



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»

Отчёт

по лабораторной работе №4

Название: Параллельное программирование

Дисциплина: Анализ алгоритмов

Студент

ИУ7-54Б

(Группа)

Л.Е.Тартыков

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

Л.Л. Волкова

(Подпись, дата)

(И.О. Фамилия)

Москва, 2021

Содержание

Введение	3
1 Аналитический раздел	4
1.1 Определение матрицы	4
1.2 Алгоритм поиска минимума в матрице.	4
1.3 Параллельная реализация алгоритма	5
1.4 Вывод	5
2 Конструкторский раздел	6
2.1 Выбранные типы и структуры данных	6
2.2 Схемы алгоритмов	6
2.3 Вывод	9
3 Технологический раздел	10
3.1 Выбор средств реализации	10
3.2 Листинги программ	10
3.3 Вспомогательные модули	12
3.4 Тестирование	12
3.5 Вывод	13
4 Исследовательский раздел	14
4.1 Технические характеристики	14
4.2 Временные характеристики выполнения	14
4.3 Вывод	15
Заключение	17
Список литературы	18

Введение

При выполнении множества задач необходимо задействовать эффективно как можно много ресурсов системы. Для этого необходимо увеличивать скорость выполнения программ. В настоящее время по определенным техническим причинам стало невозможным увеличивать тактовую частоту процессора. Однако есть другой способ увеличения производительности – размещение нескольких ядер в процессоре, но это требует другого подхода в программировании.

Параллельное программирование - новый подход в технологии разработки программного обеспечения, которое основывается на понятии "поток"[1]. Поток - часть кода программы, которая может выполняться параллельно с другими частями кода программы. Многопоточность - способность центрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько потоков.

Целью лабораторной работы является изучение и реализация параллельного программирования для решения поиска минимального элемента в матрице. Для её достижения необходимо выполнить следующие задачи:

- исследовать подходы параллельного программирования;
- привести схемы алгоритмов последовательного и параллельного поиска минимального элемента матрицы;
- описать используемые структуры данных;
- выполнить тестирование реализации алгоритмов методом черного ящика;
- провести сравнительный анализ этих алгоритмов по процессорному выполнению времени на основе экспериментальных данных.

1 Аналитический раздел

В данном разделе рассматривается описание общей задачи, принцип распределения задач.

1.1 Определение матрицы

Матрицей размера $m \times n$ называется прямоугольная таблица элементов некоторого множества (например, чисел или функций), имеющая m строк и n столбцов [2]. Элементы a_{ij} , из которых составлена матрица, называются элементами матрицы. Условимся, что первый индекс i элемента a_{ij} соответствует номеру строки, второй индекс j – номеру столбца, в котором расположен элемент a_{ij} . Матрица может быть записана по формуле (1.1).

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad (1.1)$$

1.2 Алгоритм поиска минимума в матрице.

Для решения нахождения поиска минимума в матрице необходимо выполнить следующие действия: для каждой строки матрицы $A[i]$ нужно выполнить вычисления локального максимума по формуле (1.2).

$$loc_{min} = \min(a_{i0}, \dots, a_{in}) \quad (1.2)$$

Затем полученные каждые локальные минимумы сравниваются друг с другом. Таким образом, будет выполнен поиск глобального минимума в матрице.

1.3 Параллельная реализация алгоритма

В каждой строке матрицы выполняются схожие вычисления по нахождению в ней минимального элемента. Эти действия являются независимыми, причем строка матрицы не изменяется, поэтому для параллельного вычисления целесообразно разделить данную задачу между потоками. Каждый поток будет выполняться с необходимым объемом данных.

1.4 Вывод

В данном разделе были описаны необходимая задача и принцип её разделения. На вход программному обеспечению подается матрица; на выход программа должна выдать результат - значение максимального элемента в матрице. Все данные, подаваемые на вход, являются корректными.

2 Конструкторский раздел

В данном разделе представлены схемы алгоритма поиска минимального элемента в матрице последовательным способом и с помощью параллельной реализации. Было выполнено описание способов тестирования и выделенных классов эквивалентностей. Также представлены выбранные типы и структуры данных.

