1) + 1, \quad 1 \le i \le N_1 - 1,$$

$j = 1$ – первое уравнение внутренней блочной строки, имеют 6 ненулевых коэффициентов, которые стоят перед неизвестными

$$y_{i-1,j}, y_{i-1,j+1}, \quad y_{i,j}, y_{i,j+1}, \quad y_{i+1,j}, y_{i+1,j+1}.$$

Строки матрицы системы (1.1) с номерами

$$i(N_2 - 1) + N_2 - 1, \quad 1 \le i \le N_1 - 1,$$

$j = N_2 - 1$ – последнее уравнение внутренней блочной строки, имеют 6 ненулевых коэффициентов, которые стоят перед неизвестными

$$y_{i-1,j-1}, y_{i-1,j}, \quad y_{i,j-1}, y_{i,j}, \quad y_{i+1,j-1}, y_{i+1,j}.$$

## 2 Блочный алгоритм Гаусса-Зейделя в неявной форме для систем специального вида

Пусть дана система линейных алгебраических уравнений вида $Ax = b$, где $A$ – матрица порядка $N$, векторы $b$ и $x$ являются $N$-мерными векторами. Можем выбрать некоторое число $r$, причем $r < N$ и $n = \frac{N}{r}$ – целое число. Тогда можем разбить матрицу $A$ на квадратные блоки размером $r \times r$, а вектор неизвестных и вектор правой части разобьем на блоки-векторы размерности $r$.

В свою очередь, блочный метод Гаусса-Зейделя решения системы $Ax = b$ имеет вид:

$$X_i^{k+1} = A_{i,i}^{-1}\left(B_i - \sum_{j=1}^{i-1} A_{i,j}X_j^{k+1} - \sum_{j=i+1}^{n} A_{i,j}X_j^k\right), \quad i = \overline{1,n},\ k = 0,1,2,\ldots$$

Получается, что каждая координата $(k+1)$-го приближения сразу после получения используется для вычисления (уточнения) следующих координат.

Тогда можем выписать блочный неявный метод Гаусса-Зейделя решения системы $Ax = b$:

$$A_{i,i}X_i^{k+1} = B_i - \sum_{j=1}^{i-1} A_{i,j}X_j^{k+1} - \sum_{j=i+1}^{n} A_{i,j}X_j^k, \quad i = \overline{1,n},\ k = 0,1,2,\ldots$$

Рассмотрим блочный неявный метод Гаусса-Зейделя решения систем вида:

$$\begin{cases} C_0Y_0^{k+1} = F_0 - B_0Y_1^k, \\ C_iY_i^{k+1} = F_i - A_iY_{i-1}^{k+1} - B_iY_{i+1}^k, \quad i = \overline{1,N_1-1}, \\ C_{N_1}Y_{N_1}^{k+1} = F_{N_1} - A_{N_1}Y_{N_1-1}^{k+1}, \\ k = 0,1,2,\ldots \end{cases}$$

Здесь $Y_0^0, Y_1^0, \ldots, Y_{N_1}^0$ – векторы начального приближения.

Пусть требуется выполнить $K$ итераций метода. Для каждого фиксированного итерационного шага $k$ уточняется каждая из $N_1 + 1$ компонент

приближенного вектора решения системы $(Y_0, Y_1, \dots, Y_{N_1})$, причем компонента $Y_i$, которая уточнена для очередного $i$, используется для пересчета следующих компонент. Запишем псевдокод, в котором при реализации блочного метода Гаусса-Зейделя используется только один массив неизвестных:

$Y_0 = F_0, Y_{N_1} = F_{N_1}$
do $i = \overline{1, N_1 - 1}$
  $Y_i = Y_i^0$ // выбирается начально приближение
enddo
do $k = \overline{0, K - 1}$
  do $i = \overline{1, N_1 - 1}$
    $\Phi = F_i - A_i Y_{i-1}^{k+1} - B_i Y_{i+1}^k$ // Φ – временный массив
    Решить СЛАУ $C_i Y_i^{k+1} = \Phi$
  enddo
enddo(k)

При программировании существенным является нижний индекс, верхний индекс оставлен для наглядности. Тогда имеем:

$Y_0 = F_0, Y_{N_1} = F_{N_1}$
do $i = \overline{1, N_1 - 1}$
  $Y_i = Y_i^0$ // выбирается начально приближение
enddo
do $k = \overline{0, K - 1}$
  do $i = \overline{1, N_1 - 1}$
    $\Phi = F_i - A_i Y_{i-1} - B_i Y_{i+1}$ // Φ – временный массив
    Решить СЛАУ $C_i Y_i = \Phi$
  enddo
enddo(k)

Возникающие на каждом $k$-ом шаге СЛАУ с трехдиагональными матрицами порядка $N_2 - 1$ можно решить методом прогонки [2, 3].

Теория метода прогонки (правой) для решения системы с матрицей вида

$$\begin{pmatrix} c_1 & b_1 & 0 & 0 & \cdots \\ a_2 & c_2 & b_2 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & a_{N_2-2} & c_{N_2-2} & b_{N_2-2} \\ \cdots & 0 & 0 & a_{N_2-1} & c_{N_2-1} \end{pmatrix},$$

вектором неизвестных $(y_1, y_2, \dots, y_{N_2-1})$ и вектором правой части $(\varphi_1, \varphi_2, \dots, \varphi_{N_2-1})$ следующее:

Прямая прогонка – вычисление прогоночных коэффициентов по формулам

$$\alpha_2 = -\frac{b_1}{c_1}, \qquad \beta_2 = \frac{\varphi_1}{c_1},$$

$$\alpha_{j+1} = -\frac{b_j}{c_j + \alpha_j a_j}, \qquad \beta_{j+1} = \frac{\varphi_j - a_j \beta_j}{c_j + \alpha_j a_j}, \qquad j = \overline{2, N_2 - 2},$$

$$\beta_{N_2} = \frac{\varphi_{N_2-1} - a_{N_2-1}\beta_{N_2-1}}{c_{N_2-1} + \alpha_{N_2-1} a_{N_2-1}}.$$

После вычисления прогоночных коэффициентов, используется обратная прогонка – вычисление решения по формулам

$$y_{N_2-1} = \beta_{N_2}, \qquad y_j = \alpha_{j+1} y_{j+1} + \beta_{j+1}, \qquad j = \overline{N_2 - 2, 1}$$

Алгоритм 1. Тогда можем выписать полноценный алгоритм, который считает заданное число итераций блочного неявного метода Гаусса-Зейделя решения системы (1.1).

