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**

О Т Ч Е Т

по лабораторной работе № 4

Название: Разработка параллельных алгоритмов

Дисциплина: Анализ алгоритмов

Студент ИУ7-52Б _____ Сучков А.Д.
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва, 2020

Оглавление

Введение **3**

1 Аналитическая часть **4**

1.1 Математическое описание операции умножения матриц 4

1.2 Используемые алгоритмы 4

1.3 Вывод 5

2 Конструкторская часть **6**

2.1 Стандартный алгоритм умножения 6

2.2 Алгоритм умножения параллельный по строкам 7

2.3 Алгоритм умножения параллельный по столбцам 8

2.4 Требования к программному обеспечению 10

2.5 Вывод 11

3 Технологическая часть **12**

3.1 Выбор языка программирования 12

3.2 Листинг кода 12

3.3 Результаты тестирования 16

3.4 Оценка времени 17

3.5 Вывод 20

4 Исследовательская часть **21**

4.1 Результаты экспериментов 21

4.2 Вывод 22

Заключение **24**

Список литературы **25**

Введение

В данной лабораторной работе реализуется и оценивается параллельный алгоритм классического умножения матриц.

Параллелизм – выполнение нескольких вычислений в различных потоках. При параллельном программировании в процессоре с многоядерной архитектурой несколько процессов могут выполнятся одновременно на разных ядрах. Это приводит к тому, что время выполнения параллельного алгоритма может быть ощутимо меньше, чем у его однопоточного аналога.

Стандартный алгоритм умножения матриц подразумевает проход по всем элементам результирующей матрицы C для вычисления их значений. Так как вычисление каждого элемента независимо, этот алгоритм подходит для реализации покоординатного параллелизма.

1. Аналитическая часть

Целью лабораторной работы является разработка и исследование параллельных алгоритмов умножения матриц.

Можно выделить следующие задачи лабораторной работы:

- описание понятия параллелизма и операции умножения матриц;
- описание и реализация непараллельного и двух параллельных версий алгоритма умножения матриц;
- проведение замеров процессорного времени работы алгоритмов при различном количестве потоков;
- анализ полученных результатов.

1.1. Математическое описание операции умножения матриц

Умножение матриц – операция на матрицами $A[M * N]$ и $B[N * Q]$ [2]. 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.

1.3. Вывод

Результатом аналитического раздела стало определение цели и задач работы, описано понятие операции умножения матриц и описаны используемые алгоритмы.

2. Конструкторская часть

В данном разделе рассмотрим описанные алгоритмы умножения матриц $A[M * N]$ и $B[N * Q]$ с результирующей матрицей умножения $C[M * Q]$.

