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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

ОТЧЕТ

по лабораторной работе № 4

Название: Разработка параллельных алгоритмов

Дисциплина: Анализ алгоритмов

Студент ИУ7-52Б _____ Короткая В. М.

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Волкова Л.Л.

(Подпись, дата) (И.О. Фамилия)

Москва, 2021

Оглавление

Введение

Говоря о параллелизме в контексте компьютеров, имеется в виду, что одна и та же система выполняет несколько независимых операций параллельно, а не последовательно.

Существует две основных причины для использования параллелизма в приложении:

- разделение обязанностей;

- производительность.

Разделение обязанностей почти всегда приветствуется при разработке программ: если сгруппировать взаимосвязанные и разделить несвязанные части кода, то программа станет проще для понимания и тестирования и, стало быть, будет содержать меньше ошибок.

Существует два способа применить распараллеливание для повышения производительности.

Первый, самый очевидный, разбить задачу на части и запустить их параллельно, уменьшив тем самым общее время выполнения. Это распараллеливание по задачам. Разбиение можно формулировать как в терминах обработки: один поток выполняет одну часть обработки, другой – другую, так и в терминах данных: каждый поток выполняет одну и ту же операцию, но с разными данными. Последний вариант называется распараллеливание по данным.

Второй способ применения распараллеливания для повышения производительности – воспользоваться имеющимся параллелизмом для решения более крупных задач, например, обрабатывать не один файл за раз, а сразу два, десять или двадцать. Это по сути дела пример распараллеливания по данным, так как одна и та же операция производится над несколькими наборами данных одновременно, но акцент немного иной.

Целью данной работы является разработка и исследования параллельных алгоритмов умножения матриц.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить основные методы параллельных вычислений,

- реализовать последовательный алгоритм умножения матриц,
- реализовать распареллелиный алгоритм умножения матриц,
- сравнить временные характеристики алгоритмов эксперементально.

1. Аналитическая часть

В данном разделе будут рассмотрены алгоритм умножения матриц и идея его параллельной реализации.

1.1. Математическое описание операции умножения матриц

Умножение матриц – операция на матрицами $A[M * N]$ и $B[N * Q]$. Результатом операции является матрица C размерами M * Q, в которой каждый элемент $c_{i,j}$ задаётся формулой 1.1.

$$c_{i,j} = \sum_{k=1}^{n}(a_{i,k} \cdot b_{k,j}) \tag{1.1}$$

1.2. Используемые алгоритмы

Стандартный алгоритм подразумевает циклическое сложение всех элементов вышеописанной суммы для получения каждого элемента матрицы C.

Параллелизм может быть достигнут за счёт выделения процессов, которые могут выполнятся независимо друг от друга. В данном случае вычисление каждого элемента C ведётся независимо друг от друга, поэтому в качестве параллельных алгоритмов выбраны параллельное вычисление элементов строк и параллельное вычисление элементов столбцов.

Вывод

Результатом аналитического раздела стало описано понятие операции умножения матриц и описаны используемые алгоритмы.

Входными данными реализуемого ПО являются:

- размерность первой матрицы - два натуральных числа;
- первая матрица - целочисленная;
- размерность второй матрицы - два натуральных числа;
- вторая матрица - целочисленная.

Выходными данными реализуемого ПО является результат алгоритмов умножения матриц т. е.:

- матрица (результат) - для классического умножения матриц;
- матрица (результат) - для параллельного алгоритма умножения матриц;

Ограничением для реализуемого ПО является - размерность вводимых матриц т. е. длинна строки первой матрицы должна совпадать с длинной колонки второй матрицы.

2. Конструкторская часть

В данном разделе представлены схемы алгоритмов. Так же будут описаны пользовательские структуры данных, приведены структура ПО и классы эквивалентности для тестирования реализуемого ПО.

2.1. Схемы алгоритмов

Стандартный алгоритм умножения

Данный алгоритм непосредственно использует вышеприведённую формулу. Для вычисления каждого элемента матрицы C совершается циклический обход k элементов из таблиц A и B.