Входные данные: двумерные массивы

$cC(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 1$ // $cC(i,j) = c_j^{C_i}$

$aC(i,j),\ 1 \le i \le N_1 - 1,\ 2 \le j \le N_2 - 1$ // $aC(i,j) = a_j^{C_i}$

$bC(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 2$ // $bC(i,j) = b_j^{C_i}$

$cA(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 1$ // $cA(i,j) = c_j^{A_i}$

$aA(i,j),\ 1 \le i \le N_1 - 1,\ 2 \le j \le N_2 - 1$ // $aA(i,j) = a_j^{A_i}$

$bA(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 2$ // $bA(i,j) = b_j^{A_i}$

$cB(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 1$ // $cB(i,j) = c_j^{B_i}$

$aB(i,j),\ 1 \le i \le N_1 - 1,\ 2 \le j \le N_2 - 1$ // $aB(i,j) = a_j^{B_i}$

$bB(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 2$ // $bB(i,j) = b_j^{B_i}$

$fslae(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 1$ // элементы векторов $F_i$ из правой части системы (1.1)

Выходные данные: двумерный массив

$y(i,j),\ 0 \le i \le N_1,\ 1 \le j \le N_2 - 1$ // приближенное решение $(Y_0, Y_1, \dots, Y_{N_1})$ системы (1.1) $Y_i = (y_{i,1}, \dots, y_{i,N_2-1})$

Псевдокод:

$Y_0 = F_0, Y_{N_1} = F_{N_1}$

do $i = \overline{1, N_1 - 1}$

$Y_i = Y_i^0$ // выбирается начальное приближение

enddo

do $k = \overline{0, K-1}$

do $i = \overline{1, N_1 - 1}$

do $j = \overline{1, N_2 - 1}$ // формируется вектор $\Phi = F_i - A_i Y_{i-1} - B_i Y_{i+1}$

if $j = 1$ then $phi(1) = fslae(i,1) - cA(i,1)y(i-1,1) -$

$-bA(i,1)y(i-1,2) - cB(i,1)y(i+1,1) - bB(i,1)y(i+1,2)$

// $\varphi_1$ – первый элемент вектора $F_i - A_i Y_{i-1} - B_i Y_{i+1}$

if $1 < j < N_2 - 1$ then $phi(j) = fslae(i,j) - aA(i,j)y(i-1,j-1) -$

$-cA(i,j)y(i-1,j) - bA(i,j)y(i-1,j+1) - aB(i,j) *$

$* y(i+1,j-1) - cB(i,j)y(i+1,j) - bB(i,j)y(i+1,j+1)$

// $\varphi_j$ – $j$-й $(j \neq 1, j \neq N_2 - 1)$ элемент вектора $F_i - A_i Y_{i-1} - B_i Y_{i+1}$

if $j = N_2 - 1$ then $phi(N_2 - 1) = fslae(i, N_2 - 1) - aA(i, N_2 - 1) *$

$* y(i-1, N_2 - 2) - cA(i, N_2 - 1)y(i-1, N_2 - 1) - aB(i, N_2 - 1) *$

$* y(i+1, N_2 - 2) - cB(i, N_2 - 1)y(i+1, N_2 - 1)$

// $\varphi_{N_2-1}$ – последний элемент вектора $F_i - A_i Y_{i-1} - B_i Y_{i+1}$

enddo(j)

// Начало решения СЛАУ $C_i Y_i = \Phi$ методом прогонки

$gamma = cC(i,1)$ // $\gamma = c_1$

$alpha(2) = -\frac{bC(i,1)}{gamma}$ // $\alpha_2 = -\frac{b_1}{c_1}$

$beta(2) = \frac{phi(1)}{gamma}$ // $\beta_2 = \frac{\varphi_1}{c_1}$

do $j = \overline{2, N_2 - 2}$

$gamma = cC(i,j) + aC(i,j)alpha(j)$ // $\gamma = c_j + a_j \alpha_j$

$alpha(j+1) = -\frac{bC(i,j)}{gamma}$ // $\alpha_{j+1} = -\frac{b_j}{c_j + a_j \alpha_j}$

$beta(j+1) = \frac{phi(j) - aC(i,j)beta(j)}{gamma}$ // $\beta_{j+1} = \frac{\varphi_j - a_j \beta_j}{c_j + a_j \alpha_j}$

enddo

$gamma = cC(i, N_2 - 1) + aC(i, N_2 - 1)alpha(N_2 - 1)$

$beta(N_2) = \frac{phi(N_2-1) - aC(i,N_2-1)beta(N_2-1)}{gamma}$ // $\beta_{N_2} = \frac{\varphi_{N_2-1} - a_{N_2-1}\beta_{N_2-1}}{c_{N_2-1} + a_{N_2-1}\alpha_{N_2-1}}$

$y(i, N_2 - 1) = beta(N_2)$ // $y_{N_2-1} = \beta_{N_2}$

do $j = \overline{N_2 - 2, 1}$ // цикл с шагом -1

$y(i,j) = alpha(j+1)y(i,j+1) + beta(j+1)$

// $y_j = \alpha_{j+1} y_{j+1} + \beta_{j+1}$

enddo

// конец решения СЛАУ $C_i Y_i = \Phi$ методом прогонки

enddo(i)

enddo(k)

Вычислительная сложность этого алгоритма, который выполняет $K$ итераций блочного неявного метода Гаусса-Зейделя решения системы (1.1): примерно $KN_1(6N_2 + 4N_2 + N_2) = 11KN_1N_2$ операций.

## 3 Основные положения для распараллеливания алгоритмов при программировании

Для практической реализации рассматриваемого метода на современных вычислительных системах, особенно при больших размерах задачи, актуализируется вопрос ускорения вычислений за счет параллельной обработки данных. Одним из наиболее популярных способов распараллеливания, в рамках модели разделяемой памяти, является использование OpenMP (Open Multi-Processing).

Рассмотрим основные положения для распараллеливания рассматриваемого алгоритма при программировании.

### 3.1 Выделение параллельно обрабатываемых фрагментов

Исходный блочный метод Гаусса-Зейделя предполагает последовательное уточнение компонент вектора решения по индексу $i$, что накладывает естественные ограничения на уровень параллелизма при прямом применении. Однако, в рассматриваемых системах можно существенно расширить параллелизацию за счет разбиения вычислений по нескольким уровням индексов и операций:

1) Индекс $i$ (блочные компоненты вектора решения):
Изначально $i$-ый блок решения $Y_i^{k+1}$ вычисляется, используя уже обновленный $Y_{i-1}^{k+1}$, что создает каскадную зависимость. Тем не менее, можно использовать волновую параллелизацию (wavefront parallelization). Она предполагает, что при больших размерах задачи группы блоков могут быть вычислены параллельно, если уже известна достаточная часть требуемой информации от предыдущих индексов.
2) Индекс $j$ (внутренние операции для каждого блока):
Формирование вектора $\varphi = F_i - A_iY_{i-1} - B_iY_{i+1}$, являющегося промежуточным шагом вычисления $Y_i$, можно распараллелить по индексу $j$, поскольку каждое $\varphi_j$ вычисляется независимо.

### 3.2 Организация параллельных регионов и приватность данных

При применении OpenMP необходимо грамотно организовать параллельные области так, чтобы:

1) каждому потоку был предоставлен свой собственный временный массив $\varphi$, коэффициенты для параллельной прогонки или любые другие необходимые данные. Это обеспечивается благодаря использованию private или firstprivate переменных
2) основные массивы решения $Y$ и коэффициентов $A_i, B_i, C_i$ оставались в общей памяти. Причем, важно аккуратно настроить доступ: чтение – всегда, запись – только там, где обновленные данные не вступают в конфликт с другими потоками
3) использование параллельных конструкций минимизировало синхронизацию между потоками

### 3.3 Эффективная организация данных

Для увеличения пропускной способности довольно важно оптимизировать структуру данных. Расположение элементов в памяти по основному направлению параллелизации позволяет одновременно задействовать несколько потоков без лишних обращений к памяти.

Также, хранение коэффициентов трехдиагональных матриц в формате, удобном для прогонки. Например, можем выделить отдельный массив для главной, верхней и нижней диагоналей. Это упростит доступ к памяти и повысит эффективность кэширования.