2.1. 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

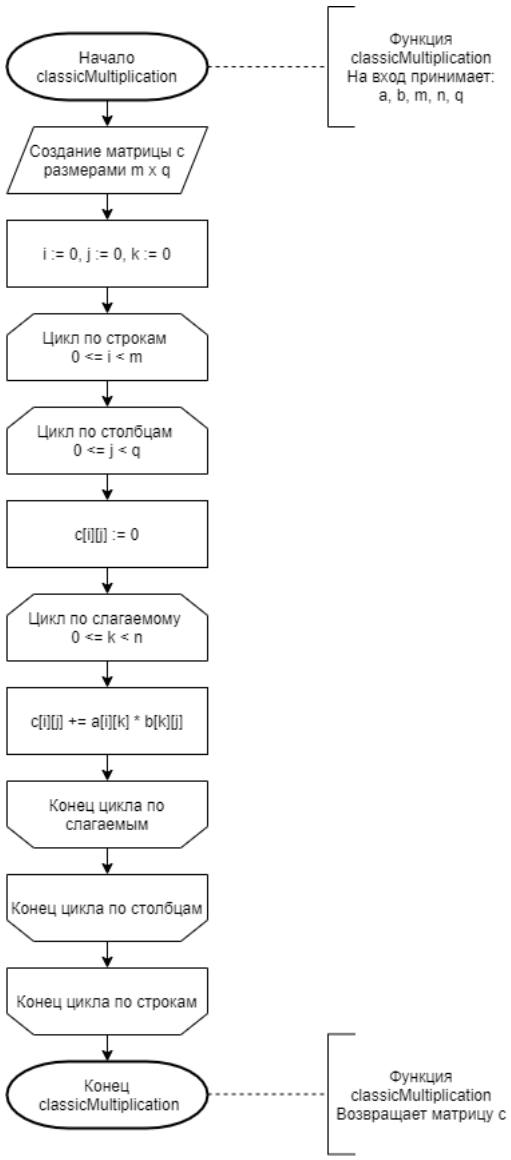

Рис. 2.1: Схема алгоритма классического умножения

2.2. 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

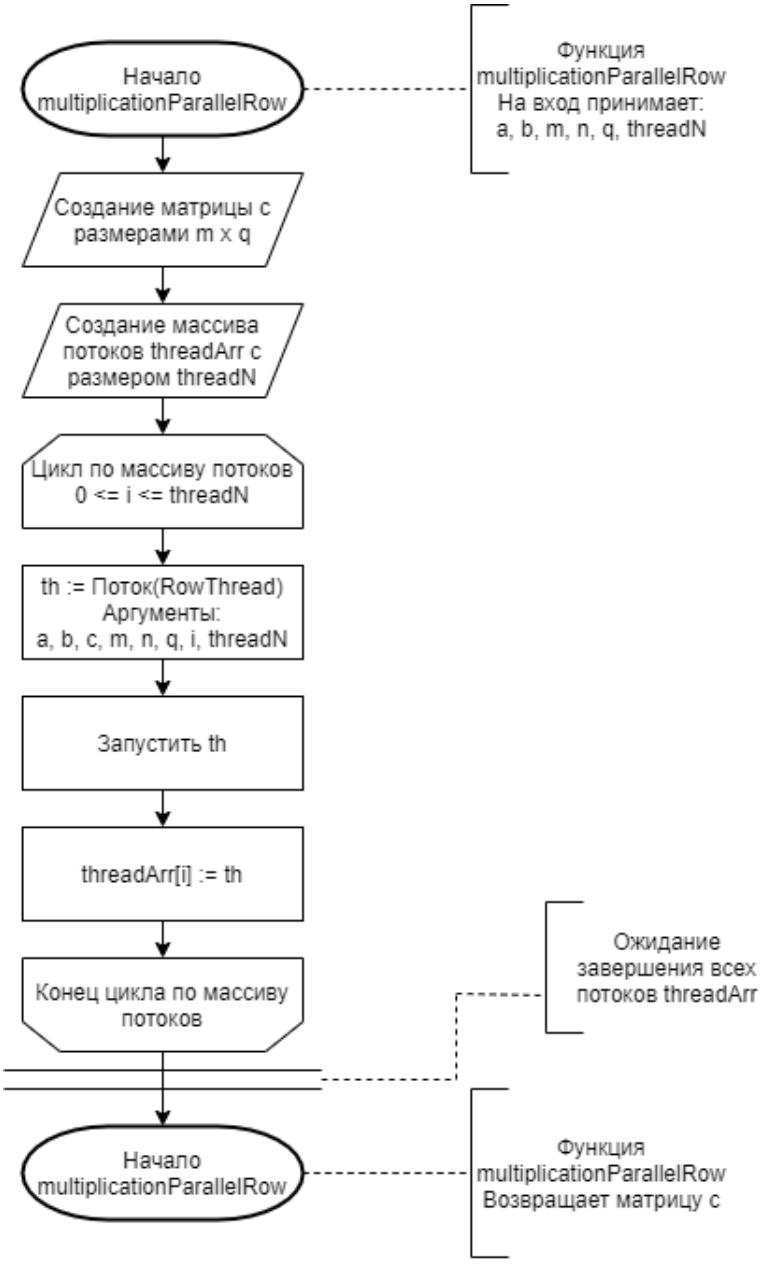

Рис. 2.2: Схема алгоритма параллельного умножения по строкам, часть 1

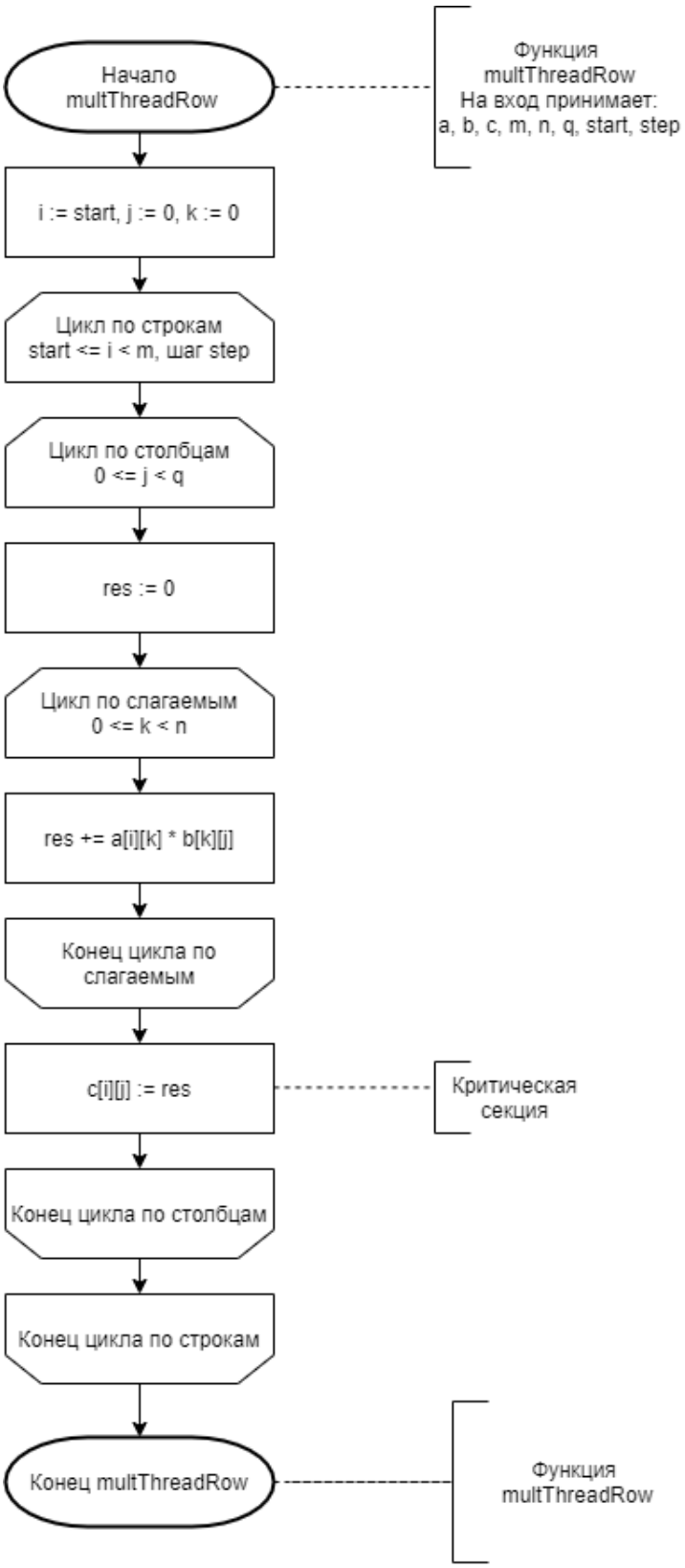

Рис. 2.3: Схема алгоритма параллельного умножения по строкам, часть 2

2.3. Алгоритм умножения параллельный по столбцам

В силу независимости вычислений каждого элемента, аналогично можно организовать и параллельное вычисление значений в столбцах матри-

цы C. В таком случае, i-й поток будет производить вычисление столбцов $i, i+T, i+2T, \ldots, ((Q-i) mod T) \cdot T + i$