Схема алгоритма приведена на рисунке 2.1

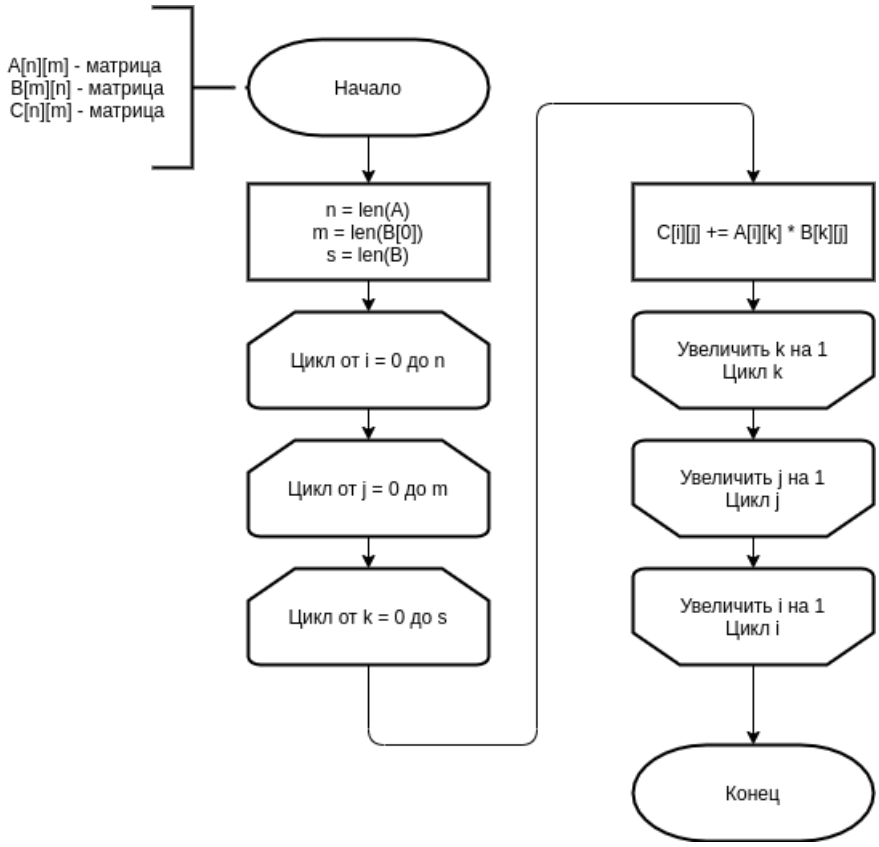

Рис. 2.1: Классический алгоритм умножения матриц.

,

Алгоритм умножения параллельный по строкам

Вычисление каждого элемента матрицы является независимым. Поэтому возможна следующая параллельная реализация данного алгоритма.

Пусть производится работа с T потоками. В таком случае, i-й поток будет производить вычисление строк $i, i+T, i+2T, \ldots, ((M-i)modT) \cdot T + i$.