## 4 Параллельный алгоритм Гаусса-Зейделя для численного решения систем специального вида, используя OpenMP

Рассмотрим блочный неявный параллельный метод Гаусса-Зейделя решения систем вида:

$$\begin{cases} C_0 Y_0^{k+1} = F_0 - B_0 Y_1^k, \\ C_i Y_i^{k+1} = F_i - A_i Y_{i-1}^{k+1} - B_i Y_{i+1}^k, \qquad i = \overline{1, N_1 - 1}, \\ C_{N_1} Y_{N_1}^{k+1} = F_{N_1} - A_{N_1} Y_{N_1-1}^{k+1}, \\ k = 0,1,2, \dots \end{cases}$$

Здесь $Y_0^0, Y_1^0, \dots, Y_{N_1}^0$ – векторы начального приближения.

Рассмотрим параллельный алгоритм более подробно:

Формирование временного вектора $\Phi = F_i - A_i Y_{i-1} - B_i Y_{i+1}$:

Для фиксированного $i$ и $k$ вычисление $\phi_j \; \forall \, j = \overline{1, N_2 - 1}$ возможно параллельно по $j$, так как каждое $\phi_j$ не зависит от других. В связи с этим, будем использовать #pragma omp parallel for для вычисления элементов вектора Φ параллельно [4].

Запишем краткий псевдокод параллельной версии алгоритма реализации блочного метода Гаусса-Зейделя:

$Y_0 = F_0, Y_{N_1} = F_{N_1}$
do $i = \overline{1, N_1 - 1}$
  do $j = \overline{0, N_2 - 2}$
    $Y_i(j) = 0$

Создать $Y_{temp}$ размером $(N\_1 + 1) \times (N_2 - 1)$
do $k = \overline{0, K - 1}$
  $Y_{old} = Y$
  #pragma omp parallel
    $phi(N_2 - 1), Y_i(N_2 - 1)$ // локальные для потока
    #pragma omp for
    do $i = \overline{1, N_1 - 1}$
      do $j = \overline{1, N_2 - 1}$
        $val = fslae(i, j)$
        Используя $Y\_old(i - 1,*), Y\_old(i + 1,*), cA, aA, bA, cB, aB, bB$
        вычислить $val = fslae(i, j) - A_i Y_{i-1} - B_i Y_{i+1}$
        $phi(j) = val$

      solveTridiagonalSystem(i-1, cC,aC,bC, phi, Y_i, N_2)
      Записать $Y_i$ в $Y_{temp}(i)$

  do $i = \overline{1, N_1 - 1}$
    do $j = \overline{0, N_2 - 2}$
      $Y(i, j) = Y_{temp}(i, j)$
endfor(k)

Возникающие на каждом шаге $k$ системы линейных алгебраических уравнений с трехдиагональными матрицами порядка $N_2 - 1$ можно решить методом прогонки.

Алгоритм 2. Тогда можем выписать полноценный алгоритм, который считает заданное число итераций блочного неявного параллельного метода Гаусса-Зейделя решения системы (1.1).

Входные данные: двумерные массивы

$cC(i, j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 1$ // $cC(i, j) = c_j^{C_i}$

$aC(i, j),\ 1 \le i \le N_1 - 1,\ 2 \le j \le N_2 - 1$ // $aC(i, j) = a_j^{C_i}$

$bC(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 2$ // $bC(i,j) = b_j^{C_i}$

$cA(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 1$ // $cA(i,j) = c_j^{A_i}$

$aA(i,j),\ 1 \le i \le N_1 - 1,\ 2 \le j \le N_2 - 1$ // $aA(i,j) = a_j^{A_i}$

$bA(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 2$ // $bA(i,j) = b_j^{A_i}$

$cB(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 1$ // $cB(i,j) = c_j^{B_i}$

$aB(i,j),\ 1 \le i \le N_1 - 1,\ 2 \le j \le N_2 - 1$ // $aB(i,j) = a_j^{B_i}$

$bB(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 2$ // $bB(i,j) = b_j^{B_i}$

$fslae(i,j),\ 1 \le i \le N_1 - 1,\ 1 \le j \le N_2 - 1$ // элементы векторов $F_i$ из правой части системы (1.1)

Выходные данные: двумерный массив

$y(i,j),\ 0 \le i \le N_1,\ 1 \le j \le N_2 - 1$ // приближенное решение $\left(Y_0, Y_1, \dots, Y_{N_1}\right)$ системы (1.1) $Y_i = \left(y_{i,1}, \dots, y_{i,N_2-1}\right)$

Псевдокод алгоритма:

// Инициализация Y:

$Y(0,j) = F(j), j = \overline{0, N_2 - 2}$

$Y(N_1,j) = F(N_1 * (N_2 - 1) + j), j = \overline{0, N\_2 - 2}$

do $i = \overline{1, N_1 - 1}$

  do $j = \overline{0, N_2 - 2}$

    $Y(i,j) = 0$

$Y_{temp}$ размер $(N_1 + 1) \times (N_2 - 1)$, заполнен нулями

do $k_{iter} = \overline{0, K - 1}$

  $Y_{old} = Y$

  // Параллельная область

  #pragma omp parallel default(none) shared(N_1,N_2, N_2-1,Y_temp,y,Y_old,fslae,F,cC,aC,bC,cA,aA,bA,cB,aB,bB)

  {

    // Локальные массивы для каждого потока:

    массив $phi$ размер $N_2 - 1$, заполнен 0

    массив $Y_i$ размер $N_2 - 1$, заполнен 0

    переменные val(double), jidx(int)

    // Параллелим цикл по i

    #pragma omp for

do $i = \overline{1, N\_1 - 1}$

// Формируем $\varphi(i)$

do $j = \overline{1, N\_2 - 1}$

$val = fslae(i-1, j-1)$

$jidx = j - 1$

// Читаем $Y_{old}(i-1,*)$ , $Y_{old}(i+1,*)$

y_im1_j = $Y_{old}(i-1, jidx)$

y_im1_jm1 = если $j > 1$ то $Y_{old}(i-1, jidx-1)$ иначе 0

y_im1_jp1 = если $j < m$ то $Y_{old}(i-1, jidx+1)$ иначе 0

y_ip1_j = $Y_{old}(i+1, jidx)$

y_ip1_jm1 = если $j > 1$ то $Y_{old}(i+1, jidx-1)$ иначе 0

y_ip1_jp1 = если $j < m$ то $Y_{old}(i+1, jidx+1)$ иначе 0

// val -= $A_i Y_{i-1} + B_i Y_{i+1}$ с учетом aA,bA,cA,aB,bB,cB

val = val - cA(i-1,j-1)*y_im1_j

если $j \geq 2$ val -= aA(i-1,j-1)*y_im1_jm1

если $j \leq m - 1$ val -= bA(i-1,j-1)*y_im1_jp1

val = val - cB(i-1,j-1)*y_ip1_j

если $j \geq 2$ val -= aB(i-1,j-1)*y_ip1_jm1

если $j \leq m - 1$ val -= bB(i-1,j-1)*y_ip1_jp1

phi(j-1)=val

endfor(j)

// Решаем трехдиагональную систему для i-й строки

solveTridiagonalSystem(i-1,cC,aC,bC,phi,Y_i,N_2)

// Записываем результат в $Y_{temp}(i)$

do $jj = \overline{0, N_2 - 2}$:

$Y_{temp}(i, jj) = Y_i(jj)$

endfor(i)

} // end parallel

// Обновляем $Y$ из $Y_{temp}$

do $i = \overline{1, N_1 - 1}$

do $j = \overline{0, N_2 - 2}$

$Y(i,j) = Y_{temp}(i,j)$

enddo(k)

Вычислительная сложность этого алгоритма останется такой же, как и в последовательной реализации: примерно $KN_1(6N_2 + 4N_2 + N_2) = 11KN_1N_2$ операций. Но, в это же время, скорость выполнения сокращается.

