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»

Лабораторная работа №4
по дисциплине "Анализ Алгоритмов"

Тема Параллельное программирование

Студент Ковалец К. Э.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Многопоточность — способность центрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой. Этот подход отличается от многопроцессорности, так как многопоточность процессов и потоков совместно использует ресурсы одного или нескольких ядер: вычислительных блоков, кэш-памяти ЦПУ или буфера перевода с преобразованием.

Если многопроцессорные системы включают в себя несколько полных блоков обработки, многопоточность направлена на максимизацию использования ресурсов одного ядра, используя параллелизм на уровне потоков, а также на уровне инструкций. Поскольку эти два метода являются взаимодополняющими, их иногда объединяют в системах с несколькими многопоточными ЦП и в ЦП с несколькими многопоточными ядрами.

Целью данной лабораторной работы является изучения параллельных вычислений на материале сортировки выбором строк матрицы.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- исследовать основы параллельных вычислений;
- привести схемы рассматриваемых алгоритмов (однопоточная сортировка выбором, многопоточная сортировка выбором);
- описать используемые структуры данных;
- описать структуру разрабатываемого ПО;
- определить средства программной реализации;
- провести сравнительный анализ времени работы алгоритмов;
- провести модульное тестирование;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будет представлено описание многопоточности и алгоритма сортировоки выбором.

1.1 Многопоточность

Смысл многопоточности — квазимногозадачность на уровне одного исполняемого процесса. Значит, все потоки процесса помимо общего адресного пространства имеют и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

Многопоточность имеет как свои преимущества, тат и недостатки. Она помогает облегчить программу посредством использования общего адресного пространства, уменьшить затраты на создание потока в сравнении с процессами, повысить производительность процесса за счёт распараллеливания процессорных вычислений. При этом несколько потоков могут вмешиваться друг в друга при совместном использовании аппаратных ресурсов, могут возникнуть проблемы планирования потоков, да и с программной точки зрения аппаратная поддержка многопоточности более трудоемка для программного обеспечения.

Так как строки матрицы можно сортировать независимо друг от друга, то для параллельной сортировки матрицы будет достаточно просто равным образом распределить её строки между потоками.

1.2 Алгоритм сортировки выбором

Алгоритм сортировки выбором [1] заключается в поиске на необработанном срезе массива или списка минимального значения и в дальнейшем обмене этого значения с первым элементом необработанного среза. На следующем шаге необработанный срез уменьшается на один элемент.

Идея алгоритма поэтапно:

- Найти наименьшее значение в списке.
- Записать его в начало списка, а первый элемент - на место, где раньше стоял наименьший.
- Снова найти наименьший элемент в списке. При этом в поиске не участвует первый элемент.
- Второй минимум поместить на второе место списка. Второй элемент при этом перемещается на освободившееся место.
- Продолжать выполнять поиск и обмен, пока не будет достигнут конец списка.

1.3 Вывод

В этом разделе был рассмотрен алгоритм сортировоки выбором. На вход ему будет поступать матрица, строки которой надо будет отсортировать выбором по возрастанию. При попытке задать некорректные данные для ввода кол-ва строк или кол-ва столбцов будет выдано сообщение об ошибке. Реализуемое ПО будет давать возможность выбрать алгоритм (с многопоточностью или без неё) и вывести для него результат вычисления, а также возможность произвести сравнение алгоритмов по затраченному времени.

2 Конструкторская часть

В данном разделе будут приведены схемы однопоточной и многопоточной реализаций алгоритма сортировки выбором строк матрицы, приведено описание используемых типов данных, а также описана структура ПО.

2.1 Схемы алгоритмов

На вход алгоритмов падаётся матрица $matr$, для алогоритма с использованием многопоточности на вход также поступает кол-во потоков. На выходе получаем матрицу $matr$ с отсортированными по возрастанию строками.

На рис. 2.1 - 2.3 приведены схемы однопоточной и многопоточной реализаций алгоритма сортировки выбором строк матрицы.