2.1 Выбранные типы и структуры данных

В данной работе используются следующие типы и структуры данных:

- `matrix_args_t` - структура, содержащая матрицу, количество строк и столбцов в ней;
- `pthread_args_t` - структура, содержащая информацию о номере потока, количестве потоков, количестве строк матрицы, локальный минимум и структуру `matrix_args_t`.

2.2 Схемы алгоритмов

Ниже представлены следующие схемы алгоритмов:

- рисунок 2.1 - схема алгоритма поиска минимума в матрице (последовательная реализация);

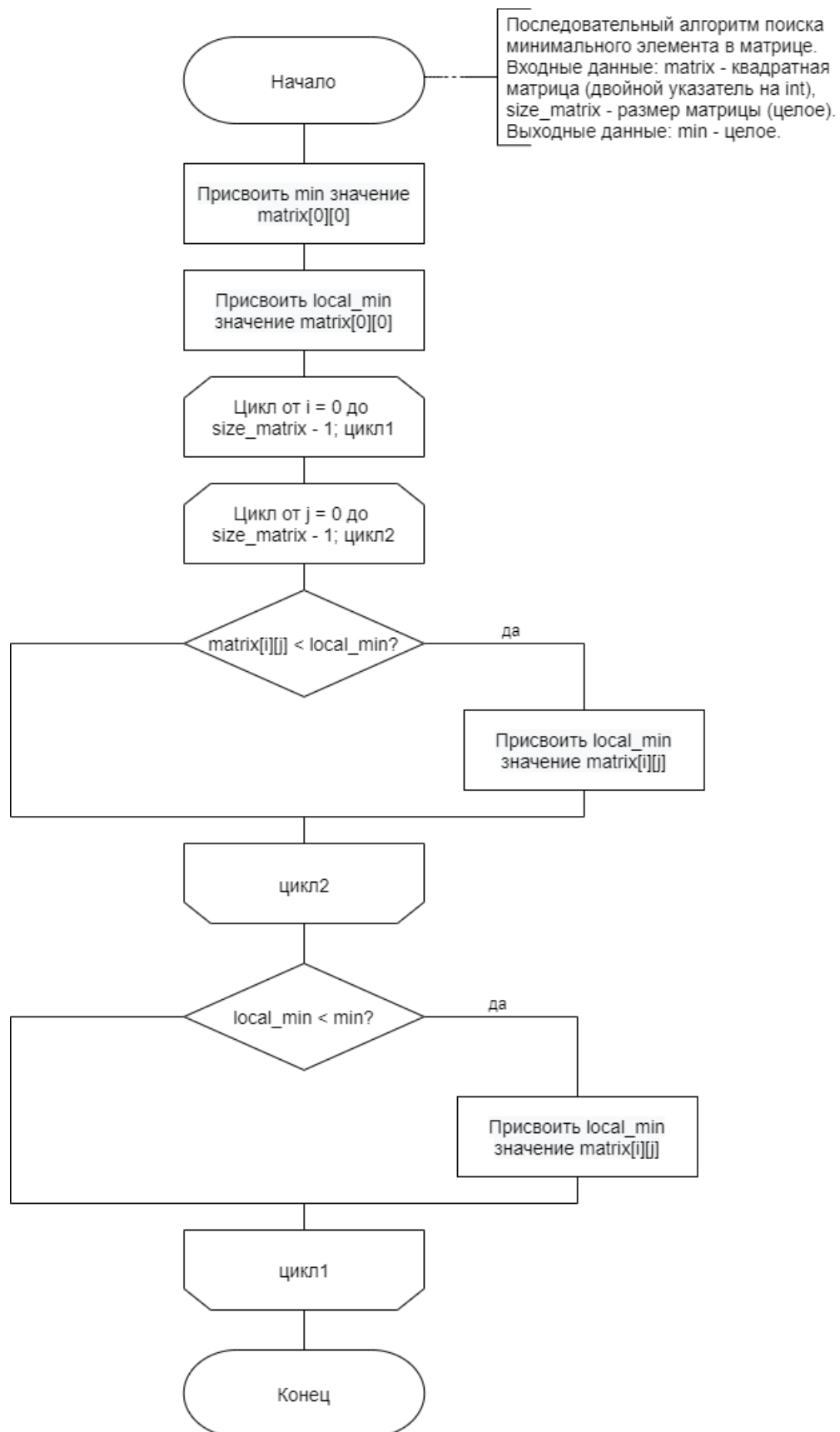


Рисунок 2.1 – Схема алгоритма поиска минимума в матрице (последовательная реализация).