## 5 Вычислительные эксперименты

При проведении вычислительных экспериментов будем выполнять заданное число итераций (не будем достигать критерия остановки), а в качестве вектора $Y^0$ будем брать нулевой вектор.

Рассматривается случай, когда $B_0, A_{N_1}$ – нулевые, а $C_0, C_{N_1}$ – единичные матрицы размеров $(N_2 - 1) \times (N_2 - 1)$. В этом случае, решения $Y_0, Y_{N_1}$ первого и последнего блочных уравнений известны заранее: $Y_0 = F_0, Y_{N_1} = F_{N_1}$.

Элементы остальных матриц-блоков системы можно задать так, чтобы получилось строгое диагональное преобладание.

Например:

все диагональные элементы диагональных матриц-блоков равны 8:

$$cC(i,j) = 8, \qquad 1 \le i \le N_1 - 1, \qquad 1 \le j \le N_2 - 1;$$

другие элементы:

$$aC(i,j) = \frac{2i + j}{2N_1 + N_2}, \qquad 1 \le i \le N_1 - 1, \qquad 2 \le j \le N_2 - 1;$$

$$bC(i,j) = \frac{2i + j}{2N_1 + N_2}, \qquad 1 \le i \le N_1 - 1, \qquad 1 \le j \le N_2 - 2;$$

$$cA(i,j) = \frac{2i + j}{2N_1 + N_2}, \qquad 1 \le i \le N_1 - 1, \qquad 1 \le j \le N_2 - 1;$$

$$aA(i,j) = \frac{2i + j}{2N_1 + N_2}, \qquad 1 \le i \le N_1 - 1, \qquad 2 \le j \le N_2 - 1;$$

$$bA(i,j) = \frac{2i + j}{2N_1 + N_2}, \qquad 1 \le i \le N_1 - 1, \qquad 1 \le j \le N_2 - 2;$$

$$cB(i,j) = \frac{2i + j}{2N_1 + N_2}, \qquad 1 \le i \le N_1 - 1, \qquad 1 \le j \le N_2 - 1;$$

$$aB(i,j) = \frac{2i + j}{2N_1 + N_2}, \qquad 1 \le i \le N_1 - 1, \qquad 2 \le j \le N_2 - 1;$$

$$bB(i,j) = \frac{2i + j}{2N_1 + N_2}, \qquad 1 \le i \le N_1 - 1, \qquad 1 \le j \le N_2 - 2.$$

Будем считать, что решение системы такое:

$$\left(y_{0,1}, \dots, y_{0,N_2-1}, y_{1,1}, \dots, y_{1,N_2-1}, \dots, y_{N_1,1}, \dots, y_{N_1,N_2-1}\right) = (1,1, \dots, 1).$$

Тогда правая часть системы – произведение матрицы системы и вектора $(1,1,\dots,1)$.

Реализуем последовательный и параллельный метод Гаусса-Зейделя для системы описанного выше вида. Код программы приведен в приложении А, Б.

Рассмотрим случай небольшой системы, т.е. $N_1, N_2 = 9, 11$ соответственно. Это значит, что матрица будет содержать $N_1 + 1 = 10$ блоков в строке и столбце, и $N_2 - 1 = 10$ строк и столбцов в блоке. Таким образом, матрица будет состоять из $10^4$ элементов.

| $K$ | Последовательный метод, сек | Параллельный метод, сек | Ускорение (последовательный к параллельному) |
|---|---|---|---|
| 2 | 1.98e-05 | 0.0010374 | 0.0190 |
| 4 | 3.5e-05 | 0.0017151 | 0.0204 |
| 8 | 0.0001395 | 0.0027391 | 0.0509 |
| 16 | 0.0002502 | 0.003391 | 0.0737 |
| 32 | 0.0002583 | 0.0058632 | 0.0440 |
| 64 | 0.0007941 | 0.0110069 | 0.0721 |

Таблица 5.1 – Время выполнения программы при фиксированном $N_1, N_2 = 9, 11$ соответственно.

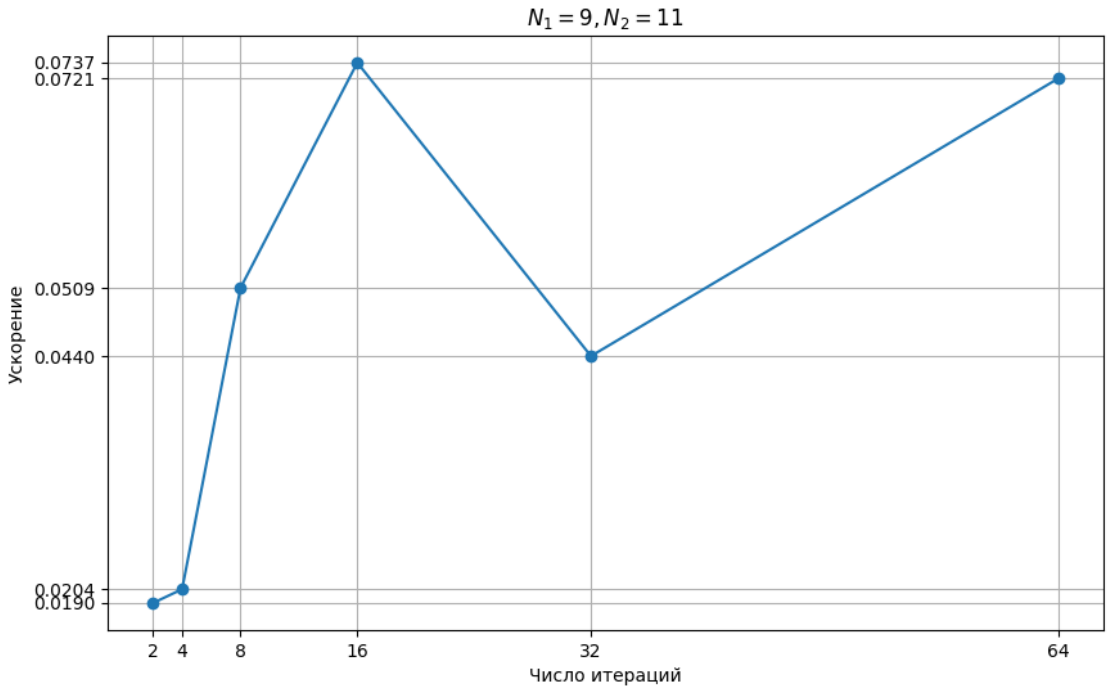


Рисунок 5.1 – График зависимости ускорения от числа итераций при $N_1, N_2 = 9, 11$ соответственно

Рассмотрим случай средней по размерам системы, т.е. $N_1, N_2 = 99, 101$ соответственно. Это значит, что матрица будет содержать $N_1 + 1 = 100$ блоков в строке и столбце, и $N_2 - 1 = 100$ строк и столбцов в блоке. Таким образом, матрица будет состоять из $10^8$ элементов.

| $K$ | Последовательный метод, сек | Параллельный метод, сек | Ускорение (последовательный к параллельному) |
|---|---|---|---|
| 2 | 0.0028022 | 0.0021766 | 1.2874 |
| 4 | 0.005659 | 0.0036886 | 1.5341 |
| 8 | 0.0074405 | 0.0042403 | 1.7547 |
| 16 | 0.0135971 | 0.0073413 | 1.8521 |
| 32 | 0.0421885 | 0.0165692 | 2.5462 |
| 64 | 0.0889856 | 0.0338153 | 2.6315 |

Таблица 5.2 – Время выполнения программы при фиксированном $N_1, N_2 = 99, 101$ соответственно.