Схема алгоритма приведена на рисунках 2.2 и 2.3

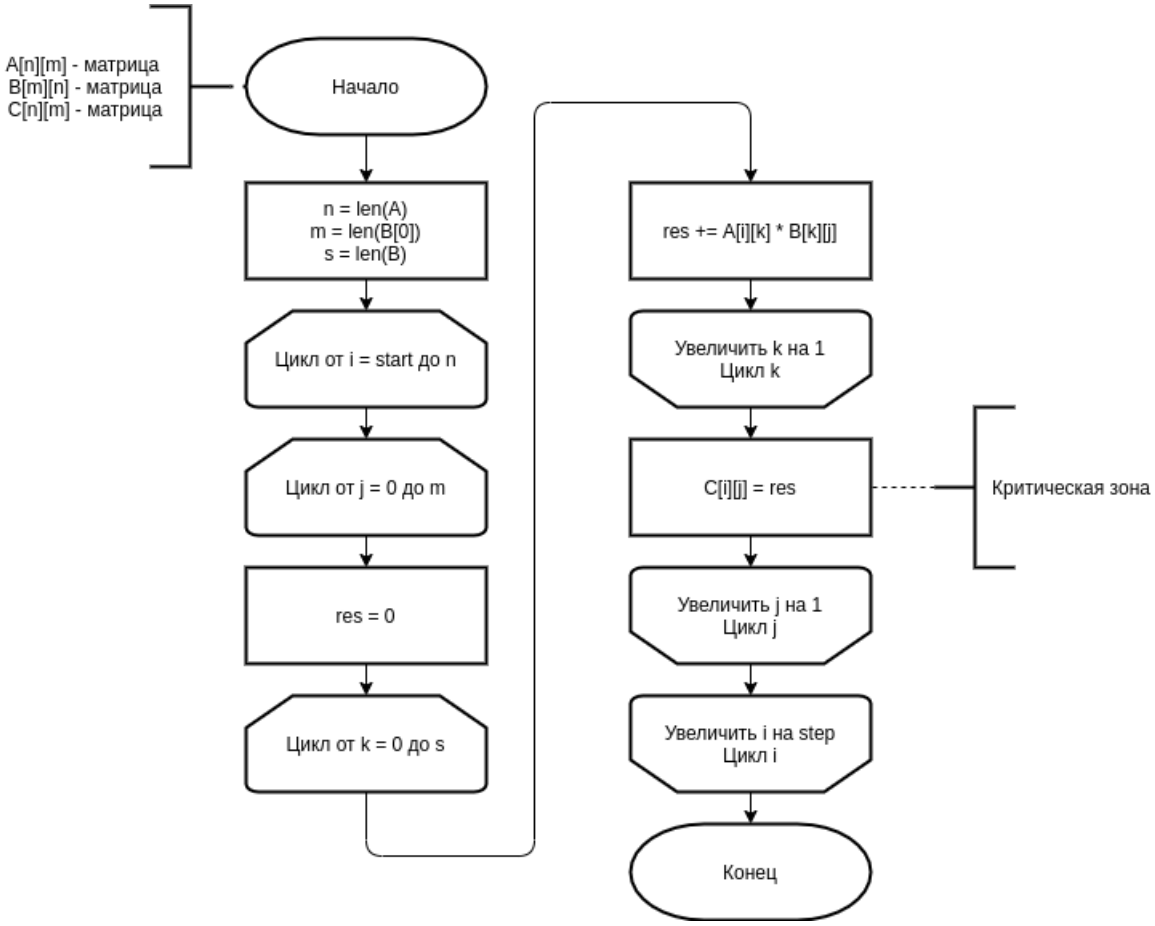

Рис. 2.2: Схема параллельного умножение матриц по строкам.(часть 1)

