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 «Анализ алгоритмов»

Тема Параллельные вычисления на основе нативных потоков

Студент Динь Вьет Ань

Группа ИУ7И-54Б

Оценка (баллы)

Преподаватель Волкова Л.Л., Строганов Ю.В.

Москва — 2023 г.

Оглавление

Введение

В настоящее время компьютерные системы оперируют большими объемами данных. Над этими данными проводится большой объем различного рода вычислений. Для того, чтобы они выполнянлись быстрее, было придумано параллельное программирование.

Его суть заключается в том, чтобы относительно равномерно разделять нагрузку между потоками ядра. Каждое из ядер процессора может обрабатывать по одному потоку, поэтому когда количество потоков на ядро становится больше, происходит квантование времени. Это означает, что на каждый процесс выделяется фиксированная величина времени (квант), после чего в течение кванта обрабатывается следующий процесс. Таким образом создается видимость параллельности. Тем не менее, данная оптимизация может сильно ускорить вычисления.

Целью данной лабораторной работы является получение навыков параллельного программирования. Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать основы параллельных вычислений;

— описать последовательный и параллельный алгоритмы;

— составить схемы данных алгоритмов;

— реализовать разработанные версии алгоритма;

— провести сравнительный анализ времени работы реализаций;

— описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будет рассмотрено понятие многопоточности и представлен алгоритм варианта задания.

1.1 Многопоточность

Многопоточность [1] — способность центрального процессора или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой.

Процесс — это программа в ходе своего выполнения. Когда мы выполняем программу или приложение, запускается процесс. Каждый процесс состоит из одного или нескольких потоков.

Поток — это сегмент процесса. Потоки представляют собой исполняемые сущности, которые выполняют задачи, стоящие перед исполняемым приложением. Процесс завершается, когда все потоки заканчивают выполнение.

Каждый поток в процессе — это задача, которую должен выполнить процессор. Большинство процессоров сегодня умеют выполнять одновременно две задачи на одном ядре, создавая дополнительное виртуальное ядро. Это называется одновременная многопоточность или многопоточность $Hyper - Threading$, если речь о процессоре от Intel.

Эти процессоры называются многоядерными процессорами. Таким образом, двухъядерный процессор имеет 4 ядра: два физических и два виртуальных. Каждое ядро может одновременно выполнять только один поток.

Как упоминалось выше, один процесс содержит несколько потоков, и одно ядро процессора может выполнять только один поток за единицу времени. Если мы пишем программу, которая запускает потоки последовательно, то есть передает выполнение в очередь одного конкретного ядра процессора, мы не раскрываем весь потенциал многоядерности. Остальные ядра просто стоят без дела, в то время как существуют задачи, которые необходимо выполнить. Если мы напишем программу таким образом, что она создаст несколько потоков для отнимающих много времени независимых

функций, то мы сможем использовать другие ядра процессора, которые в противном случае пылились бы без дела. Можно выполнять эти потоки параллельно, тем самым сократив общее время выполнения процесса.

1.2 Алгоритм варианта задания

Задание требует найти первое число в матрице, которое больше среднего геометрического, и заменить его на максимальный элемент матрицы. Если невозможно найти среднее геометрическое число или нет элемент, большего среднего геометрического, сообщить на экране.

Чтобы решить задание, сначала нужно найти среднее геометрическое матрицы. Затем необходимо найти максимальный элемент и положение первого элемента больше среднего геометрического. И, наконец, замените этот элемент на самый максимальный элемент. Процесс нахождения максимального элемента и элемента, большего среднего геометрического, может использовать параллельные вычисления.

Вывод

В данном разделе было рассмотрено понятие многопоточности и был представлен алгоритм варианта задания.

2 Конструкторская часть

В этом разделе будут приведены требования к вводу и программе, а также схемы последовательного и параллельного алгоритмов.

2.1 Требования к вводу

На вход подаются два числа — размер матрицы и матрица действительных чисел.

2.2 Требования к программе

Вывод программы — матрица с заменой или сообщение в случае, если среднее геометрическое число не может быть найдено или нет элементов больше среднего геометрического.

2.3 Разработка алгоритмов

На рисунках 2.1, 2.2 представлены схемы последовательного и параллельного алгоритмов.