- рисунок 2.2 - схема алгоритма поиска минимума в матрице (параллельная реализация);

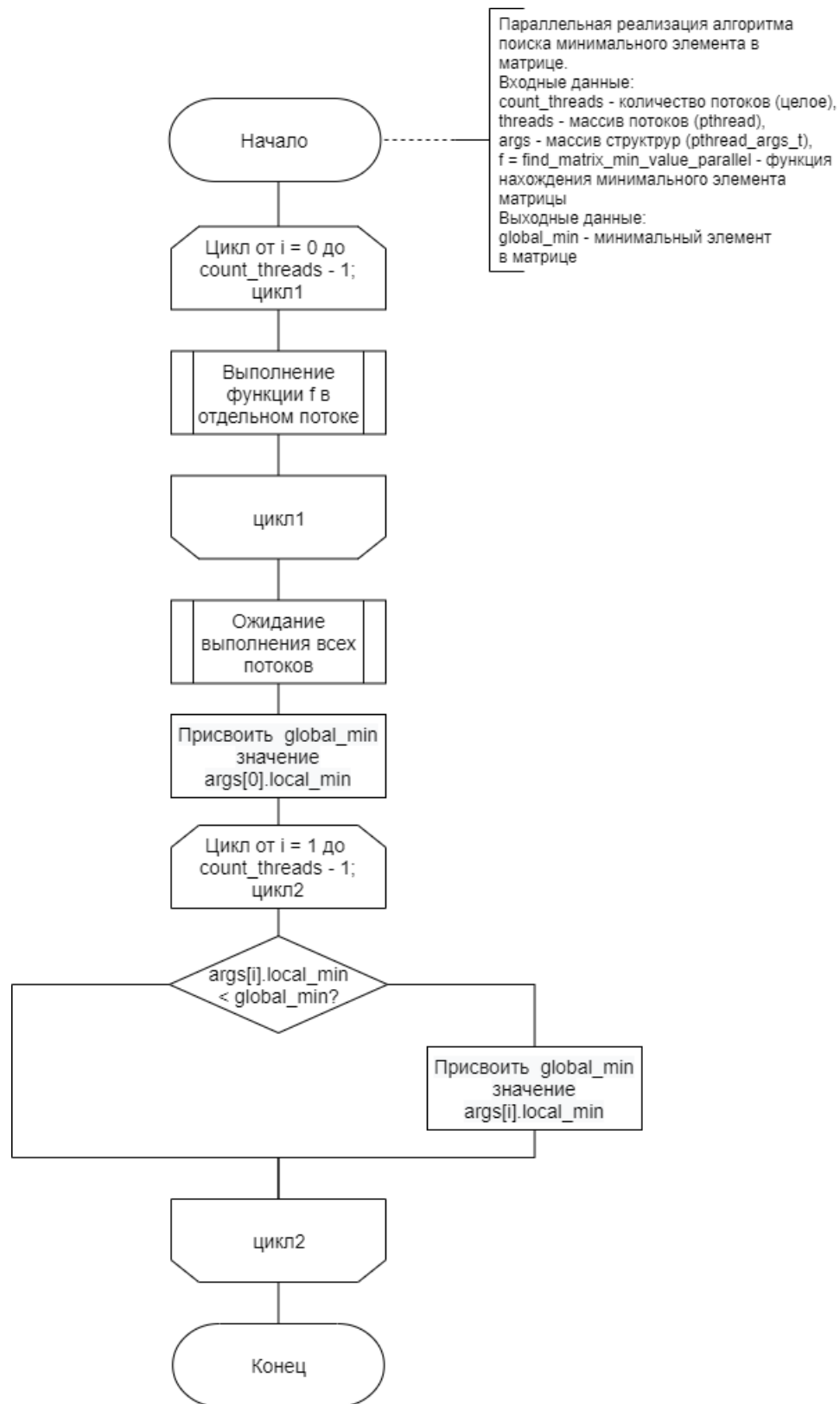


Рисунок 2.2 – Схема алгоритма поиска минимума в матрице (параллельная реализация).