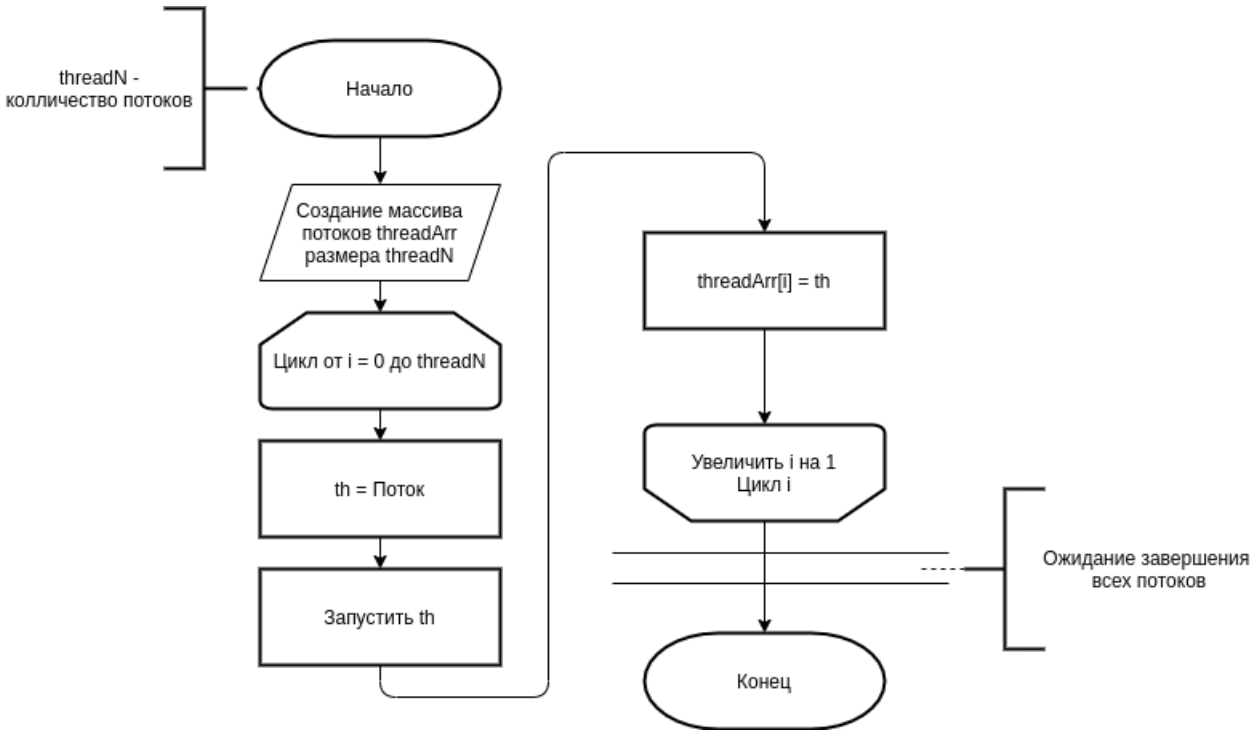

Рис. 2.3: Схема параллельного умножение матриц по строкам.(часть 2)

2.2. Алгоритм умножения параллельный по столбцам

В силу независимости вычислений каждого элемента, аналогично можно организовать и параллельное вычисление значений в столбцах матрицы C. В таком случае, i-й поток будет производить вычисление столбцов $i, i+T, i+2T, \ldots, ((Q-i)modT) \cdot T + i$