Схема алгоритма приведена на рисунках 2.4 и 2.5

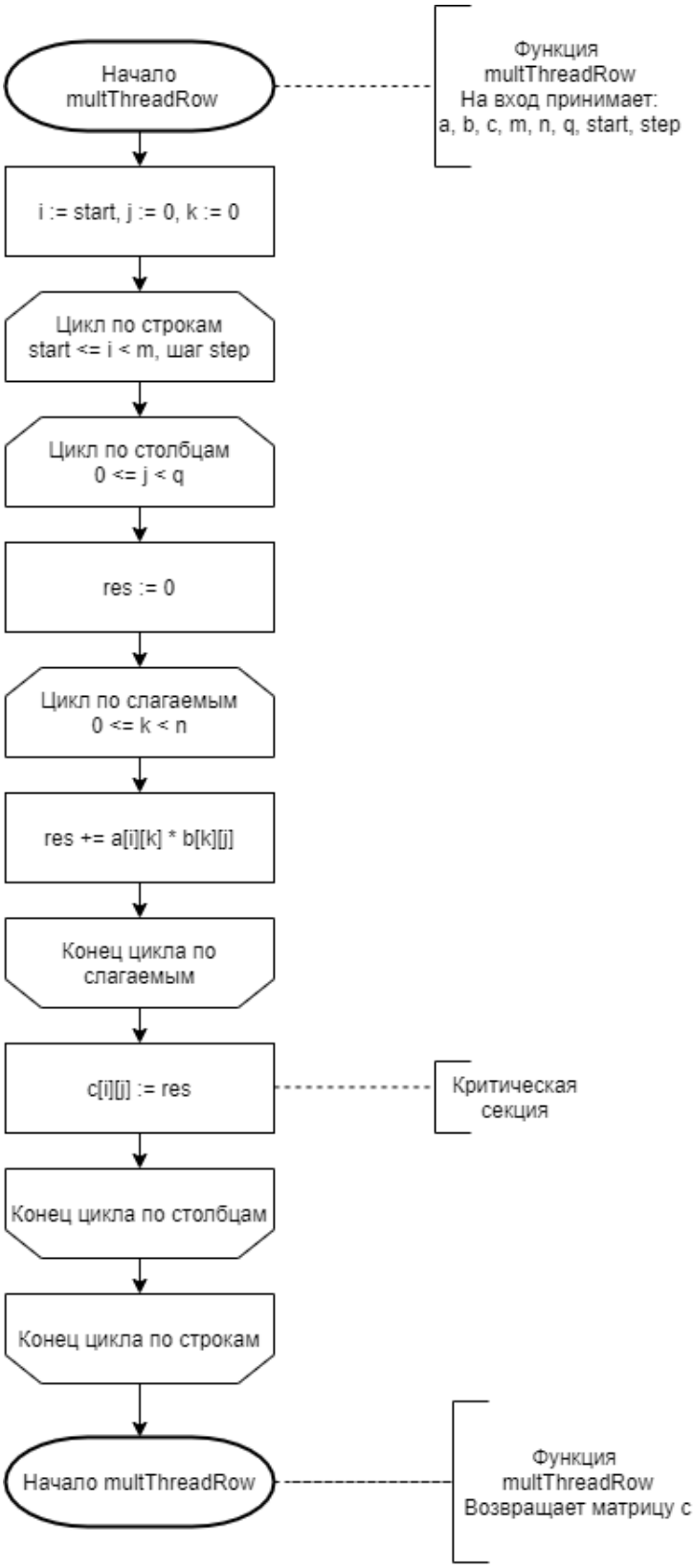

Рис. 2.4: Схема алгоритма параллельного умножения по столбцам, часть 1

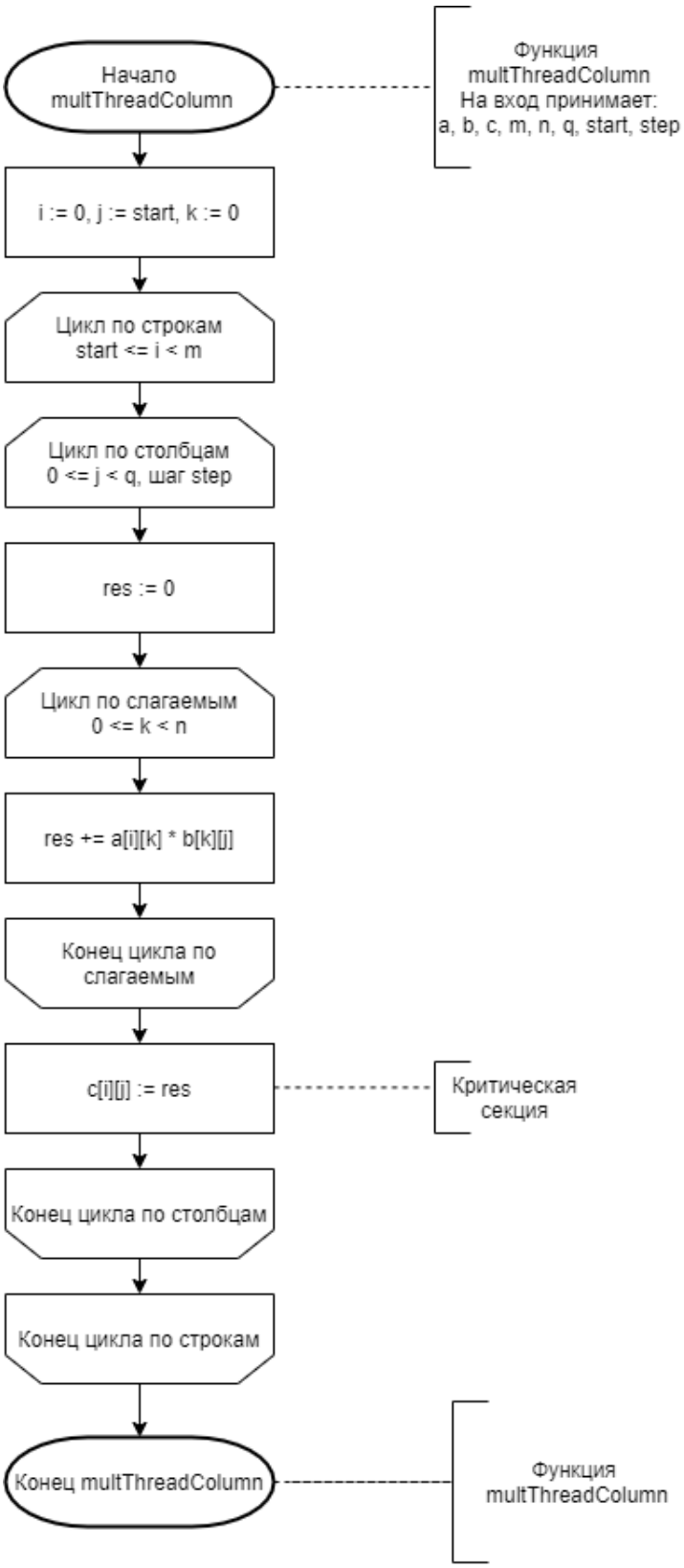

Рис. 2.5: Схема алгоритма параллельного умножения по столбцам, часть 1

2.4. Требования к программному обеспечению

Для полноценной проверки и оценки алгоритмов необходимо выполнить следующее:

- предоставить выбор алгоритма для умножения, обеспечить возможность консольного ввода двух матриц и количества используемых потоков (в случае выбора многопоточного алгоритма). Программа должна вывести результирующую матрицу.
- реализовать функцию замера процессорного времени, затраченного функциями.

2.5. Вывод

Результатом конструкторской части стало схематическое описание алгоритмов умножения матриц, сформулированы тесты и требования к программному обеспечению.

3. Технологическая часть

3.1. Выбор языка программирования

В качестве языка программирования был выбран C++ [1], так как имеется опыт работы с ним и с библиотеками, позволяющими провести исследование и тестирование программы. Также в языке имеются средства для использования многопоточности. Разработка проводилась в среде Visual Studio Code.

3.2. Листинг кода

В листингах 3.1 - 3.3 приведены реализации умножения матриц.

Листинг 3.1: функция классического умножения матриц

```
matrixType classicMultiplication(matrixType a,
                                 matrixType b,
                                 int m, int n, int q)
{
    if (m != q)
        return;

    matrixType c = createMatrix(m, q);

    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < q; j++)
        {