- рисунок 2.3 - распределение задач поиска минимума в матрице (параллельная реализация);

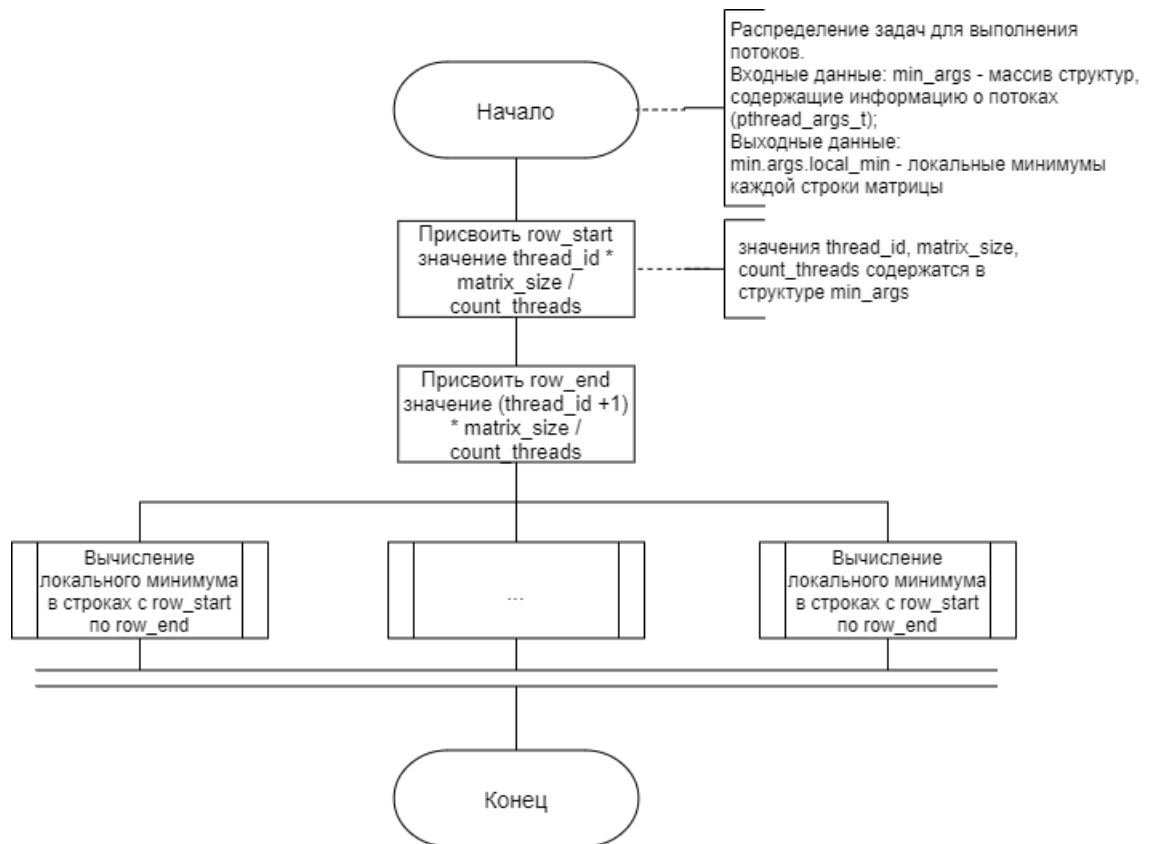


Рисунок 2.3 – Распределение задач поиска минимума в матрице (параллельная реализация).

2.3 Вывод

На основе теоретических данных, полученных в аналитическом разделе, были построены схемы необходимых реализаций алгоритма, представлены необходимые типы и структуры данных.

3 Технологический раздел

В данном разделе приведены средства реализации и листинги кода.

3.1 Выбор средств реализации

Для реализации алгоритмов в данной лабораторной работе был выбран язык программирования C++ [3]. Данный выбор обусловлен тем, что он содержит нативный код, что необходимо для уменьшения времени выполнения алгоритмов с помощью распараллеливания. Выбранным механизмом работы с потоками предоставляет библиотека pthread[5]. POSIX определяет набор интерфейсов для программирования потоков. В качестве среды разработки был использован Visual Studio Code[4], так как в нем присутствует поддержка практически всех языков программирования. Выполнение замера процессорного времени осуществлено при помощи библиотеки chrono [6].