Схема алгоритма приведена на рисунках 2.4 и 2.5

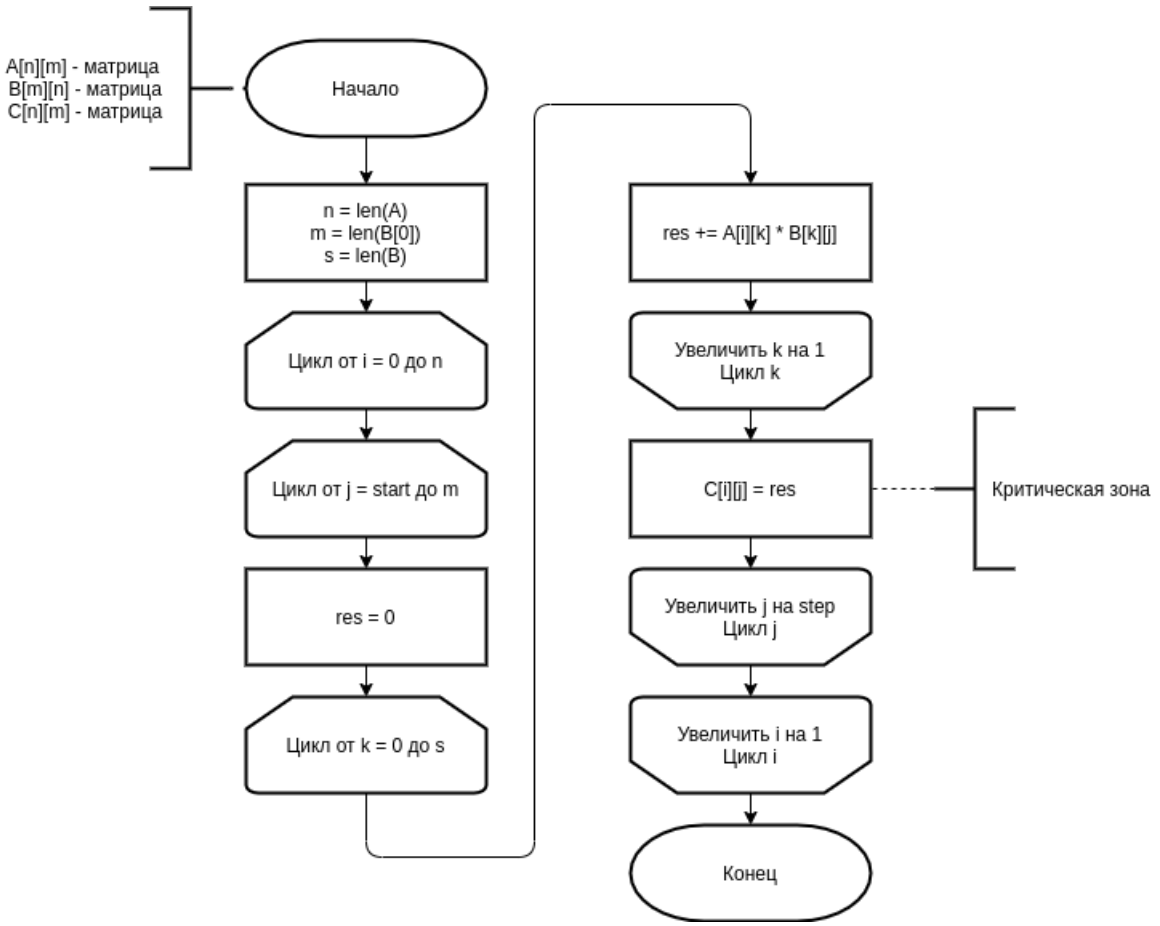

Рис. 2.4: Схема параллельного умножение матриц по столбцам.(часть 1)

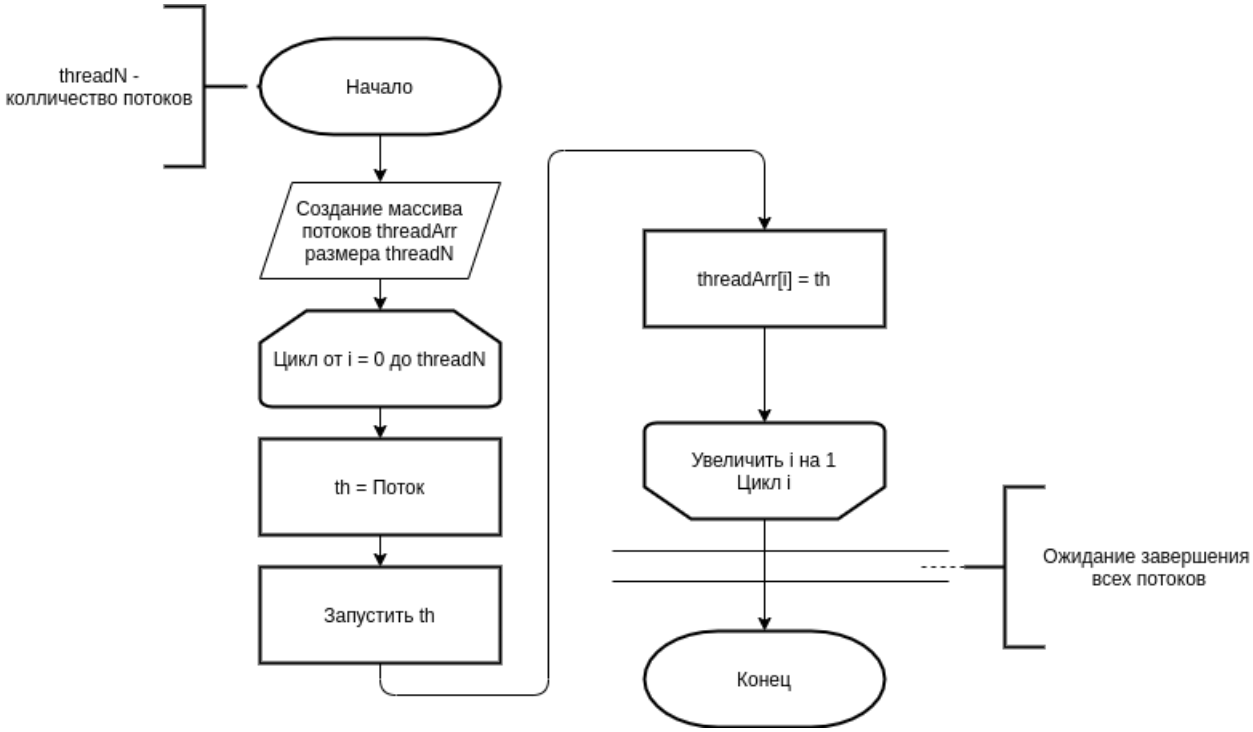

Рис. 2.5: Схема параллельного умножение матриц по столбцам.(часть 2)

2.3. Структура ПО

На рисунке 2.6 представлена диограмма классов.