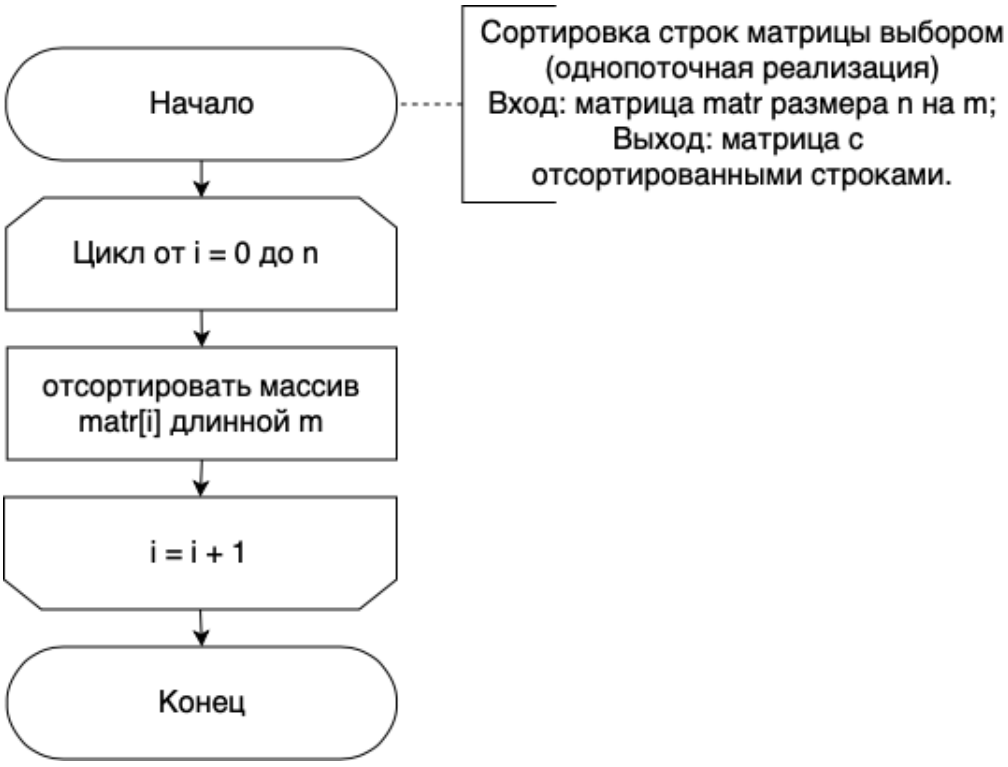

Рисунок 2.1 – Схема однопоточной реализации сортировки строк матроицы

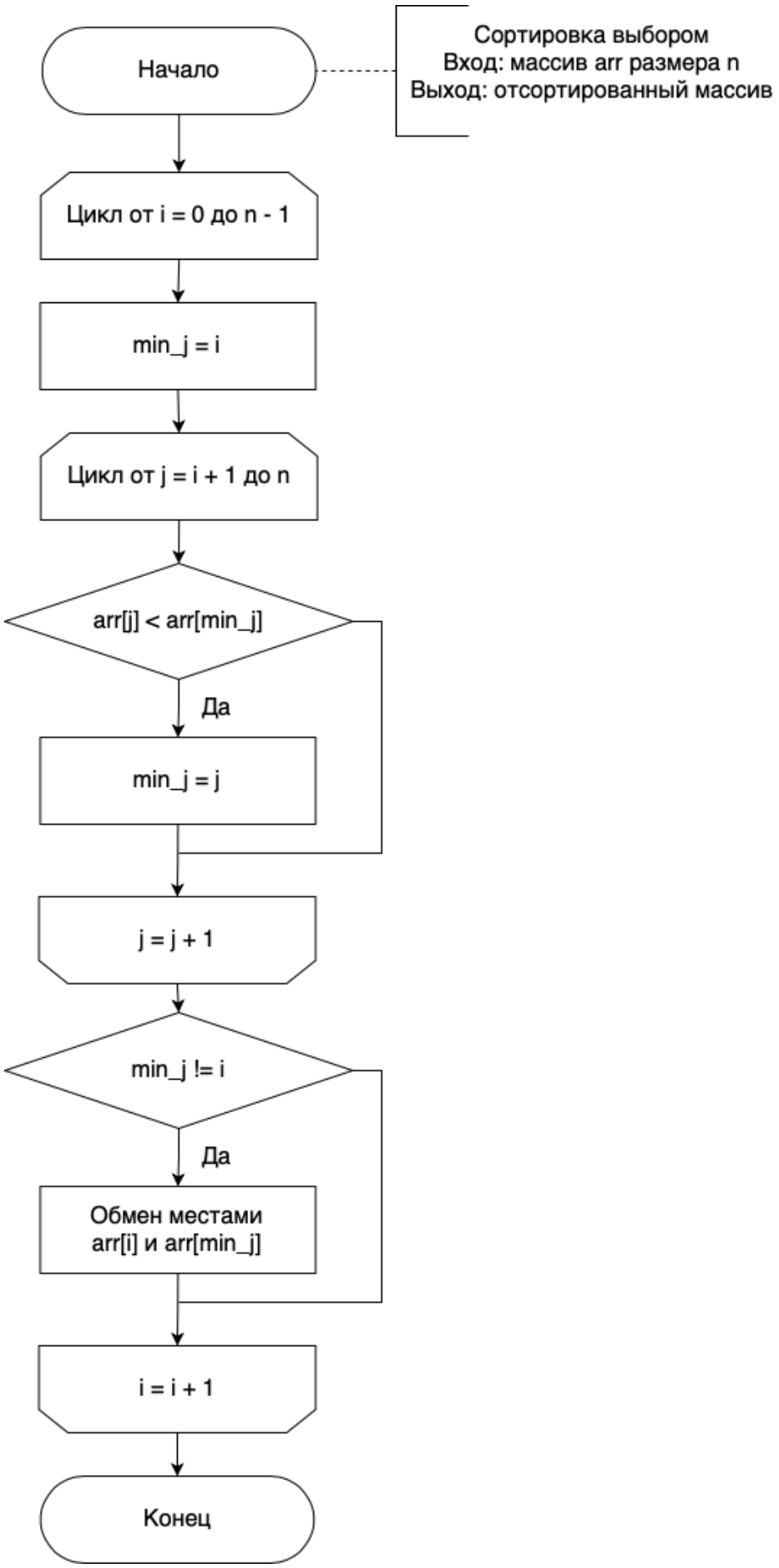

Рисунок 2.2 – Схема сортировки массива выбором

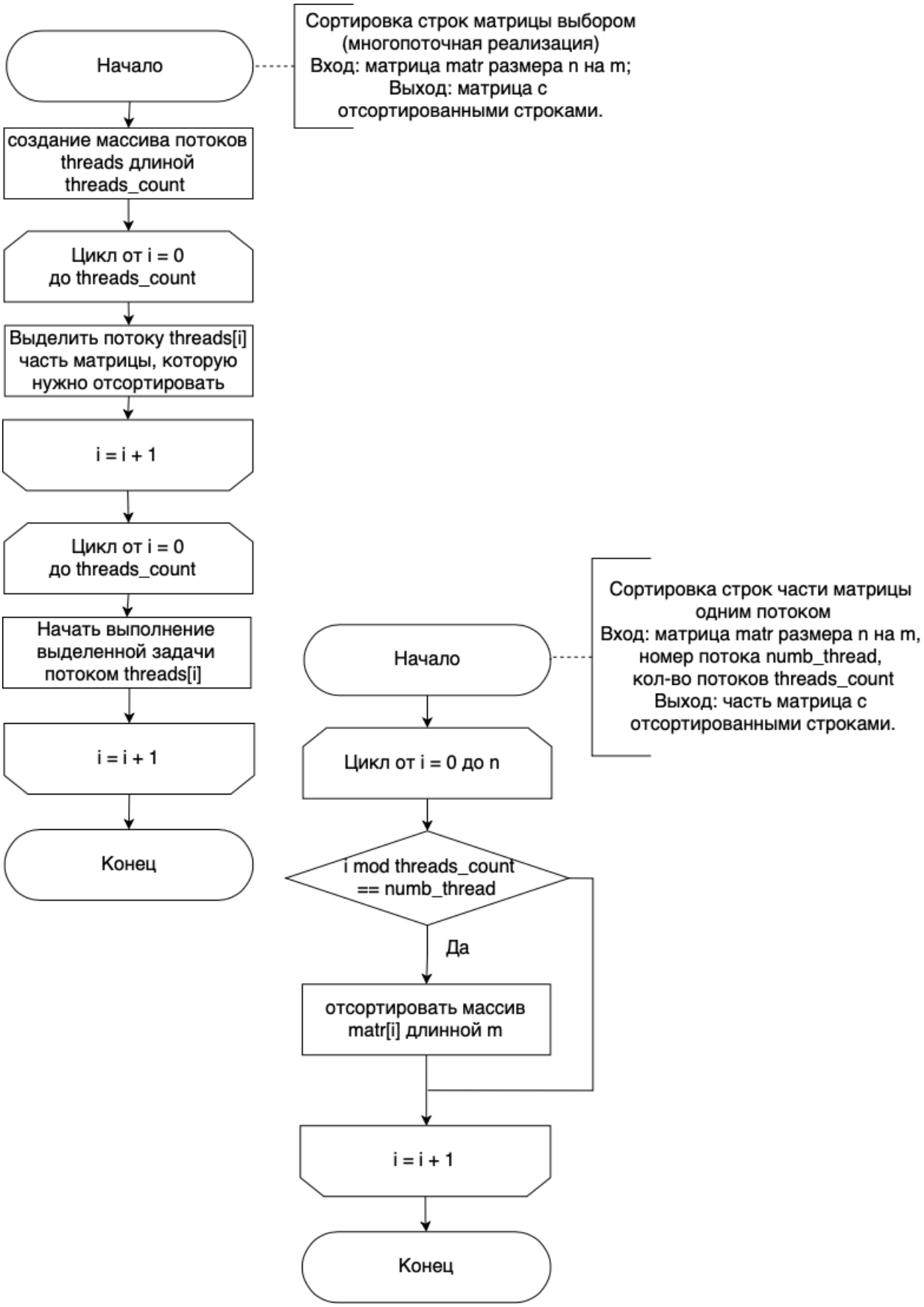

Рисунок 2.3 – Схема многопоточной реализации сортировки строк матроицы

2.2 Классы эквивалентности

Выделенные классы эквивалентности для тестирования:

- кол-ва строк матрицы <= 0;
- кол-ва столбцов матрицы <= 0;
- кол-во строк и столбцов матрицы - целые числа;
- кол-во строк и столбцов матрицы равно 1;
- строка матрицы содержит одинаковые числа;
- все числа в строке матрицы уже отсортированны;
- все числа в строке матрицы отсортированны в обратном порядке;
- кол-во строк матрицы кратно кол-ву потоков;
- кол-во строк матрицы некратно кол-ву потоков.

2.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

- кол-во строк в матрице - целое число типа int;