            int res = 0;

            for (int k = 0; k < n; k++)
                res += a[i][k] * b[k][j];

            *(c + i * q + j) = res;
        }
    }
```

```
22      return c;
23  }
```

Листинг 3.2: функции параллельного умножения матриц по строкам

```
1   void multThreadRow(matrixType& c, matrixType& a,
2                      matrixType&b, int m, int n,
3                      int q, int start, int step)
4   {
5       for (int i = start; i < m; i += step)
6       {
7           for (int j = 0; j < q; j++)
8           {
9               int res = 0;
10
11              for (int k = 0; k < n; k++)
12              {
13                  res += a[i][k] * b[k][j];
14              }
15              mtx.lock();
16              *(c + i * q + j) = res;
17              mtx.unlock();
18          }
19      }
20  }
21
22  matrixType multiplicationParallelRow(matrixType a,
23                                       matrixType b,
24                                       int m, int n,
25                                       int q,
26                                       int threadCnt)
27  {
28      if (m != q)
29          return;
30
31      matrixType c = createMatrix(m, q);
```

```
32      vector<thread> threadArray;
33
34      for (int i = 0; i < threadCnt; i++)
35      {
36          threadArray.push_back(thread(multThreadRow,
37                                       ref(c), ref(a),
38                                       ref(b), m, n, q,
39                                       i, threadCnt));
40      }
41
42      for (int i = 0; i < threadCnt; i++)
43      {
44          threadArray[i].join();
45      }
46
47      return c;
48  }
```