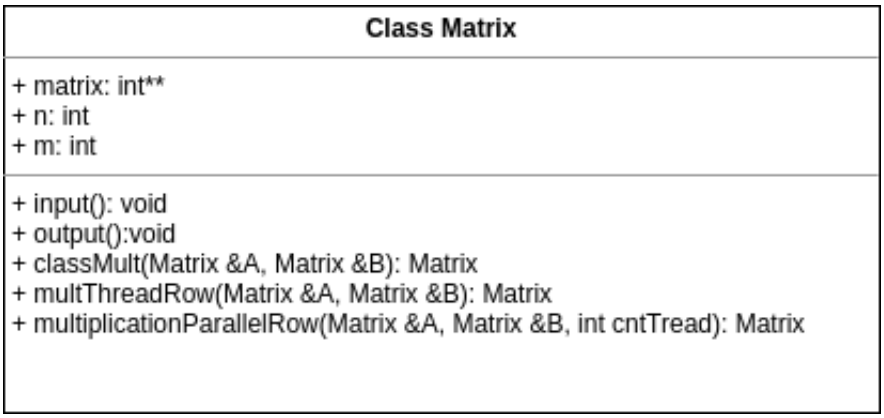

Рис. 2.6: Диаграмма классов реализуемого ПО

2.4. Описание структур данных

Для реализации данных алгоритмов, введем некоторые типы данных:

- matrixType - тип данных описывающий матрицу.

2.5. Тестирование

В рамках данной лабораторной работы были выделены следующие классы эквивалентности:

- входными данными являются две матрицы размерностью 1*1;
- входными данными являются две матрицы размерностью 1*n и n*1 соответственно;
- входными данными являются две матрицы квадратные матрицы равной размерностью;
- входными данными являются две матрицы размерностью n*k и к*т.

Для проверки работы программы будет осуществлено тестирование согласно классам эквивалентности.

Вывод

На основе теоритических данных, полученных из аналитического раздела, были построены схемы реализируемых алгоритмов.

Так же, было приведено описание вводимых типов данных.

3. Технологическая часть

В данном разделе приведены средства реализации, требования к ПО и листинги кода.

3.1. Средства реализации

В качестве языка программирования был выбран c++. Данный язык знаком и предостовляет все необходимые ресурсы. В качестве среды разработки я использовала Visual Studio Code, т.к. считаю его достаточно удобным и легким. Visual Studio Code подходит не только для Windows, но и для Linux, это еще одна причина, по которой я выбрала VS code, т.к. у меня установлена ОС fedora 34.

3.2. Сведенья о модулях программы

- main.cpp - файл, содержащий точку входа в программу;
- matrix.cpp - файл, содержащий реализацию алгоритмов умножения матриц.

3.3. Реализация алгоритмов

Листинг 3.1: Реализация алгоритма классического умножения матриц.

```
1  matrixType classicMultiplication(matrixType a,
2                                   matrixType b, int m, int n, int q)
3  {
4      matrixType c = createMatrix(m, q);
5      for (int i = 0; i < m; i++)
6      {
7          for (int j = 0; j < q; j++)
8          {
9              int res = 0;
10             for (int k = 0; k < n; k++)
11             {
12                 res += a[i][k] + b[k][j];
13             }
```

```
14                c[i][j] = res;
15            }
16        }
17        return c;
18    }
```