- кол-во столбцов в матрице - целое число типа int;
- матрица - двумерный массив типа int.

2.4 Структура ПО

ПО будет состоять из следующих модулей:

- *main.cpp* - файл, содержащий функцию *main*;
- *matrix.cpp* - файл, содержащий функции для работы с матрицами;
- *compare.cpp* - файл, в котором содержатся функции для замера времени работы алгоритмов;
- *read.cpp* - файл, в котором содержатся функции ввода данных;
- *sort.cpp* - файл, в котором содержатся функции для сортировки строк матрицы (с однопоточной и многопоточной реализациями);
- *alloc_free_memory.cpp* - файл, в котором содержатся функции для выделения и очищения памяти;
- *errors.h* - файл, в котором содержатся классы для всех ошибок, которые могут возникнуть во время работы программы;
- *color.h* - файл, который содержит макросы для цветного вывода результата работы программы в консоль.

Каждый файл с расширением *.cpp* имеет вспомогательный файл с расширением *.hpp*, где содержатся объявления функций, инклуды и дефайны.

2.5 Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов сортировки строк матрицы, выбраны используемые типы данных, выделены классы эквивалентности для тестирования, а также была описана структура ПО.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинги кода, а также функциональные тесты.

3.1 Требования к программному обеспечению

- входные данные - кол-во строк и столбцов матрицы *matr* должно быть > 0, все элементы матрицы имеют тип *int*;
- выходные данные - матрица *matr* с отсортированными по возрастанию строками.

3.2 Средства реализации

В данной работе для реализации был выбран язык программирования *C++* [2]. Выбор обсуловлен наличием опыта работы с ним. Время работы было замерено с помощью функции *std::chrono::system_clock::now()* [3].

3.3 Листинги кода

В листингах 3.1 - 3.4 представлены схемы рассматриваемых алгоритмов сортировки строк матрицы.

