Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по Лабораторной работе №4

по курсу «Анализы Алгоритмов»

на тему: «Параллельные вычисления на основе нативных потоков»

Студент группы ИУ7-56Б _____ (Подпись, дата) Мамврийский И. С. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Строганов Д. В.. (Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение

Сегодня компьютерам требуется производить все более трудоемкие вычисления на больших объемах данных. При этом предъявляются требования к скорости вычислений.

Одним из возможных решений уменьшения скорости выполнения задач является параллельное программирование. На одном устройстве параллельные вычисления можно организовать с помощью многопоточности – способности центрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой. При последовательной реализации какого-либо алгоритма, его программу выполняет только одно ядро процессора. Если же реализовать алгоритм так, что независимые вычислительные задачи смогут выполнять несколько ядер параллельно, то это приведет к ускорению решения всей задачи в целом [1].

Для реализации параллельных вычислений требуется выделить те участки алгоритма, которые могут выполняться параллельно без изменения итогового результат, также необходимо правильно организовать работу с данными, чтобы не потерять вычисленные значения.

Целью данной работы является исследование параллельного программирования на основе алгоритма нахождения обратной матрицы.

Для поставленной цели необходимо выполнить следующие задачи:

1) описать основы распараллеливания вычислений;

2) разработать программное обеспечение, которое реализует однопоточный алгоритм нахождения обратной матрицы;

3) разработать и реализовать многопоточную версию данного алгоритма;

4) выполнить замеры процессорного времени работы алгоритма;

5) провести сравнительный анализ по времени работы реализаций алгоритма.

1 Аналитическая часть

В данном разделе представлено теоретическое описание алгоритмов нахождения обратной матрицы.

1.1 Алгоритм нахождения обратной матрицы методом Гаусса-Жордана

Пусть A и B – квадратные матрицы порядка n и $A*B = E$, тогда матрица B называется **обратной** к A и обозначается как A^{-1}, то есть A*A^{-1} = A^{-1}*A = E [2].

В данной лабораторной работе рассмотрим метод Гаусса-Жордана, целью которого является перевод матрицы (A|E)

$$\left(\begin{array}{ccc|ccc} -5 & 23 & -24 & 1 & 0 & 0 \\ -1 & 4 & -5 & 0 & 1 & 0 \\ 9 & -40 & 43 & 0 & 0 & 1 \end{array}\right), \tag{1.1}$$

к виду (E|A^{-1})

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 14 & \frac{29}{2} & \frac{19}{2} \\ 0 & 1 & 0 & 1 & \frac{-1}{2} & \frac{1}{2} \\ 0 & 0 & 1 & -2 & \frac{-7}{2} & \frac{-3}{2} \end{array}\right). \tag{1.2}$$

Данный метод заключается в последовательном обходе строк матрицы, то есть:

1) на k-м шаге работаем с k-й строкой r_k, k-ый элемент которой обозначим как a_k;

2) если $a_k = 0$, то меняем местами строку r_k с одной из тех нижележащих строк, у которых k-ый элемент отличен от нуля. Если таких строк нет, то обратная матрица не существует;

3) если $a_k \neq 1$, умножаем строку r_k на $\frac{1}{a_k}$, если $a_k = 1$, то никакого домножения делать не надо;

4) с помощью строки r_k производим обнуление всех остальных ненулевых элементов k-ого столбца, переходим к следующему шагу.

После обработки последней строки, матрица до черты станет единичной, алгоритм завершится. [3]

1.2 Параллельный алгоритм

В методе нахождения обратной матрицы Гаусса-Жордана умножение r_k строки на $\frac{1}{a_k}$ и обнуление всех ненулевых жлементов k-ого столбца происходит независимо, поэтому есть возможность произвести распараллеливание данных вычислений. Количество строк, на которых производит вычисление один поток, будет определяться количеством потоков, исходная и обратная матрицы будут храниться в разделяемой переменной, доступ к которым будут иметь все потоки.

Вывод

В данном разделе был рассомотрен алгоритм нахождения обратной матрицы методом Гаусса-Жордана и описана возможность его распараллеливания.

2 Конструкторская часть

В данном разделе будет рассмотрена схема алгоритма нахождения обратной матрицы методом Гаусса-Жордана и его версия параллельная версия.