Листинг 3.2: Реализация параллельного алгоритма умножения матриц по столбцам.

```
1  void multThreadRow(matrixType& c, matrixType& a,
2      matrixType&b, int m, int n, int q, int start, int step)
3  {
4      for (int i = start; i < m; i += step)
5      {
6          for (int j = 0; j < q; j++)
7          {
8              int res = 0;
9              for (int k = 0; k < n; k++)
10             {
11                 res += a[i][k] * b[k][j];
12             }
13             mtx.lock();
14             c[i][j] = res;
15             mtx.unlock();
16         }
17     }
18 }
```

Листинг 3.3: Реализация функции распараллеливания по столбцам.

```
1  matrixType multiplicationParallelRow(matrixType a,
2      matrixType b, int m, int n, int q, int threadCnt)
3  {
4      matrixType c = createMatrix(m, q);
5      vector<thread> threadArray;
6      for (int i = 0; i < threadCnt; i++)
7      {
```

```
8             threadArray.push_back(thread(multThreadRow, ref(c),
9             ref(a), ref(b), m, n, q, i, threadCnt));
10        }
11        for (int i = 0; i < threadCnt; i++)
12        {
13            threadArray[i].join();
14        }
15        return c;
16    }
```

Листинг 3.4: Реализация параллельного алгоритма умножения матриц по строкам.

```
1  void multThreadColumn(matrixType& c, matrixType& a,
2      matrixType& b, int m, int n, int q, int start, int step)
3  {
4      for (int i = 0; i < m; i++)
5      {
6          for (int j = start; j < q; j += step)
7          {
8              int res = 0;
9              for (int k = 0; k < n; k++)
10             {
11                 res += a[i][k] * b[k][j];
12             }
13             mtx.lock();
14             c[i][j] = res;
15             mtx.unlock();
16         }
17     }
18 }
```

Листинг 3.5: Реализация функции распараллеливания по строкам.

```
1  matrixType multiplicationParallelColumn(matrixType a,
2     matrixType b, int m, int n, int q, int threadCnt)
3  {
```

```
4       matrixType c = createMatrix(m, q);
5       vector<thread> threadArray;
6       for (int i = 0; i < threadCnt; i++)
7       {
8           threadArray.push_back(thread(multThreadColumn,
9           ref(c), ref(a), ref(b), m, n, q, i, threadCnt));
10      }
11      for (int i = 0; i < threadCnt; i++)
12      {
13          threadArray[i].join();
14      }
15      return c;
16  }
```

3.4. Тестирование

В данном разделе будет приведена таблица с тестами (таблица 3.1).

Таблица 3.1: Таблица тестов

Матрица A	Матрица B	Результат
2 2 1 0 0 1	2 2 1 0 0 1	Ответ верный
3 2 2 3 1 0 2 2	2 4 2 2 1 9 4 2 8 1	Ответ верный
2 1 2 1	12 12	Ответ верный
2 2 1 0 0 1	1 1 0	Сообщение о неверном вводе размерностей
0 0	0 0	

Все тесты пройдены.

4. Исследовательская часть

В данном разделе будет произведено измерение временных характеристик.

4.1. Технические характеристики

Технические характеристики устройства на котором выполнялось исследование:

- процессор Intel® Core™ i5-10210U CPU @ 1.60GHz × 8;
- память 15.3 GiB;
- операционная система Fedora 34 (Workstation Edition) 64-bit.

4.2. Временные характеристики

Измерения процессорного времени проводятся на квадратных матрицах с размерами: 32, 100, 250, 500, 1000. Содержание матриц сгенерировано случайным образом. Изучается серия экспериментов с количеством потоков 1, 2, 3, 4, 8, 16, 32.

Для повышения точности, каждый замер производится 5 раз, за конечный результат берётся среднее арифметическое.

По результатам измерений процессорного времени можно составить таблицы 4.1 - 4.5.