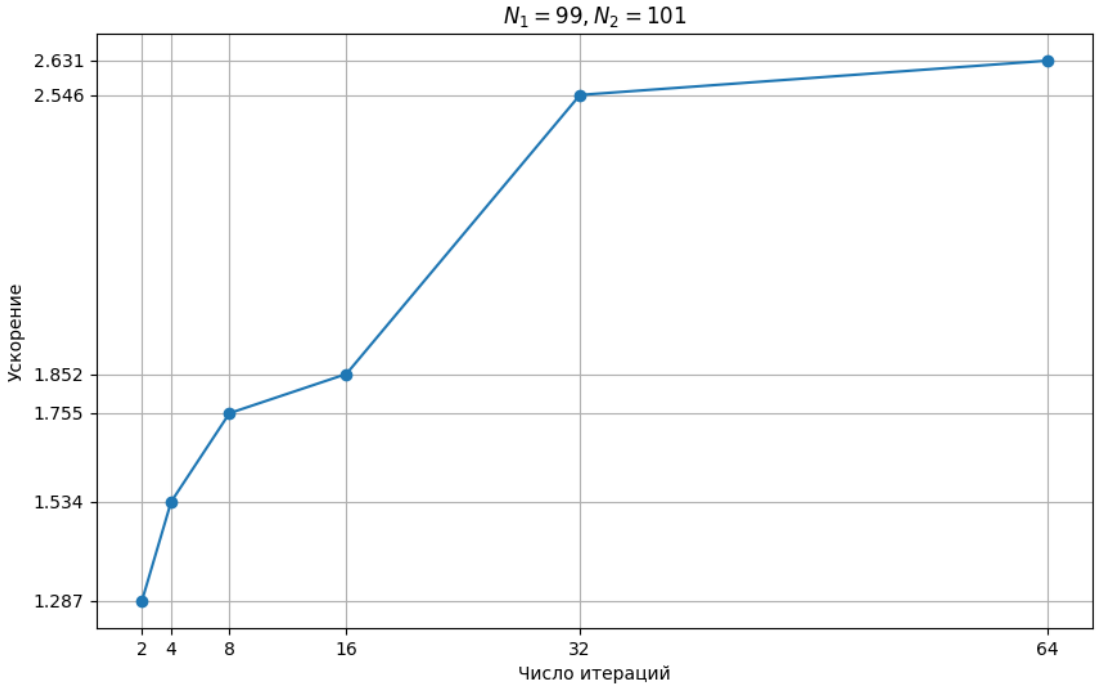


Рисунок 5.2 – График зависимости ускорения от числа итераций при $N_1, N_2 = 99, 101$ соответственно

Рассмотрим случай огромной по размерам системы, т.е. $N_1, N_2 = 999, 1001$ соответственно. Это значит, что матрица будет содержать $N_1 + 1 = 1000$ блоков в строке и столбце, и $N_2 - 1 = 1000$ строк и столбцов в блоке. Таким образом, матрица будет состоять из $10^{12}$ элементов.

| $K$ | Последовательный метод, сек | Параллельный метод, сек | Ускорение (последовательный к параллельному) |
|---|---|---|---|
| 2 | 0.19471 | 0.0744198 | 2.6163 |
| 4 | 0.362488 | 0.128353 | 2.8241 |
| 8 | 0.747323 | 0.241043 | 3.1003 |
| 16 | 1.47704 | 0.457367 | 3.2294 |
| 32 | 4.20664 | 1.03025 | 4.0831 |
| 64 | 8.65762 | 1.88462 | 4.5938 |

Таблица 5.3 – Время выполнения программы при фиксированном $N_1, N_2 = 999, 1001$ соответственно.

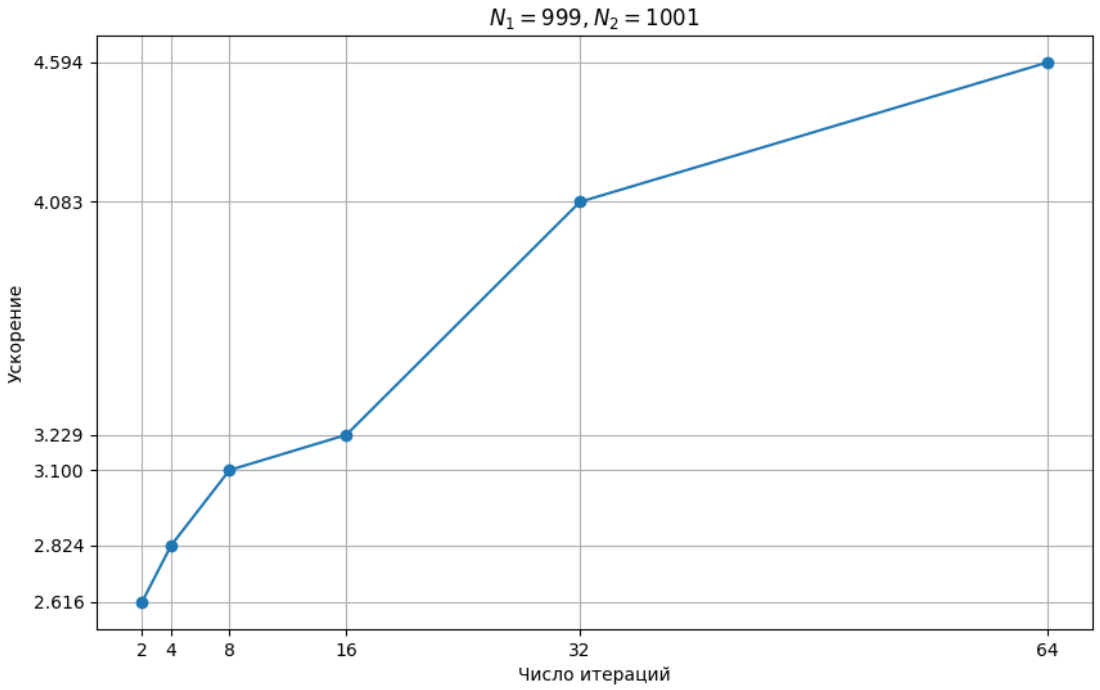


Рисунок 5.3 – График зависимости ускорения от числа итераций при $N_1, N_2 = 999, 1001$ соответственно

Сделаем выводы о полученных результатах.

Если проанализировать полученные в таблице 5.1 значения, можно явно заметить, что для маленьких систем последовательный метод работает намного быстрее, т.е. реализация и использование параллельного метода не оправданы.

После анализа результатов таблицы 5.2 и 5.3, наглядно видно, что для матрицы таких размерностей появляется смысл использовать параллельный метод. При малом числе итераций ускорение не такое весомое, но с увеличением $K$ мы получаем более серьезную производительность.

# ЗАКЛЮЧЕНИЕ

В данной курсовой работе была рассмотрена последовательная и параллельная реализация метода Гаусса-Зейделя с использованием технологии OpenMP для решения блочно-трехдиагональных систем с трехдиагональными блоками. Работа включала в себя разработку псевдокодов последовательного и параллельного алгоритмов, а также их реализация и сравнительный анализ производительности.