2.1 Требование к прогаммному обеспечению

К программе предъявляются следующие требования:

— на вход подается размерность матрицы, элементы матрицы;

— на выходе — матрица, обратная данной.

2.2 Разработка алгоритмов

На рисунках 2.1-2.2 представлена схема алгоритма метода Гаусса-Жордана для нахождения обратной матрицы, на рисунках 2.6-2.10 – схема параллельного алгоритма. На рисунках 2.3-2.5 представлены вспомогательные функции.

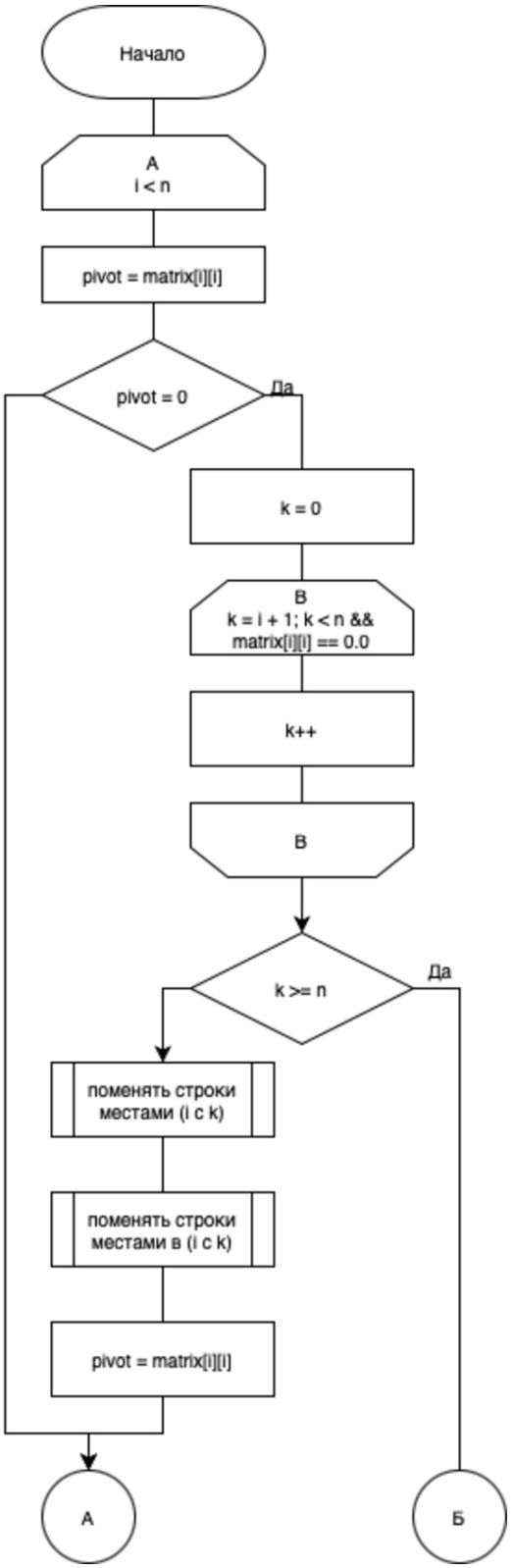

Рисунок 2.1 – Схема алгоритма основной функции метода Гаусса-Жордана

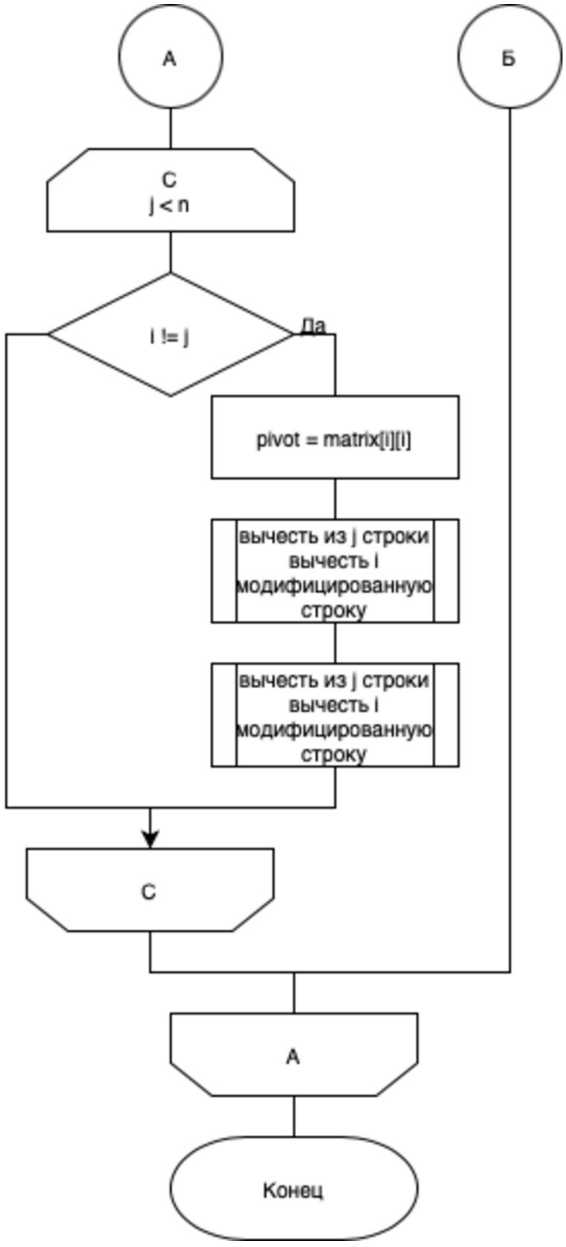

Рисунок 2.2 – Схема алгоритма основной функции метода Гаусса-Жордана

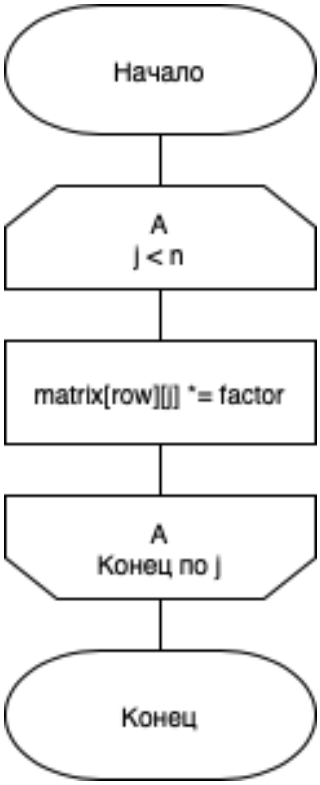

Рисунок 2.3 – Схема алгоритма умножения строки матрицы на элемент

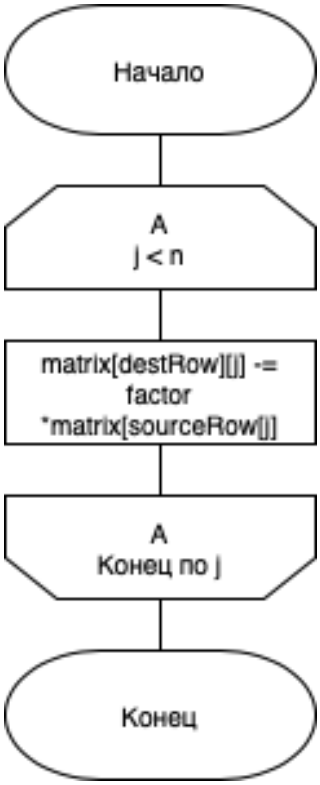

Рисунок 2.4 – Схема алгоритма вычитания из строк матрицы модифицированной текущей строки