Таблица 4.1: Результаты замеров процессорного времени при размере 32 (в миллисекундах)

Потоки	1	2	4	8	16	32
Многопоточно по строкам	0.24	0.29	0.36	0.58	1.04	2.02
Многопоточно по столбцам	0.24	0.30	0.35	0.59	1.03	1.97
Однопоточно	0.096					

Таблица 4.2: Результаты замеров процессорного времени при размере 100 (в миллисекундах)

Потоки	1	2	4	8	16	32
Многопоточно по строкам	5.52	2.73	2.80	2.78	3.09	3.51
Многопоточно по столбцам	4.87	3.14	2.87	2.89	3.15	3.57
Однопоточно	2.96					

Таблица 4.3: Результаты замеров процессорного времени при размере 250 (в секундах)

Потоки	1	2	4	8	16	32
Многопоточно по строкам	0.068	0.042	0.031	0.024	0.026	0.024
Многопоточно по столбцам	0.064	0.047	0.035	0.027	0.034	0.030
Однопоточно	0.051					

Таблица 4.4: Результаты замеров процессорного времени при размере 500 (в секундах)

Потоки	1	2	4	8	16	32
Многопоточно по строкам	0.61	0.38	0.26	0.20	0.21	0.20
Многопоточно по столбцам	0.62	0.42	0.28	0.23	0.24	0.24
Однопоточно	0.051					

Таблица 4.5: Результаты замеров процессорного времени при размере 1000 (в секундах)

Потоки	1	2	4	8	16	32
Многопоточно по строкам	8.22	4.63	2.64	2.18	2.22	2.27
Многопоточно по столбцам	8.33	5.64	3.05	2.62	2.82	2.89
Однопоточно	7.13					

Вывод

По результатам экспериментов можно заключить следующее:

- при относительно небольшом размере матриц (менее 100x100) исполь-

зование потоков для уменьшения времени исполнения нецелесообразно, так как накладные расходы времени на управление потоками и mutex-ами больше, чем выигрыш от параллельного выполнения выполнения вычислений;

- использование по крайней мере двух потоков даёт ощутимый выигрыш по времени по сравнению с однопоточной версией алгоритма;

- использование одного потока в многопоточных версиях алгоритма проигрывает по времени по сравнению с однопоточной версией алгоритма, что объясняется накладными расходами времени на управление потоками и mutex-ами;

- использование 16 и 32 потоков показывает результат по времени несколько хуже, чем при 8 потоках, из чего следует, что увеличение потоков даёт выигрыш по времени лишь до достижения определённого количества, так как появляются большие накладные затраты по времени для управления большим количеством потоков и mutex-ов;

- параллельные версии алгоритма выполняются за приблизительно одинаковое время при одном потоке. Однако, использование большего количества потоков выявляет, что многопоточность по строкам быстрее многопоточности по столбцам вплоть до 20%;

- наиболее быстродейственно алгоритм действует на 8 потоках, что равно количеству логических процессоров на испытуемом компьютере.

Заключение

В данной работе были рассмотрены параллельные алгоритмы умножения матриц.

Из проведённых экспериментов было выявлено, что наиболее быстродеиственным является использование количество потоков, которое совпадает с количеством логических процессоров процессора. Увеличение или уменьшение количества потоков ведёт к большему времени выполнения вычислений. Однако, использование потоков даёт выигрыш по времени работы только для относительно больших размеров матриц, иначе их использование лишь увеличит время вычислений за счёт накладных расходов. Также было установлено, что алгоритм, использующий многопоточность по строкам показывает себя несколько быстрее алгоритма с многопоточностью по столбцам.

Цель лабороторной работы достигнутаю Выполнены поставленные задачи:

- изучены основные методы параллельных вычислений,
- реализован последовательный алгоритм умножения матриц,
- реализован распареллелиный алгоритм умножения матриц,
- проведено сравнение временных характеристики алгоритмов эксперементально.

Список литературы

1. Дж. Макконнел. Анализ алгоритмов. Активный обучающий подход. – М.: Техносфера, 2017. – 267с.

2. Основы программирования на языках Си и C++ для начинающих[Электронный ресурс]. Режим доступа: http://cppstudio.com/ (дата обращения 10.10.2021)

3. LINUX.ORG.RU - Русскоязычная информация о ОС Linux[Электронный ресурс] Режим доступа://www.linux.org.ru/(дата обращения 25.10.2021)

4. Документация языка C++ 98 [Электронный ресурс], режим доступа: http://www.open-std.org/JTC1/SC22/WG21/ (дата обращения 10.12.2021)

5. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.