В процессе исследования были достигнуты следующие результаты:

- Разработан и описан псевдокодом последовательный и параллельный вариант метода Гаусса-Зейделя для систем специального вида
- Реализованы программные решения на основе технологии OpenMP, позволяющие эффективно использовать возможности многоядерных процессоров для параллельных вычислений
- Проведено сравнение производительности последовательного и параллельного алгоритмов. Анализ показал, что параллельная версия метода позволяет существенно сократить время вычислений при решении крупных систем уравнений специального вида.

Полученные результаты подтверждают актуальность и целесообразность применения параллельных технологий для ускорения решения задач численной математики.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Yang W, Li K, Li K. A parallel solving method for block-tridiagonal equations on CPU-GPU heterogeneous computing systems // Supercomputing. 2017. Vol. 73, No 5. P. 1760-1781.

2. Самарский А.А. Николаев Е.С. Методы решения сеточных уравнений. М: Наука. 1978. 592 с.

3. [Электрон. ресурс – https://edufpmi.bsu.by/ Численные методы (5 семестр, 1 поток) / Лекции], лекции «Метод прогонки».

4. [Электрон. ресурс – https://edufpmi.bsu.by/ Параллельные и распределенные вычисления / Лекции], лекции «OpenMP»

# ПРИЛОЖЕНИЯ

## ПРИЛОЖЕНИЕ А

## Код последовательного и параллельного метода Гаусса-Зейделя для систем специального вида на языке C++ с использованием OpenMP

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <cmath>
#include "omp.h"
#include <cassert>
#include <chrono>

void solveTridiagonalSystem(int i,
                        const std::vector<std::vector<double>> &cC,
                        const std::vector<std::vector<double>> &aC,
                        const std::vector<std::vector<double>> &bC,
                        std::vector<double> &phi,
                        std::vector<double> &Y_i,
                        int N_2)
{
   int m = N_2 - 1;
   std::vector<double> alpha(m+1,0.0);
   std::vector<double> beta(m+1,0.0);

   // Инициализация первых коэффициентов прогонки
   double gamma = cC[i][0];
   alpha[1] = -bC[i][0]/gamma;
   beta[1]  = phi[0]/gamma;

   // Прямая прогонка для j=1..m-2
   for (int j=1; j<=m-2; j++) {
      gamma = cC[i][j] + aC[i][j]*alpha[j];
      alpha[j+1] = -bC[i][j]/gamma;
      beta[j+1]  = (phi[j]-aC[i][j]*beta[j])/gamma;
   }

   // Последний элемент
   gamma = cC[i][m-1] + aC[i][m-1]*alpha[m-1];
```

```
    beta[m] = (phi[m-1]-aC[i][m-1]*beta[m-1])/gamma;

    // Обратная прогонка
    Y_i[m-1] = beta[m];
    for (int j=m-2; j>=0; j--) {
        Y_i[j] = alpha[j+1]*Y_i[j+1] + beta[j+1];
    }
}

void initializeSystem(int N_1, int N_2,
                      std::vector<std::vector<double>> &cC,
                      std::vector<std::vector<double>> &aC,
                      std::vector<std::vector<double>> &bC,
                      std::vector<std::vector<double>> &cA,
                      std::vector<std::vector<double>> &aA,
                      std::vector<std::vector<double>> &bA,
                      std::vector<std::vector<double>> &cB,
                      std::vector<std::vector<double>> &aB,
                      std::vector<std::vector<double>> &bB,
                      std::vector<std::vector<double>> &fslae,
                      std::vector<double> &F)
{
    // Размер для внутренних массивов: (N_1-1)x(N_2-1)
    cC.assign(N_1-1,std::vector<double>(N_2-1,0.0));
    aC.assign(N_1-1,std::vector<double>(N_2-1,0.0));
    bC.assign(N_1-1,std::vector<double>(N_2-1,0.0));

    cA.assign(N_1-1,std::vector<double>(N_2-1,0.0));
    aA.assign(N_1-1,std::vector<double>(N_2-1,0.0));
    bA.assign(N_1-1,std::vector<double>(N_2-1,0.0));

    cB.assign(N_1-1,std::vector<double>(N_2-1,0.0));
    aB.assign(N_1-1,std::vector<double>(N_2-1,0.0));
    bB.assign(N_1-1,std::vector<double>(N_2-1,0.0));

    fslae.assign(N_1-1,std::vector<double>(N_2-1,0.0));

    double denom = 2.0*N_1 + N_2;
    for (int i=1; i<=N_1-1; i++) {
        for (int j=1; j<=N_2-1; j++) {
```

```
        // C_i
        cC[i-1][j-1] = 8.0;
        if (j>=2) aC[i-1][j-1]=(2.0*i+j)/denom;
        if (j<=N_2-2) bC[i-1][j-1]=(2.0*i+j)/denom;

        // A_i
        cA[i-1][j-1]=(2.0*i+j)/denom;
        if (j>=2) aA[i-1][j-1]=(2.0*i+j)/denom;
        if (j<=N_2-2) bA[i-1][j-1]=(2.0*i+j)/denom;

        // B_i
        cB[i-1][j-1]=(2.0*i+j)/denom;
        if (j>=2) aB[i-1][j-1]=(2.0*i+j)/denom;
        if (j<=N_2-2) bB[i-1][j-1]=(2.0*i+j)/denom;
    }
}

// Формируем F для решения (1,...,1)
int N = (N_1+1)*(N_2-1);
F.assign(N,0.0);

// F_0=(1,...,1)
for (int j=1; j<=N_2-1; j++) {
    F[(0)*(N_2-1)+(j-1)] = 1.0;
}

// F_(N_1)=(1,...,1)
for (int j=1; j<=N_2-1; j++) {
    F[(N_1)*(N_2-1)+(j-1)] = 1.0;
}

// Внутренние F_i
for (int i=1; i<=N_1-1; i++) {
    for (int j=1; j<=N_2-1; j++) {
        double val=0.0;
        // A_i
        val+= cA[i-1][j-1];
        if (j>=2) val+=aA[i-1][j-1];
        if (j<=N_2-2) val+=bA[i-1][j-1];
```

```
            // C_i
            val+= cC[i-1][j-1];
            if (j>=2) val+=aC[i-1][j-1];
            if (j<=N_2-2) val+=bC[i-1][j-1];

            // B_i
            val+= cB[i-1][j-1];
            if (j>=2) val+=aB[i-1][j-1];
            if (j<=N_2-2) val+=bB[i-1][j-1];

            // F_i
            F[i*(N_2-1)+(j-1)] = val;
        }
    }

    // fslae(i,j) = F_i(j) для i=1..N_1-1
    for (int i=1; i<=N_1-1; i++) {
        for (int j=1; j<=N_2-1; j++) {
            fslae[i-1][j-1] = F[i*(N_2-1)+(j-1)];
        }
    }
}

void gaussSeidel(int N_1, int N_2, int K,
                 const std::vector<std::vector<double>> &cC,
                 const std::vector<std::vector<double>> &aC,
                 const std::vector<std::vector<double>> &bC,
                 const std::vector<std::vector<double>> &cA,
                 const std::vector<std::vector<double>> &aA,
                 const std::vector<std::vector<double>> &bA,
                 const std::vector<std::vector<double>> &cB,
                 const std::vector<std::vector<double>> &aB,
                 const std::vector<std::vector<double>> &bB,
                 const std::vector<std::vector<double>> &fslae,
                 const std::vector<double> &F,
                 std::vector<std::vector<double>> &y)
{
    // Начальные условия
    // Y_0 = F_0, Y_(N_1)=F_(N_1)
    for (int j=0; j<N_2-1; j++) {
```