Листинг 3.1 – Функция алгоритма сортировки массива вставками по возрастанию

```
1  void select_sort(int *arr, int n)
2  {
3      int min_j, tmp;
4
5      for (int i = 0; i < n - 1; i++)
6      {
7          min_j = i;
8
9          for (int j = i + 1; j < n; j++)
10         {
11             if (arr[j] < arr[min_j])
12             {
13                 min_j = j;
14             }
15         }
16
17         if (min_j != i)
18         {
19             tmp = arr[i];
20             arr[i] = arr[min_j];
21             arr[min_j] = tmp;
22         }
23     }
24 }
```

Листинг 3.2 – Функция однопоточной реализации сортировки выбором строк матрицы

```
void select_sort_one_thread(matrix_h &matr)
{
    for (int i = 0; i < matr.n; i++)
    {
        select_sort(matr.matrix[i], matr.m);
    }
}
```

Листинг 3.3 – Функция многопоточной реализации сортировки выбором строк матрицы

```
void select_sort_many_threads(matrix_h &matr, int threads_count)
{
    std::vector<std::thread> threads(threads_count);

    for (int i = 0; i < threads_count; i++)
    {
        threads[i] = std::thread(sort_part_matrix, std::ref(matr),
            i, threads_count);
    }

    for (int i = 0; i < threads_count; i++)
    {
        threads[i].join();
    }
}
```

Листинг 3.4 – Функция сортировки строк части матрицы для отдельного потока

```
void sort_part_matrix(matrix_h &matr, int numb_thread, int
    threads_count)
{
    for (int i = 0; i < matr.n; i++)
    {
        if (i % threads_count == numb_thread)
            select_sort(matr.matrix[i], matr.m);
    }
}
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для однопоточной и многопоточной реализаций сортировки выбором строк матрицы. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Кол-во строк	Кол-во столбцов	Сама матрица	Ожидаемый результат
0	1		Сообщение об ошибке
1	-1		Сообщение об ошибке
k	6		Сообщение об ошибке
1	1	$\begin{pmatrix}5\end{pmatrix}$	$\begin{pmatrix}5\end{pmatrix}$
1	3	$\begin{pmatrix}8 & 8 & 8\end{pmatrix}$	$\begin{pmatrix}8 & 8 & 8\end{pmatrix}$
1	5	$\begin{pmatrix}1 & 2 & 3 & 4 & 5\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3 & 4 & 5\end{pmatrix}$
1	5	$\begin{pmatrix}5 & 4 & 3 & 2 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3 & 4 & 5\end{pmatrix}$
4	5	$\begin{pmatrix}17 & 97 & 11 & 52 & 46\\ 23 & 18 & 81 & 34 & 50\\ 78 & 78 & 44 & 18 & 20\\ 61 & 98 & 10 & 12 & 91\end{pmatrix}$	$\begin{pmatrix}11 & 17 & 46 & 52 & 97\\ 18 & 23 & 34 & 50 & 81\\ 18 & 20 & 44 & 78 & 78\\ 10 & 12 & 61 & 91 & 98\end{pmatrix}$
5	3	$\begin{pmatrix}17 & 97 & 11\\ 23 & 18 & 81\\ 78 & 78 & 44\\ 61 & 98 & 10\\ 11 & 33 & 22\end{pmatrix}$	$\begin{pmatrix}11 & 17 & 97\\ 18 & 23 & 81\\ 44 & 78 & 78\\ 10 & 61 & 98\\ 11 & 22 & 33\end{pmatrix}$

3.5 Вывод

В данном разделе были разработаны алгоритмы для однопоточной и многопоточной реализаций сортировки выбором строк матрицы, проведено тестирование, описаны средства реализации и требования к ПО.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее.

- Операционная система: macOS 11.5.2. [4]
- Память: 8 GiB.
- Процессор: 2,3 GHz 4-ядерный процессор Intel Core i5. [5]

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

```
                Меню

        1. Сортировка строк матрицы выбором (с однопоточностью)
        2. Сортировка строк матрицы выбором (с многопоточностью)
        3. Замеры времени
        0. Выход

        Выбор:
2

Введите кол-во строк матрицы: 4
Введите кол-во столбцов матрицы: 8

Заполненная рандомными числами матрица:

77 89 33 48 50 72 34 18
63 29 10 45 52 72 37 93
17 89 80 92 43 29 19 57
50 43 79 28 36 35 17 66

Матрица с отсортированными строками:

18 33 34 48 50 72 77 89
10 29 37 45 52 63 72 93
17 19 29 43 57 80 89 92
17 28 35 36 43 50 66 79

                Меню

        1. Сортировка строк матрицы выбором (с однопоточностью)
        2. Сортировка строк матрицы выбором (с многопоточностью)
        3. Замеры времени
        0. Выход