3.2 Листинги программ

Ниже представлены листинги разработанных алгоритмов умножения матриц.

Листинг 3.1 – Программный код алгоритма нахождения минимума в матрице (последовательная реализация).

```
1      int find_matrix_min_value(int **matrix, int size_matrix)
2      {
3          int min = matrix[0][0], local_min = matrix[0][0];
4          for (int i = 0; i < size_matrix; i++)
5          {
6              for (int j = 0; j < size_matrix; j++)
7              {
8                  if (matrix[i][j] < local_min)
9                      local_min = matrix[i][j];
10             }
11             if (local_min < min)
12                 min = local_min;
13         }
14         return min;
15     }
```

Листинг 3.2 – Программный код алгоритма нахождения минимума в матрице (параллельная реализация).

```
1      void *find_matrix_min_value_parallel(void *args)
2      {
3          pthread_args_t *mult_args = (pthread_args_t *)args;
4          int row_start = mult_args->thread_id * (mult_args->matrix_size /
5              mult_args->count_threads);
6          int row_end = (mult_args->thread_id + 1) * (mult_args->
7              matrix_size / mult_args->count_threads);
8          int **matrix = mult_args->args->matrix;
9          mult_args->local_min = matrix[0][0];
10
11         for (int i = row_start; i < row_end; i++)
12         {
13             for (int j = 0; j < mult_args->matrix_size; j++)
14             {
15                 if (matrix[i][j] < mult_args->local_min)
16                     mult_args->local_min = matrix[i][j];
17             }
18         }
19         return NULL;
```

3.3 Вспомогательные модули

На листингах представлены программные модули, которые используются в данных функциях:

Листинг 3.3 – Программный код класса по работе с матрицами.

```
1   for (int k = 0; k < count_threads; k++){
2       pthread_create(threads + k, NULL, find_matrix_min_value_parallel
3           , args + k);
4
5   for (int k = 0; k < count_threads; k++){
6       pthread_join(threads[k], NULL);
7   }
8   int global_min = args[0].local_min;
9   for (int i = 1; i < count_threads; i++){
10      if (args[i].local_min < global_min)
11          global_min = args[i].local_min;
12  }
```

3.4 Тестирование

Для тестирования используется метод черного ящика. В данном разделе приведена таблица 3.1, в которой указаны классы эквивалентностей тестов:

Таблица 3.1 – Таблица тестов

№	Описание теста	Матрица	Ожидаемый результат
1	Квадратная матрица	$\begin{pmatrix} 6 & 9 & 8 \\ 0 & 3 & 6 \\ 4 & 9 & 5 \end{pmatrix}$	0
2	Прямоугольная матрица	$\begin{pmatrix} 2 & 3 & -2 \\ 3 & 9 & 2 \end{pmatrix}$	-2
3	Несколько минимумов	$\begin{pmatrix} 5 & 1 \\ 1 & 4 \\ 6 & 4 \end{pmatrix}$	1

Для последовательной и параллельной реализации алгоритма поиска

минимума в матрице все тесты пройдены успешно.

3.5 Вывод

В данном разделе был выбран язык программирования, среда разработки, описан механизм работы с потоками. Выполнена реализация последовательного и параллельного алгоритма. Было успешно проведено их тестирование методом черного ящика по таблице 3.1.

4 Исследовательский раздел

В данном разделе приведены технические характеристики устройства, а также таблица сравнения скорости выполнения последовательной и параллельной реализации алгоритма.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Ubuntu 20.04;
- память: 8 GiB;
- процессор: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz;
- количество ядер: 4;
- количество логических ядер: 8.

Тестирование проводилось на ноутбуке, который был подключен к сети питания. Во время проведения тестирования ноутбук был нагружен только встроенными приложениями окружения, самим окружением и системой тестирования.

4.2 Временные характеристики выполнения

Ниже был проведен анализ времени работы алгоритма для последовательной и параллельной реализации. Проведем работу каждого из них $n = 10$ раз и поделим на число n . Получим среднее значение работы каждого из алгоритмов. Для параллельного алгоритма выполнение вычислений производилось на $M \in \{1, 2, 4, 8, \dots, 4 \cdot L\}$ потках, где L - число логических ядер данной ЭВМ. Тестирование проведено для квадратных матриц целых чисел.