```
    y[0][j] = F[j];
    y[N_1][j] = F[N_1*(N_2-1)+j];
}

// Внутренние y(i,j)=0
for (int i=1; i<=N_1-1; i++) {
    for (int j=0; j<N_2-1; j++)
        y[i][j]=0.0;
}

std::vector<double> phi(N_2-1,0.0);
std::vector<double> Y_i(N_2-1,0.0);

for (int k_iter=0; k_iter<K; k_iter++) {
    for (int i=1; i<=N_1-1; i++) {
        for (int j=1; j<=N_2-1; j++) {
            double val = fslae[i-1][j-1];

            int jidx=j-1;
            double y_im1_j   = y[i-1][jidx];
            double y_im1_jm1 = (j>1)     ? y[i-1][jidx-1]:0.0;
            double y_im1_jp1 = (j<N_2-1) ? y[i-1][jidx+1]:0.0;

            double y_ip1_j   = y[i+1][jidx];
            double y_ip1_jm1 = (j>1)     ? y[i+1][jidx-1]:0.0;
            double y_ip1_jp1 = (j<N_2-1) ? y[i+1][jidx+1]:0.0;

            // Вычитаем A_i Y_(i-1)
            val -= cA[i-1][j-1]*y_im1_j;
            if (j>=2) val -= aA[i-1][j-1]*y_im1_jm1;
            if (j<=N_2-2) val -= bA[i-1][j-1]*y_im1_jp1;

            // Вычитаем B_i Y_(i+1)
            val -= cB[i-1][j-1]*y_ip1_j;
            if (j>=2) val -= aB[i-1][j-1]*y_ip1_jm1;
            if (j<=N_2-2) val -= bB[i-1][j-1]*y_ip1_jp1;

            phi[j-1]=val;
        }
```

```
            solveTridiagonalSystem(i-1,cC,aC,bC,phi,Y_i,N_2);

            for (int j=0; j<N_2-1; j++) {
               y[i][j]=Y_i[j];
            }
         }
      }
   }

// Максимум-норма ошибки: max|x_k - 1|
double maxNormError(const std::vector<double> &x) {
   double max_error = 0.0;
   for (double val : x) {
      double err = std::fabs(val - 1.0);
      if (err > max_error)
         max_error = err;
   }
   return max_error;
}

void printMatrixBlock(const std::vector<std::vector<double>> &mat) {
   for (size_t i = 0; i < mat.size(); i++) {
      for (size_t j = 0; j < mat[i].size(); j++) {
         std::cout << std::setw(10) << mat[i][j] << " ";
      }
      std::cout << "\n";
   }
}

void printVector(const std::vector<double> &vec) {
   for (double val : vec) {
      std::cout << val << " ";
   }
   std::cout << "\n";
}

void putBlock(std::vector<std::vector<double>> &FullMat,
              int N_1, int N_2,
              int blockRow, int blockCol,
              const std::vector<std::vector<double>> *cM,
```

```
            const std::vector<std::vector<double>> *aM,
            const std::vector<std::vector<double>> *bM,
            bool isIdentity = false,
            bool isZero = false)
{
   int rowStart = blockRow*(N_2-1);
   int colStart = blockCol*(N_2-1);

   int m = N_2-1;
   if (isIdentity) {
      for (int j=0; j<m; j++) {
         FullMat[rowStart+j][colStart+j] = 1.0;
      }
      return;
   }
   if (isZero) {
      // Нулевой блок - ничего не делаем
      return;
   }

   // обычный трехдиагональный блок
   // i_index = blockRow-1 для внутренних блоков (A_i, B_i, C_i)
   // но если blockRow=0 или blockRow=N_1, значит это граничные блоки.
   int i_index = blockRow-1;
   for (int j=1; j<=m; j++) {
      int rr = rowStart + (j-1);
      double c_val = (*cM)[i_index][j-1];
      FullMat[rr][colStart+(j-1)] = c_val;

      if (j>=2) {
         double a_val = (*aM)[i_index][j-1];
         FullMat[rr][colStart+(j-1)-1] = a_val;
      }
      if (j<=m-1) {
         double b_val = (*bM)[i_index][j-1];
         FullMat[rr][colStart+(j-1)+1] = b_val;
      }
   }
}
```

```
void printFullMatrix(int N_1, int N_2,
                     const std::vector<std::vector<double>> &cC,
                     const std::vector<std::vector<double>> &aC,
                     const std::vector<std::vector<double>> &bC,
                     const std::vector<std::vector<double>> &cA,
                     const std::vector<std::vector<double>> &aA,
                     const std::vector<std::vector<double>> &bA,
                     const std::vector<std::vector<double>> &cB,
                     const std::vector<std::vector<double>> &aB,
                     const std::vector<std::vector<double>> &bB)
{
   int N = (N_1+1)*(N_2-1);
   std::vector<std::vector<double>> FullMat(N, std::vector<double>(N,0.0));

   // Первая строка блоков: C_0=I, B_0=0
   putBlock(FullMat, N_1,N_2, 0,0,nullptr,nullptr,nullptr,true,false); // C_0=I
   if (N_1>=1) putBlock(FullMat, N_1,N_2, 0,1,nullptr,nullptr,nullptr,false,true); // B_0=0

   // i=1..N_1-1: [A_i, C_i, B_i]
   for (int i=1; i<=N_1-1; i++) {
      putBlock(FullMat,N_1,N_2, i,i-1,&cA,&aA,&bA);
      putBlock(FullMat,N_1,N_2, i,i,  &cC,&aC,&bC);
      putBlock(FullMat,N_1,N_2, i,i+1,&cB,&aB,&bB);
   }

   // i=N_1: A_(N_1)=0, C_(N_1)=I
   if (N_1>=1) {
      putBlock(FullMat,N_1,N_2, N_1, N_1-1,nullptr,nullptr,nullptr,false,true); // A_(N_1)=0
      putBlock(FullMat,N_1,N_2, N_1, N_1,   nullptr,nullptr,nullptr,true,false); // C_(N_1)=I
   }

   // Выводим полную матрицу
   for (int i=0; i<N; i++) {
      for (int j=0; j<N; j++) {
         std::cout << std::setw(10) << FullMat[i][j] << " ";
      }
      std::cout << "\n";
   }
}
```

```
void gaussSeidelParallel(int N_1, int N_2, int K,
                         const std::vector<std::vector<double>> &cC,
                         const std::vector<std::vector<double>> &aC,
                         const std::vector<std::vector<double>> &bC,
                         const std::vector<std::vector<double>> &cA,
                         const std::vector<std::vector<double>> &aA,
                         const std::vector<std::vector<double>> &bA,
                         const std::vector<std::vector<double>> &cB,
                         const std::vector<std::vector<double>> &aB,
                         const std::vector<std::vector<double>> &bB,
                         const std::vector<std::vector<double>> &fslae,
                         const std::vector<double> &F,
                         std::vector<std::vector<double>> &y)
{
   // Инициализация
   for (int j=0;j<N_2-1;j++){
      y[0][j]=F[j];
      y[N_1][j]=F[N_1*(N_2-1)+j];
   }
   for (int i=1;i<=N_1-1;i++){
      for (int j=0;j<N_2-1;j++)
         y[i][j]=0.0;
   }

   int m=N_2-1;
   std::vector<std::vector<double>> Y_temp(N_1+1,std::vector<double>(m,0.0));

   for (int k_iter=0;k_iter<K;k_iter++){
      std::vector<std::vector<double>> Y_old=y;