        Выбор:
3

Сортировка матрицы размером 100x100

  Потоков |  Время
---------------------
     1    | 0.001763
     2    | 0.000974
     4    | 0.000597
     8    | 0.000581
    16    | 0.000639
    32    | 0.000783
```

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Результаты замеров времени работы однопоточной и многопоточной реализаций алгоритма сортировки выбором строк матрицы приведены на рисунках 4.1 - 4.2. Замеры времени проводились в секундах и усреднялись для каждого набора одинаковых экспериментов.

Таблица 4.1 – Зависимость времени работы алгоритмов от кол-ва потоков

Кол-во потоков	Время работы для матрицы NxM		
	100x100	150x150	200x200
1	0.001472	0.004008	0.008690
2	0.000770	0.002127	0.004595
4	0.000488	0.001503	0.002507
8	0.000445	0.001324	0.002071
16	0.000488	0.001144	0.002152
32	0.000672	0.001240	0.002262

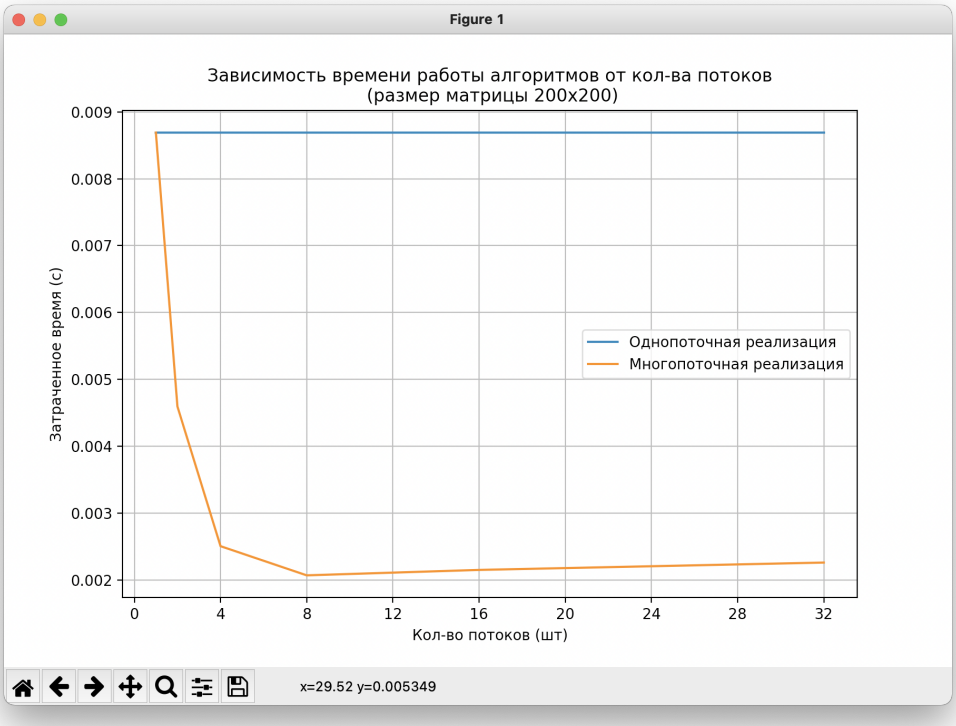

Рисунок 4.2 – Зависимость времени работы алгоритмов от кол-ва потоков (размер матрицы 200x200)

Таблица 4.2 – Зависимость времени работы алгоритмов от размера матрицы (в многопоточной реализации используется 8 потоков)

Размер матрицы	Время для типа реализации	
	Однопоточного	Многопоточного
100x100	0.001472	0.000445
110x110	0.001822	0.000552
120x120	0.002237	0.000755
130x130	0.002762	0.000937
140x140	0.003358	0.001029
150x150	0.004008	0.001324
160x160	0.004776	0.001319
170x170	0.005545	0.001393
180x180	0.006478	0.001912
190x190	0.007525	0.001833
200x200	0.008690	0.002071

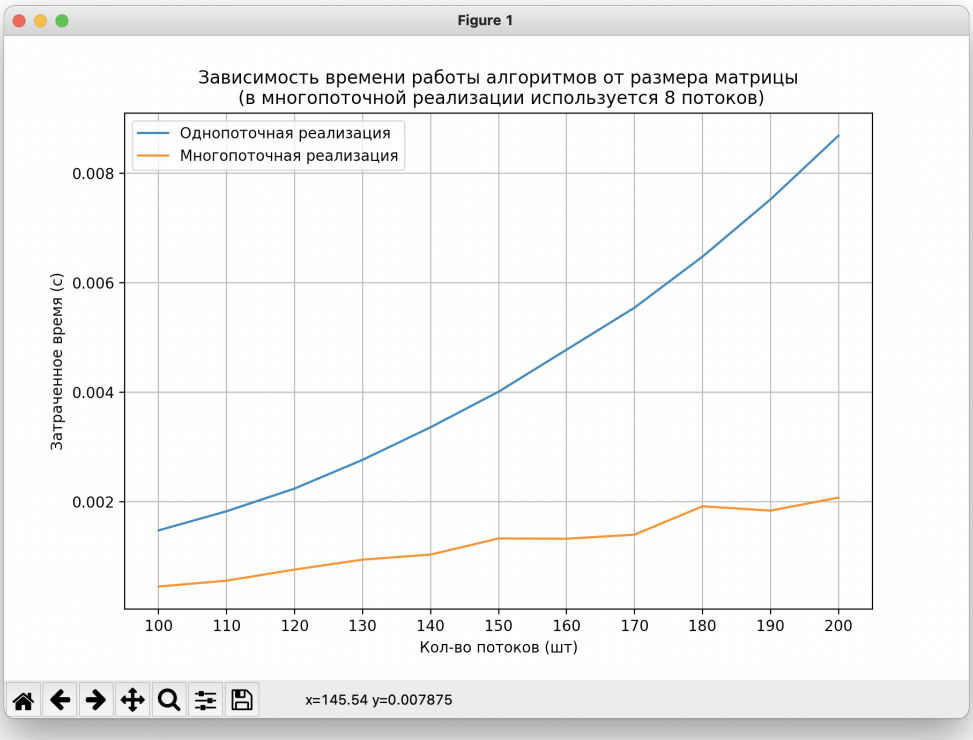

Рисунок 4.3 – Зависимость времени работы алгоритмов от размера матрицы (в многопоточной реализации используется 8 потоков)

4.4 Вывод

В этом разделе были указаны технические характеристики машины, на которой происходило сравнение времени работы однопоточной и многопоточной реализаций алгоритма сортировки выбором строк матрицы.