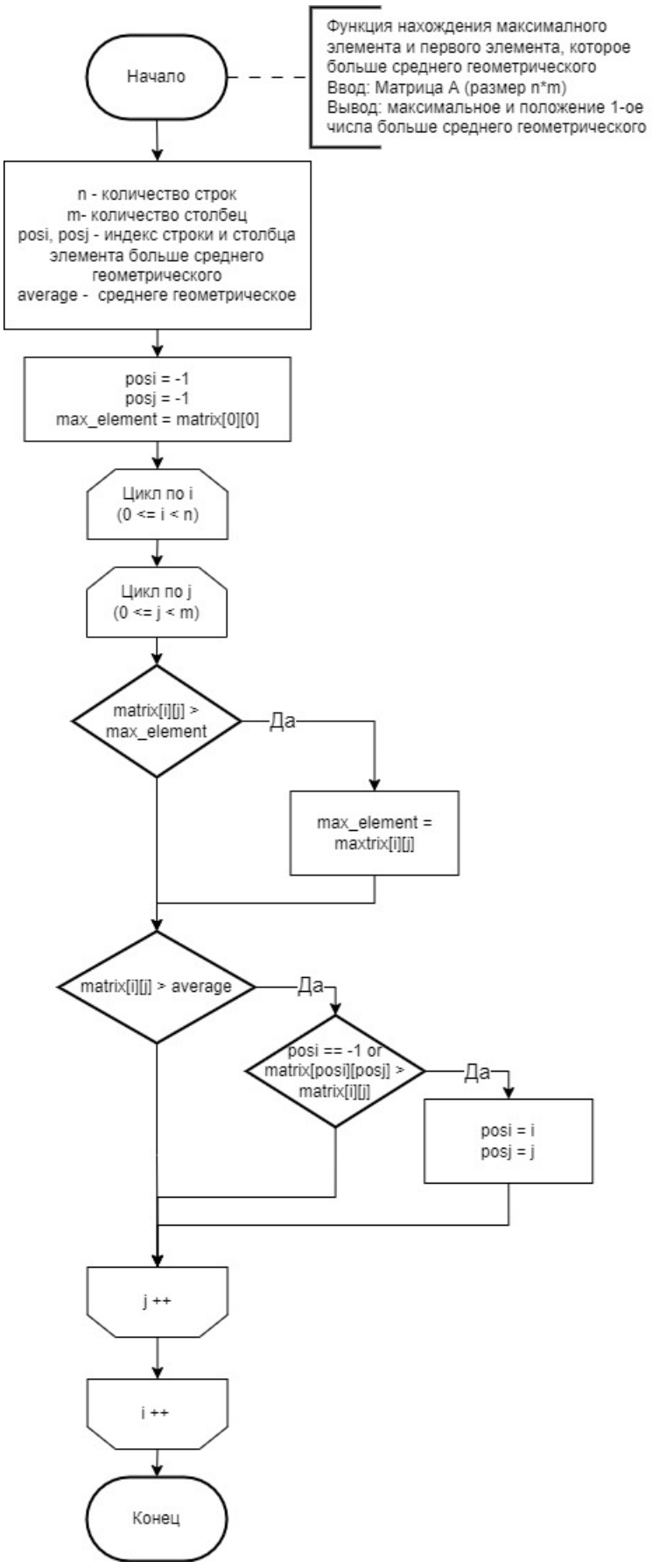

Рисунок 2.1 – Схема последовательного алгоритма

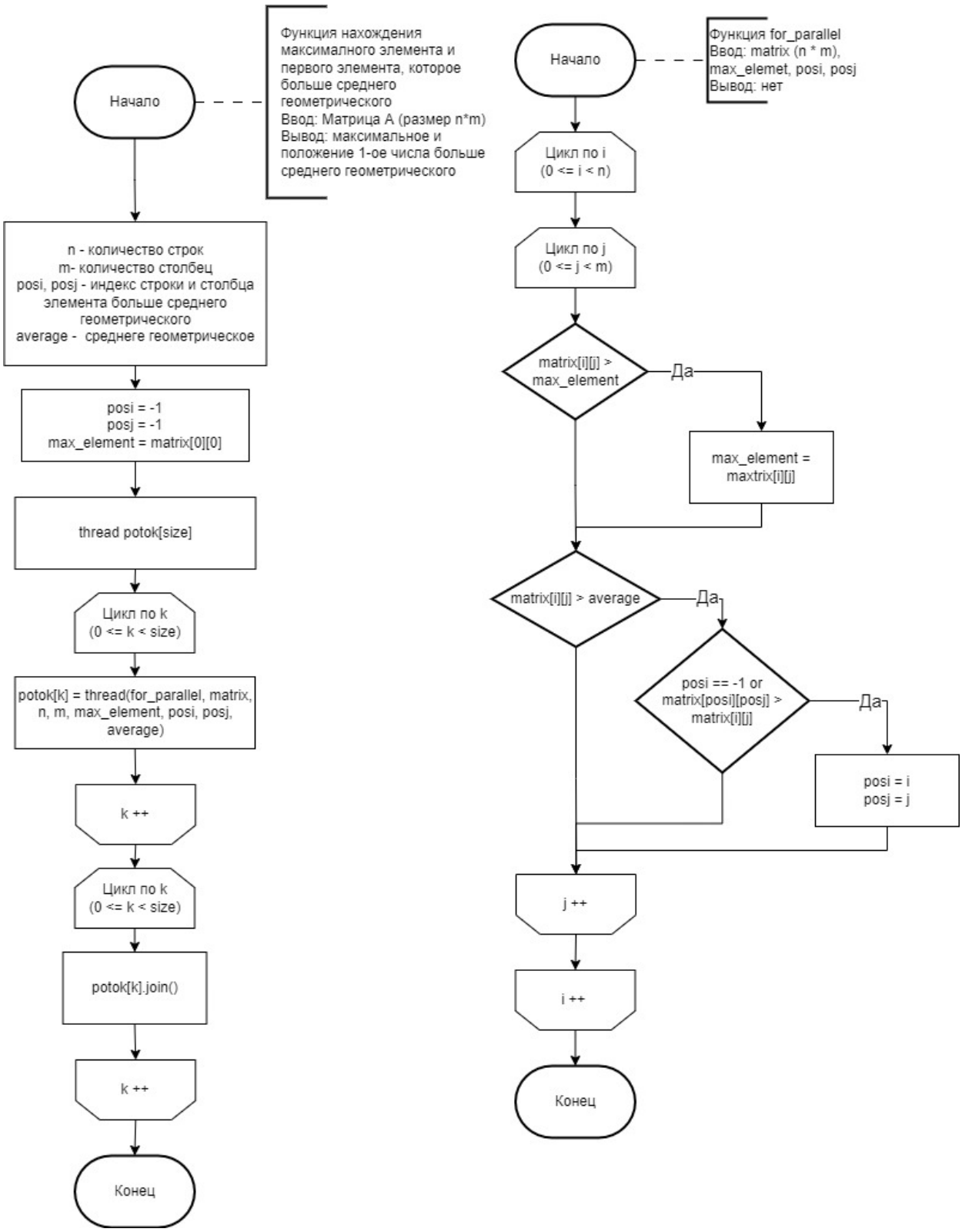

Рисунок 2.2 – Схема параллельного алгоритма

Вывод

В данном разделе были приведены требования к вводу и программе, а также построены схемы последовательного и параллельного алгоритмов.

3 Технологическая часть

В данном разделе были приведены средства реализации и листинги кода.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования C++ [2]. Данный язык имеет все небходимые инструменты для решения поставленной задачи.

3.2 Сведения о модулях программы

Программа состоит из одного модуля — *main.cpp* — файл, содержащий код последовательного и параллельного алгоритмов и вывода замеренного процессорного времени.

3.3 Реализация алгоритмов

В листинге 3.1 и 3.2 представлена реализация последовательного и параллельного алгоритма.

Листинг 3.1 – Последовательный алгоритм

```
1  void find_standard(double **matrix, int n, int m, double
       *max_element, int *posi, int *posj, double average)
2  {
3      *max_element = matrix[0][0];
4      *posi = -1;
5      *posj = -1;
6      for (int i = 0; i < n; i++)
7          for(int j = 0; j < m; j++)
8          {
9              if (*max_element < matrix[i][j])
10                 *max_element = matrix[i][j];
11             if (matrix[i][j] > average)
```

```
12                  if (*posi == -1)
13                  {
14                      *posi = i;
15                      *posj = j;
16                  }
17                  else if (matrix[*posi][*posj] > matrix[i][j])
18                  {
19                      *posi = i;
20                      *posj = j;
21                  }
22          }
23  }
```