      // i,j не указываем в private, т.к. они будут объявлены как итераторы цикла for,
      // что делает их автоматически private для данного цикла.
      #pragma omp parallel default(none) \
         shared(N_1,N_2,m,Y_temp,y,Y_old,fslae,F,cC,aC,bC,cA,aA,bA,cB,aB,bB)
      {
         // Локальные для каждого потока переменные
         std::vector<double> phi(m,0.0);
         std::vector<double> Y_i(m,0.0);
         double val;
         int jidx;
```

```
    #pragma omp for
    for (int i=1;i<=N_1-1;i++){
      for (int j=1;j<=m;j++){
        val=fslae[i-1][j-1];

        jidx=j-1;
        double y_im1_j=Y_old[i-1][jidx];
        double y_im1_jm1=(j>1)?Y_old[i-1][jidx-1]:0.0;
        double y_im1_jp1=(j<m)?Y_old[i-1][jidx+1]:0.0;

        double y_ip1_j=Y_old[i+1][jidx];
        double y_ip1_jm1=(j>1)?Y_old[i+1][jidx-1]:0.0;
        double y_ip1_jp1=(j<m)?Y_old[i+1][jidx+1]:0.0;

        val -= cA[i-1][j-1]*y_im1_j;
        if (j>=2) val -= aA[i-1][j-1]*y_im1_jm1;
        if (j<=m-1) val -= bA[i-1][j-1]*y_im1_jp1;

        val -= cB[i-1][j-1]*y_ip1_j;
        if (j>=2) val -= aB[i-1][j-1]*y_ip1_jm1;
        if (j<=m-1) val -= bB[i-1][j-1]*y_ip1_jp1;

        phi[j-1]=val;
      }

      solveTridiagonalSystem(i-1,cC,aC,bC,phi,Y_i,N_2);

      for (int jj=0;jj<m;jj++){
        Y_temp[i][jj]=Y_i[jj];
      }
    }
  }

  for (int i=1;i<=N_1-1;i++){
    for (int j=0;j<m;j++){
      y[i][j]=Y_temp[i][j];
    }
  }
}
```

```
}

int main() {
    int N_1=100; // N_1 + 1 = количество блоков в строке и столбце
    int N_2=100; // N_2 - 1 = размерность блока (количество строк и столбцов в блоке)
    int K=50; // Число итераций

    std::vector<std::vector<double>> cC1,aC1,bC1,cA1,aA1,bA1,cB1,aB1,bB1,fslae1;
    std::vector<double> F1;

    // Инициализация системы
    initializeSystem(N_1,N_2,cC1,aC1,bC1,cA1,aA1,bA1,cB1,aB1,bB1,fslae1,F1);

    // Выводим вектор правой части
    // std::cout << "Right-hand side vector F:\n";
    // printVector(F1);
    // std::cout << "\n";

    // Выводим один из блоков
    // std::cout << "\nBlock aC:\n";
    // printMatrixBlock(aC1);
    // std::cout << "\n";

    // Вывод полной матрицы
    // std::cout << "\nFull Matrix:\n";
    // printFullMatrix(N_1,N_2,cC1,aC1,bC1,cA1,aA1,bA1,cB1,aB1,bB1);
    // std::cout << "\n";

    // Массив решения y(i,j)
    std::vector<std::vector<double>> y1(N_1+1, std::vector<double>(N_2-1,0.0));

    // Запуск метода Гаусса–Зейделя
    auto start_seq=std::chrono::high_resolution_clock::now();
    gaussSeidel(N_1,N_2,K,cC1,aC1,bC1,cA1,aA1,bA1,cB1,aB1,bB1,fslae1,F1,y1);
    auto end_seq=std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> seq_dur=end_seq-start_seq;
    double seq_time=seq_dur.count();

    std::cout << "Sequential time: " << seq_time << " s\n";
```

```
// Двумерный массив решения y в одномерный вектор x
std::vector<double> x1((N_1+1)*(N_2-1),0.0);
for (int i=0; i<=N_1; i++) {
    for (int j=0; j<N_2-1; j++) {
        int idx = i*(N_2-1) + j;
        x1[idx] = y1[i][j];
    }
}

// std::cout << "Solution vector-1:\n";
// printVector(x1);

// Вычисляем максимум-норму ошибки
double error1 = maxNormError(x1);
std::cout << "Maximum norm of error: " << error1 << "\n\n";

// ---------- Параллельный метод Гаусса-Зейделя ---------- //
std::vector<std::vector<double>> cC2,aC2,bC2,cA2,aA2,bA2,cB2,aB2,bB2,fslae2;
std::vector<double> F2;

// Инициализация системы
initializeSystem(N_1,N_2,cC2,aC2,bC2,cA2,aA2,bA2,cB2,aB2,bB2,fslae2,F2);

// Выводим вектор правой части
// std::cout << "Right-hand side vector F:\n";
// printVector(F2);
// std::cout << "\n";

std::vector<std::vector<double>> y2(N_1+1,std::vector<double>(N_2-1,0.0));

start_seq=std::chrono::high_resolution_clock::now();
gaussSeidelParallel(N_1,N_2,K,cC2,aC2,bC2,cA2,aA2,bA2,cB2,aB2,bB2,fslae2,F2,y2);
end_seq=std::chrono::high_resolution_clock::now();
seq_dur=end_seq-start_seq;
seq_time=seq_dur.count();
std::cout << "Parallel time: " << seq_time << " s\n";

// Двумерный массив решения y в одномерный вектор x
std::vector<double> x2((N_1+1)*(N_2-1),0.0);
for (int i=0; i<=N_1; i++) {
```

```
        for (int j=0; j<N_2-1; j++) {
            int idx = i*(N_2-1) + j;
            x2[idx] = y2[i][j];
        }
    }

    // std::cout << "Solution vector-2:\n";
    // printVector(x2);

    // Вычисляем максимум-норму ошибки
    double error2 = maxNormError(x2);
    std::cout << "Maximum norm of error: " << error2 << "\n\n";

    return 0;
}
```

## ПРИЛОЖЕНИЕ Б

### Код построения визуализации ускорения на языке Python с использованием matplotlib

```
import matplotlib.pyplot as plt

K = [2, 4, 8, 16, 32, 64]
speedup_1 = [0.0190, 0.0204, 0.0509, 0.0737, 0.0440, 0.0721]
speedup_2 = [1.2874, 1.5341, 1.7547, 1.8521, 2.5462, 2.6315]
speedup_3 = [2.6163, 2.8241, 3.1003, 3.2294, 4.0831, 4.5938]

plt.figure(figsize=(10, 6))
plt.plot(K, speedup_1, marker='o', label='$N_1=9, N_2=11$')
plt.xticks(K)
plt.yticks(speedup_1)
plt.xlabel('Число итераций')
plt.ylabel('Ускорение')
plt.title('$N_1=9, N_2=11$')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(K, speedup_2, marker='o', label='$N_1=99, N_2=101$')
plt.xticks(K)
plt.yticks(speedup_2)
plt.xlabel('Число итераций')
plt.ylabel('Ускорение')
plt.title('$N_1=99, N_2=101$')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(K, speedup_3, marker='o', label='$N_1=999, N_2=1001$')
plt.xticks(K)
plt.yticks(speedup_3)
plt.xlabel('Число итераций')
plt.ylabel('Ускорение')
plt.title('$N_1=999, N_2=1001$')
plt.grid(True)
plt.show()
```