Многопоточная реализация алгоритма лучше всего работает на 8 потоках. Это связано с тем, что машина, на которой проводилось тестирование, имеет 8 логических ядер. При размере матрицы 200x200 8-ми поточная реализация работает на 17.4% быстрее 4-х поточной и на 3.8% быстрее 16-ти поточной. При размере матрицы 100x100 многопоточная реализация (8 потоков) работает на 230.1% быстрее однопоточной, а при размере 200x200 уже на 319.6% быстрее.

Заключение

Было экспериментально подтверждено различие во временной эффективности однопоточной и многопоточной реализаций сортировки выбором строк матрицы. В результате исследований можно сделать вывод о том, что многопоточная реализация лучше всего работает на 8 потоках, так как машина, на которой проводилось тестирование, имеет 8 логических ядер. Так, например, на матрице размером 200x200 многопоточня (8 потоков) реализация работает на 320% быстрее однопоточной.

В ходе выполнения данной лабораторной работы были решены следующие задачи:

- изучены основы параллельных вычислений;
- применены изученные основы для реализации многопоточного алгоритма сортировки строк матрицы;
- экспериментально подтверждено различие во временной эффективности однопоточной и многопоточной реализаций алгоритмов при помощи разработанного программного обеспечения на материале замеров процессорного времени;
- описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

Поставленная цель была достигнута.

Литература

[1] Сортировка выбором [Электронный ресурс]. Режим доступа: https://kvodo.ru/sortirovka-vyiborom-2.html (дата обращения: 14.10.2021).

[2] C++ — Типизированный язык программирования / Хабр[Электронный ресурс]. Режим доступа: https://habr.com/ru/hub/cpp/ (дата обращения: 20.10.2021).

[3] std::chrono::system_clock::now - cppreference.com [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono/system_clock/now (дата обращения: 20.10.2021).

[4] macOS Monterey - Apple(RU) [Электронный ресурс]. Режим доступа: https://www.apple.com/ru/macos/monterey/ (дата обращения: 14.10.2021).

[5] Intel [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/details/processors/core/i5.html (дата обращения: 14.10.2021).