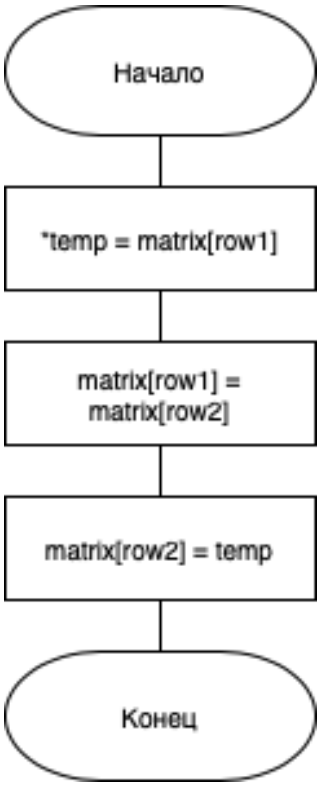

Рисунок 2.5 – Схема алгоритма смены строк местами

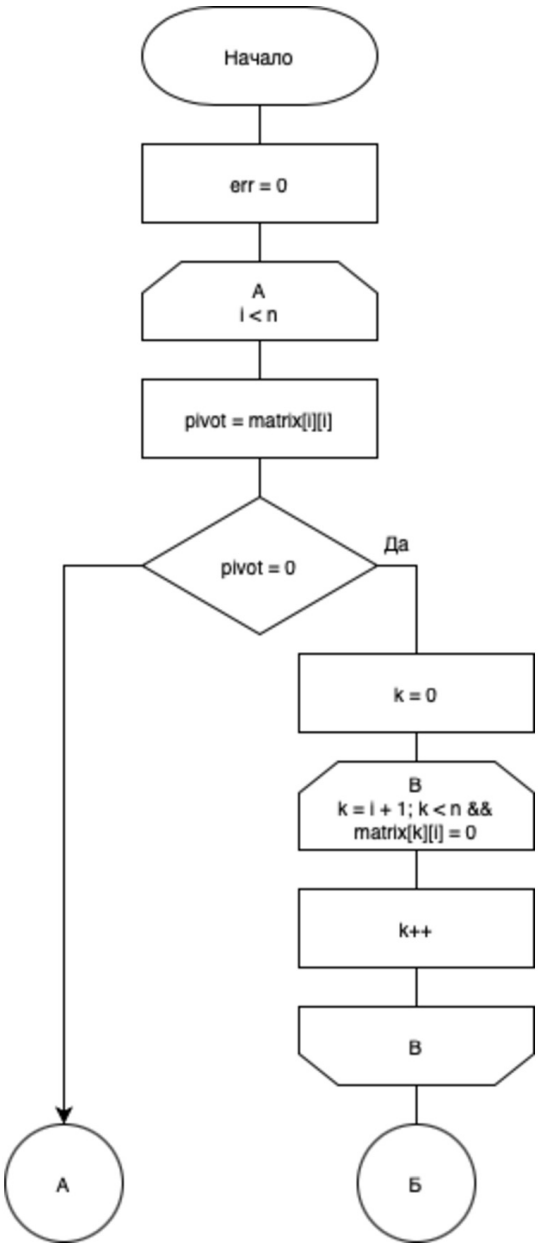

Рисунок 2.6 – Схема параллельного алгоритма основной функции метода Гаусса-Жордана