Листинг 3.2 – Параллельный алгоритм

```
1  void for_parallel(double **matrix, int n, int m, double
       *max_element, int *posi, int *posj, double average)
2  {
3      for (int i = 0; i < n; i++)
4          for(int j = 0; j < m; j++)
5          {
6              if (*max_element < matrix[i][j])
7                  *max_element = matrix[i][j];
8              if (matrix[i][j] > average)
9                  if (*posi == -1)
10                 {
11                     *posi = i;
12                     *posj = j;
13                 }
14                 else if (matrix[*posi][*posj] > matrix[i][j])
15                 {
16                     *posi = i;
17                     *posj = j;
18                 }
19         }
20 }
21
22 void find_parallel(double **matrix, int n, int m, double
       *max_element, int *posi, int *posj, double average, int size)
23 {
24     *posi = -1;
25     *posj = -1;
```

```
26      *max_element = matrix[0][0];
27      thread potok[size];
28      for (int i = 0; i < size; i++)
29          potok[i] = thread(for_parallel, matrix, n, m,
                max_element, posi, posj, average);
30      for (int i = 0; i < size; i++)
31          potok[i].join();
32  }
```

3.4 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы изменения матрицы действительных чисел. Тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица A	Результат
$\begin{pmatrix} 1 & 1 \\ 27 & 729 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 729 & 729 \end{pmatrix}$
$\begin{pmatrix} 5 & -1 \end{pmatrix}$	Can't find average of matrix
$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	Can't find number greater than average of matrix
$\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 3 \\ 3 & 3 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 6 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$

Вывод

В этом разделе была представлена реализация последовательного и параллельного алгоритмов Тестирование показало, что алгоритмы реализованы правильно и работают корректно.

4 Исследовательская часть

В данном разделе приводятся результаты замеров затрат реализаций алгоритмов по процессорному времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

— операционная система Window 10 Home Single Language;

— память 8 Гб;

— процессор 11th Gen Intel(R) Core(TM) i5-1135G7 2.42 ГГц, 4 ядра.

Во время замера устройство было подключено к сети электропитания, нагружено приложениями окружения и самой системой замера.

4.2 Время выполнения реализации алгоритмов

Для замера процессорного времени использовалась функция *std::chrono::system_ clock::now(...)* из библиотеки *chrono* [3] на C++. Функция возвращает процессорное время типа float в секундах.

Контрольная точка возвращаемого значения не определна, поэтому допустима только разница между результатами последовательных вызовов.

Замеры времени для каждого размера матрицы проводились 200 раз. В качестве результата взято среднее время работы алгоритма на данном размере. При каждом запуске алгоритма, на вход подавались случайно сгенерированные матриц. Тестовые пакеты создавались до начала замера времени.

В таблице 4.1 приведены результаты замеров времени работы реализаций последовательного и параллельного алгоритмов при 4 потоков.

Таблица 4.1 – Результат замеров времени (мкс)

Размер	Без многоточности	4 потока
100x100	0.210	0.120
200x200	0.195	0.105
300x300	0.245	0.230
500x500	0.760	0.430
750x750	1.805	0.815
1000x1000	2.540	1.290
2000x2000	9.085	4.020
5000x5000	64.065	44.95

В таблице 4.2 приведены результаты замеров времени работы реализации параллельного алгоритмов на квадратной матрице размером 2000.

Таблица 4.2 – Результат замеров времени

Количество потоков	Время
1	9.040
2	4.945
4	3.445
8	3.500
16	3.845

Вывод

В данном разделе было произведено сравнение времени выполнения реализации алгоритма при последовательной реализации и многопоточности. Резуьтат показал, что выгоднее всего по времени использовать столько потоков, сколько у процессора логических ядер.

Заключение

В ходе лабораторной работы поставленная ранее цель была достигнута: были получены навыки организации параллельных вычислений. Выяснилось, что при работе с матрицами размерами до 2000x2000 элементов наиболее рационально использовать именно многопоточную реализацию алгоритма. Также в ходе выполнения лабораторной работы были выполнены все задачи:

— описаны основы параллельных вычислений;

— описаны последовательный и параллельный алгоритмы;

— составлены схемы данных алгоритмов;

— реализованы разработанные версии алгоритма;

— проведен сравнительный анализ времени работы реализаций;

— подготовлен отчет о лабораторной работе.

Список использованных источников

1. Что такое многопоточность [Электронный ресурс]. Режим доступа: https://ru.coursera.org/lecture/ios-multithreading/chto-takoie-mnoghopotochnost-4MMgN (дата обращения: 20.03.2023).

2. Рэнди Дэвис Стефан. C++ для чайников. Для чайников. Вильямс, 2018. с. 400.

3. Date and time utilities [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 20.03.2023).