Выполним анализ скорости выполнения последовательной и параллельной реализации алгоритма поиска минимума в матрице. Результат приведен в таблицах 4.1.

Таблица 4.1 – Таблица времени реализации последовательного и параллельного алгоритмов (в секундах).

Размер матрицы	Последовательный	Количество потоков		
		1 поток	2 потока	4 потока
100	2.75149e-05	7.89085e-05	8.27494e-05	9.9439e-05
250	0.00017374	0.000256578	0.000144029	0.000149204
500	0.000787548	0.000863721	0.000454607	0.000420068
1000	0.00316544	0.00327061	0.00177363	0.00145989
2000	0.0135465	0.0129128	0.00711915	0.0055497
3500	0.0385047	0.04707425	0.0213427	0.0202112
5000	0.0768607	0.0804154	0.047094	0.0342458

Таблица 4.2 – Таблица времени реализации последовательного и параллельного алгоритмов (в секундах).

Размер матрицы	Последовательный	Количество потоков		
		8 потоков	16 потоков	32 потока
100	2.75149e-05	0.0019957	0.00431042	0.000867361
250	0.00017374	0.000243679	0.000460978	0.0010327
500	0.000787548	0.000504962	0.000642802	0.000964574
1000	0.00316544	0.00165852	0.0016858	0.00174856
2000	0.0135465	0.00564542	0.00490783	0.00503544
3500	0.0385047	0.0182714	0.0142113	0.0137701
5000	0.0768607	0.0296331	0.0280696	0.0267538

4.3 Вывод

По результатам проведенного эксперимента можно сделать вывод о скорости выполнения каждой из реализацией и целесообразности создания потоков: для любых размеров матриц не имеет смысла создавать параллельную реализацию для одного потока, так как для этого требуются дополнительные временные затраты на его создание. Скорость выполнения при использовании двух потоков меньше на 87% скорости последовательной реализации на размерах 100, 250. Также для этих размеров

квадратной матрицы время выполнения для 8 потоков, что соответствует числу логических процессоров данной ЭВМ, меньше по сравнению с временем для 16 и 32 на 45%. Для размерностей, начиная с 2000, эта разница составляет уже 2,5 раза.

Заключение

В данной лабораторной работе были рассмотрены основополагающие материалы, которые потребовались при параллельной и однопоточной реализации алгоритма поиска минимального элемента в матрице. Были рассмотрены схемы последовательной реализации и многопоточной, произведен их сравнительный анализ. Экспериментально подтверждены различия во временной эффективности последовательной программы и с использованием одного и более потоков: для любых размеров матриц не имеет смысла создавать параллельную реализацию для одного потока, так как для этого требуются дополнительные временные затраты на его создание. Для небольших размеров матриц порядка 500, 1000 наилучшее время среди остальных вариаций была выполнена программа с использованием восьми потоков, что соответствует числу логических ядер данной машины. При размерах квадратной матрицы, больше тысячи, с увеличением числа потоков, начиная с восьми, увеличивается скорость выполнения программы на 7%.

Список литературы

- [1] Таненбаум, Э., Бос Х. Современные операционные системы. 4-е изд. / Э. Таненбаум, Х. Бос. — СПб.: Питер, 2015. - 1120 с.: ил. - (Серия "Классика computer science")
- [2] Канатников, А. Н. Аналитическая геометрия: учебник для вузов / А.Н.Канатников, А.П.Крищенко; под ред В.С.Зарубина, А.П.Крищенко. - 9-е изд. - Москва: Издательство МГТУ им. Н.Э.Баумана, 2019. - 374, [2]с.:ил. - (Математика в техническом университете; вып 3).
- [3] C++11. [электронный ресурс], режим доступа: <https://en.cppreference.com/w/cpp/11> (дата обращения 19.10.2021)
- [4] Documentation for Visual Studio Code. [электронный ресурс], режим доступа: <https://code.visualstudio.com/docs> (дата обращения: 19.10.2021)
- [5] Posix threads. [электронный ресурс], режим доступа: <https://man7.org/linux/man-pages/man7/pthreads.7.html> (дата обращения 19.10.2021)
- [6] Data and time utilities. [электронный ресурс], режим доступа: <https://en.cppreference.com/w/cpp/chrono> (дата обращения 20.10.2021)