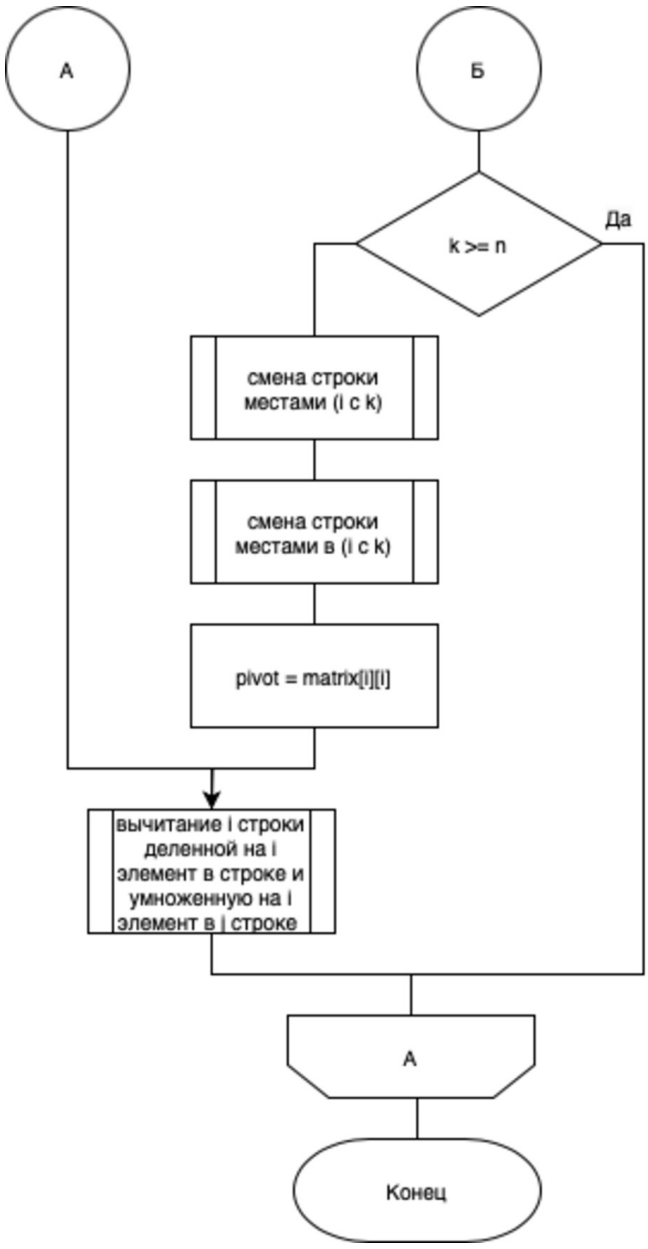

Рисунок 2.7 – Схема параллельного алгоритма основной функции метода Гаусса-Жордана

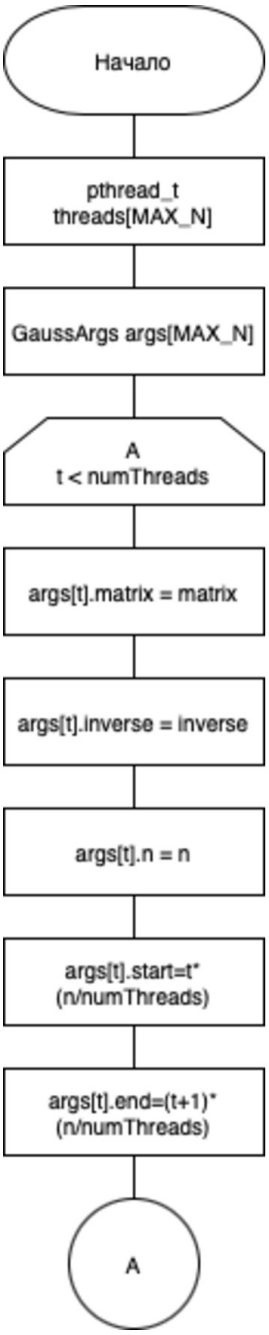

Рисунок 2.8 – Схема функции алгоритма, где происходит распараллеливание

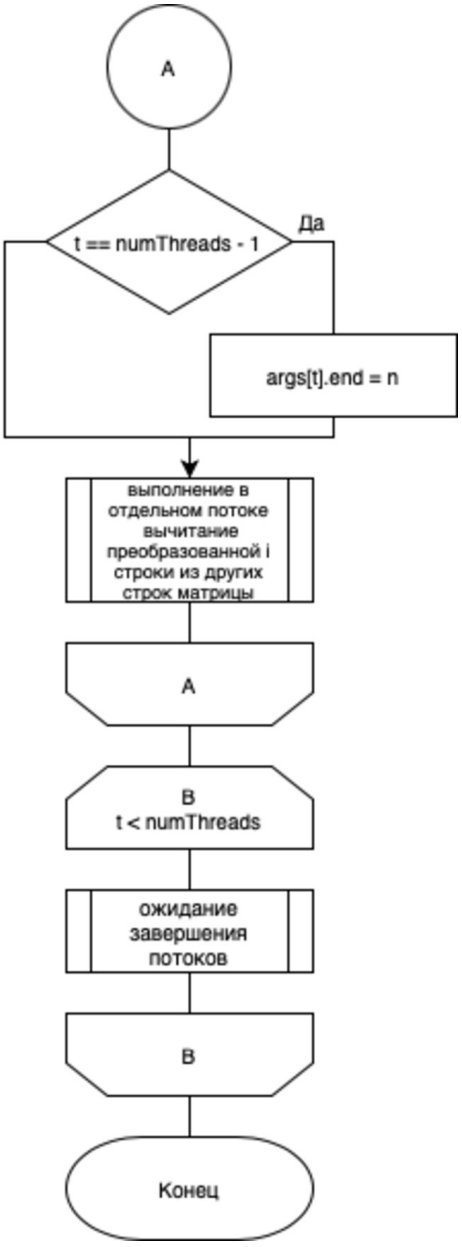

Рисунок 2.9 – Схема функции алгоритма, где происходит распараллеливание

Рисунок 2.10 – Схема параллельного алгоритма вычитания из строк матрицы модифицированной текущей строки

2.3 Структура разрабатываемого программного обеспечения

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода матрицы, функции для выделения памяти.

Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы для алгоритмов метода Гаусса-Жордана и параллельной реализации.

3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования *C* [4]. Данный выбор обусловлен наличием у языка встроенного модуля для измерения процессорного времени и соответствием с выдвинутыми требованиям.

Работа с потоками осуществлялась с помощью функций из модуля <pthread.h> [5]. Для работы с сущностью вспомогательного потока необходимо воспользоваться функцией *pthread_create*() для создания потока и указания функции, которую начнет выполнять созданный поток. Далее при помощи вызова *pthread_join*() необходимо (в рамках данной лабораторной работы) дождаться завершения всех вспомогательных потоков, чтобы в главном потоке обработать результаты их работы.

3.2 Реализация алгоритмов

В данном подразделе представлены листинги кода ранее описанных алгоритмов:

— алгоритм метода Гаусса-Жордана (листинг 3.1);

— параллельный алгоритм метода Гаусса-Жордана (листинг 3.2);

— алгориты вспомогательных функция (листинг 3.3).

Листинг 3.1 – Реализация алгоритма метода Гаусса-Жордана.

```
1  void gauss(double **matrix, double **inverse, int n) {
2      for (int i = 0; i < n; i++) {
3          double pivot = matrix[i][i];
4
5          if (pivot == 0.0) {
6              int k;
7              for (k = i + 1; k < n && matrix[k][i] == 0.0; k++);
8              if (k >= n) {
9                  break;
10             } else {
11                 swapRows(matrix, i, k, n);
12                 swapRows(inverse, i, k, n);
13                 pivot = matrix[i][i];
14             }
15         }
16
17         for (int j = 0; j < n; j++) {
18             if (j != i) {
19                 double factor = matrix[j][i];
20                 subtractRows(matrix, j, i, factor / pivot, n);
21                 subtractRows(inverse, j, i, factor / pivot, n);
22             }
23         }
24     }
25 }
```

Листинг 3.2 – Реализация параллельного алгоритма метода Гаусса-Жордана.

```
1  void subtractRowsEx(double **matrix, double **inverse, int
       start, int end, int n, int i) {
2      for (int j = start; j < end; j++) {
3          if (j != i) {
4              double factor = matrix[j][i];
5              subtractRows(matrix, j, i, factor / matrix[i][i],
                   n);
6              subtractRows(inverse, j, i, factor / matrix[i][i],
                   n);
7          }
8      }
9  }
10
11
12 void *subtractRowsThread(void *args) {
13     GaussArgs *gaussArgs = (GaussArgs *)args;
14     subtractRowsEx(gaussArgs->matrix, gaussArgs->inverse,
           gaussArgs->start, gaussArgs->end, gaussArgs->n,
           gaussArgs->num);
15     return NULL;
16 }
17
18 void gauss_thread(double **matrix, double **inverse, int n, int
       numThreads) {
19     int err = 0;
20     for (int i = 0; i < n; i++) {
21         double pivot = matrix[i][i];
22         if (pivot == 0.0) {
23             int k;
24             for (k = i + 1; k < n && matrix[k][i] == 0.0; k++);
25             if (k >= n) {
26                 break;
27             } else {
28                 swapRows(matrix, i, k, n);
29                 swapRows(inverse, i, k, n);
30                 pivot = matrix[i][i];
31             }
32         }
33
34         subtractRowsThreadEx(matrix, inverse, n, numThreads, i);
35     }
```

```
36 }
37
38 void subtractRowsThreadEx(double **matrix, double **inverse,
       int n, int numThreads, int i) {
39     pthread_t threads[MAX_N];
40     GaussArgs args[MAX_N];
41     for (int t = 0; t < numThreads; t++) {
42         args[t].matrix = matrix;
43         args[t].inverse = inverse;
44         args[t].n = n;
45         args[t].start = t * (n / numThreads);
46         args[t].end = (t + 1) * (n / numThreads);
47         args[t].num = i;
48
49         if (t == numThreads - 1) {
50             args[t].end = n;
51         }
52
53         pthread_create(&threads[t], NULL, subtractRowsThread,
               (void *)&args[t]);
54     }
55
56     for (int t = 0; t < numThreads; ++t) {
57         pthread_join(threads[t], NULL);
58     }
59 }
```

Листинг 3.3 – Реализация алгоритмов вспомогательных функций.

```
1  void multiplyRow(double **matrix, int row, double factor, int
       n) {
2      for (int j = 0; j < n; j++) {
3          matrix[row][j] *= factor;
4      }
5  }
6
7  void subtractRows(double **matrix, int destRow, int sourceRow,
       double factor, int n) {
8      for (int j = 0; j < n; j++) {
9          matrix[destRow][j] -= factor * matrix[sourceRow][j];
10     }
11 }
12
13 void swapRows(double **matrix, int row1, int row2, int n) {
14     double *temp = matrix[row1];
15     matrix[row1] = matrix[row2];
16     matrix[row2] = temp;
17 }
```

3.3 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов нахождения обратной матрицы.

Таблица 3.1 – Функциональные тесты

Матрица	**Ожидаемый результат**
$\begin{pmatrix} 5 & 9 & 2 \\ 6 & 11 & 4 \\ 1 & 3 & 1 \end{pmatrix}$	$\begin{pmatrix} 0.1 & 0.3 & -2 \\ 0.2 & 0 & 0.9 \\ -1 & 0.7 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.33333 \end{pmatrix}$
$\begin{pmatrix} 2 & 1 & -1 \\ 3 & 2 & -2 \\ 1 & -1 & 2 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 \\ -8 & 5 & 1 \\ -5 & 3 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -2 & 1 \\ 1.5 & -0.5 \end{pmatrix}$

Алгоритмы, реализованные в данной лабораторной работе, функциональные тесты прошли успешно.

Вывод

В данном разделе были реализованы алгоритм нахождения обратой матрицы методом Гаусса-Жорадана и его параллельная версия. Также было выполнено тестирования данных алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее:

— процессор – 2 Гц 4-ядерный процессор Intel Core i5;

— оперативная память – 16 ГБайт;

— операционная система – macOS Venura 13.5.2.

4.2 Примеры работы программы

На рисунке 4.1 представлен пример работы программы.

Листинг 4.1 – Пример работы программы

```
ivanmamvriyskiy@h54 main % ./a.out
Меню:
1)Одинчный подсчет;
2)Замер времени;
3)Выход.
1
Введите размерность матрицы N: 3
1 2 3
4 0 5
6 7 0
Обратная матрица:
-0.321101 0.192661 0.091743
0.275229 -0.165138 0.064220
0.256881 0.045872 -0.073394
Обратная матрица параллельным алгоритмом:
-0.321101 0.192661 0.091743
0.275229 -0.165138 0.064220
0.256881 0.045872 -0.073394
```

4.3 Время выполнения алгоритмов

На рисунке 4.1 представлена зависимость времени выполнения алгоритма от размерности матрицы при различных количествах потоков: последовательный алгоритм, 8 потоков (оптимальное количество), и 32 потока (для демонстрации затрат на переключение ядер при превышении оптимального количества потоков). Таким образом, график показывает время работы программы с и без распараллеливания.

Таблица 4.1 – Время выполнения работы алгоритмов в зависимости от размерности матрицы и количества потокв(мс).

Размеренность	1	2	4	8	16	32
1 000	13	7	4	3	3	4
2 000	57	31	15	13	13	13
3 000	127	65	48	55	56	61
4 000	264	140	111	92	100	103
5 000	434	245	182	162	164	166
6 000	636	367	268	240	243	244
7 000	948	610	411	356	358	370
8 000	1 347	769	618	443	480	496
9 000	1 516	930	687	553	607	732
10 000	1 710	1 093	741	698	816	878

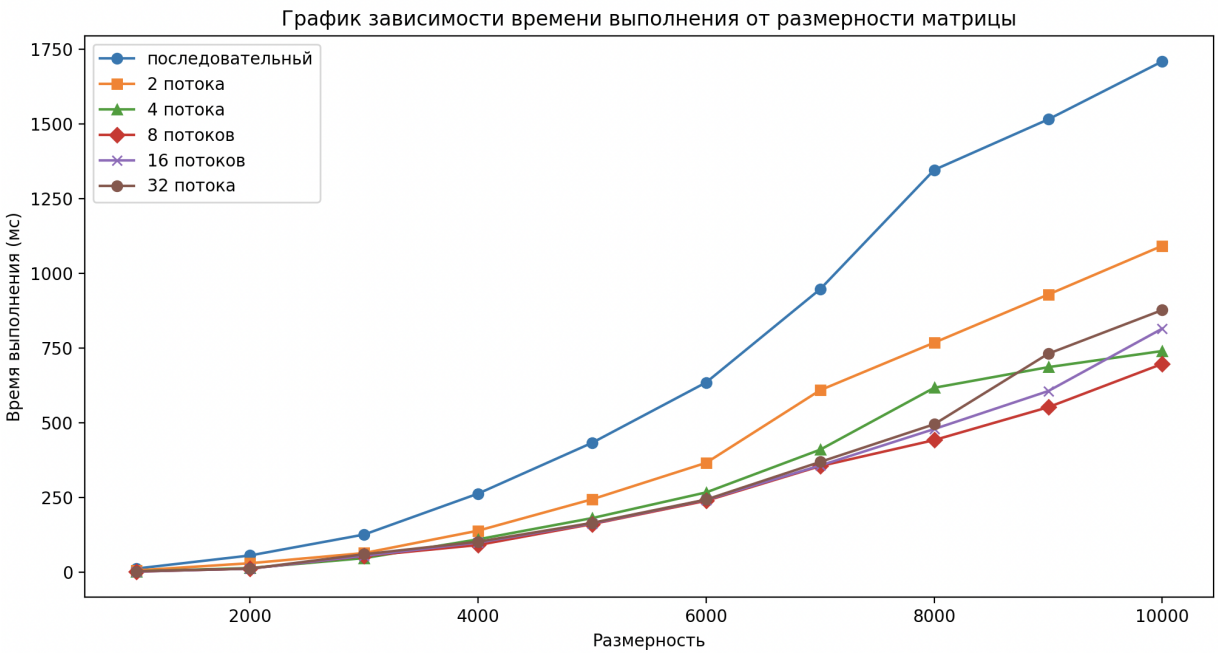

Рисунок 4.1 – График зависимости времени выполнения работы от размерности матрицы при различных количествах потоков

По результатам замера времени можно сделать следующие выводы:

— при движении от наименьшего числа потоков к числу логических ядер время работы алгоритма уменьшается;

— при превышении числа логических ядер время увеличивается, так как затрачивается время на переключение ядра между потоками;

— последовательная реализация работает примерно в 3 раз медленне, чем реализация с использованием оптимального количества потоков, то есть 8 потоков.

4.4 Вывод

Таким образом, для максимального ускорения работы программы с помощью распараллеливания необходимо использовать оптимальное число логических ядер.

Заключение

В ходе исследования был проведен сравнительный анализ алгоритмов, в результате которого было выяснено, что при движении от минимального количества ядер к оптимальному(8 потоков) время работы алгоритма уменьшается, при превышении оптимального количества ядер время работы алгоритма начинает замедляться, так как происходят затраты времени на переключение ядра между потоками.

Исходя из данных показателей для более быстрой работы необходимо использовать оптимальное количество ядер.

Цель данной лабораторной работы, которая заключается в исследовании параллельного программирования на основе алгоритма нахождения обратной матрицы выполнены. Также были выполнены следующие задачи:

— описаны основы распараллеливания вычислений;

— разработано программное обеспечение, реализующее однопоточный алгоритм нахождения обратной матрицы;

— разработано программное обеспечение, реализующее многопоточную версию данного алгоритма;

— выполнены замеры процессорного времени работы алгоритмов;

— проведен сравнительный анализ по времени работы данных алгоритмов.

Список используемых источников

1. Многопоточность [Электронный ресурс]. Режим доступа: https://ru.coursera.org/lecture/ios-multithreading/chto-takoie-mnoghopotochnost-4MMgN.

2. Ващенко Г. В. Вычислительная математика. Основы конечных методов решения систем линейных алгебраических уравнений. [Электронный ресурс]. Режим доступа: https://ru.coursera.org/lecture/ios-multithreading/chto-takoie-mnoghopotochnost-4MMgN.

3. Метод элементарных преобразований. [Электронный ресурс]. Режим доступа: https://amkbook.net/mathbook/inverse-matrix-gauss-jordan-methods.

4. Документация Си. [Электронный ресурс]. Режим доступа: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf.

5. Документация по pthread. [Электронный ресурс]. Режим доступа: https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html.