Листинг 3.3: функция параллельного умножения матриц по столбцам

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

```
17                *(c + i * q + j) = res;
18                mtx.unlock();
19            }
20        }
21    }
22
23    matrixType multiplicationParallelColumn(matrixType a,
24                                            matrixType b,
25                                            int m, int n,
26                                            int q,
27                                            int threadCnt)
28    {
29        if (m != q)
30            return;
31
32        matrixType c = createMatrix(m, q);
33        vector<thread> threadArray;
34
35        for (int i = 0; i < threadCnt; i++)
36        {
37            threadArray.push_back(thread(multThreadColumn,
38                                         ref(c), ref(a),
39                                         ref(b), m, n, q,
40                                         i, threadCnt));
41        }
42
43        for (int i = 0; i < threadCnt; i++)
44        {
45            threadArray[i].join();
46        }
47        return c;
48    }
```

3.3. Результаты тестирования

На рисунках 3.1 - 3.2 приведены скриншоты интерфейса программы и тестирования, которые проводились в ручную.

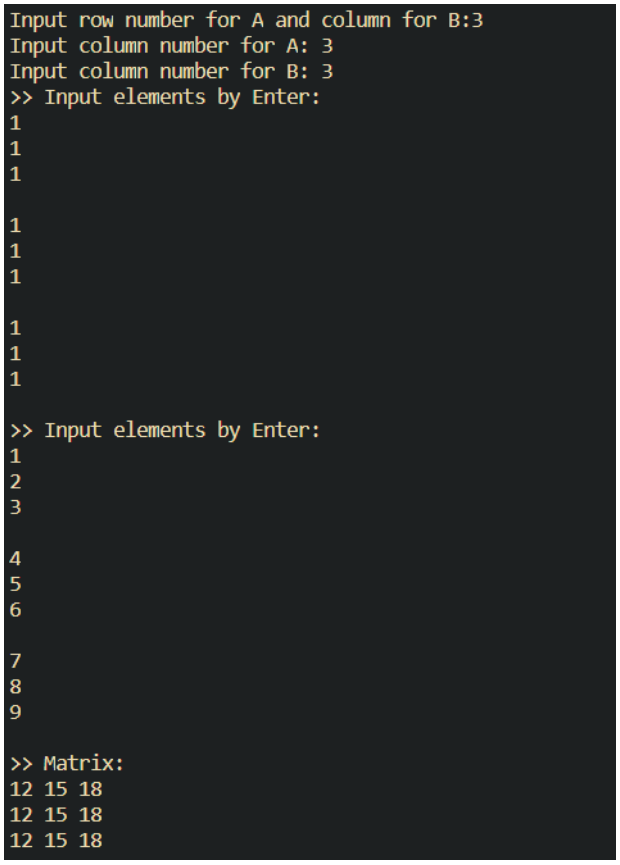

Рис. 3.1: Пример 1

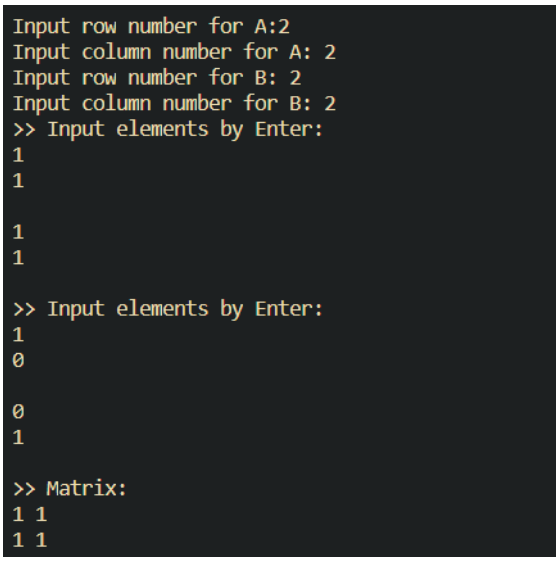

Рис. 3.2: Пример 2

Все тесты прошли успешно.

3.4. Оценка времени

Для замера процессорного времени исполнения функции используется функция QueryPerfomanceCounter библиотеки windows.h.

Измерение производится в функциях, которые приведены в листинге 3.4. Замеры времени на различных размерах матриц производятся в написанной функции в листинге 3.5.

Листинг 3.4: функции замера процессорного времени

```
1  double PCFreq = 0.0;
2  __int64 CounterStart = 0;
3
4  void StartCounter()
5  {
6      LARGE_INTEGER li;
7      if(!QueryPerformanceFrequency(&li))
8      std::cout << "QueryPerformanceFrequency failed!\n";
9
10     PCFreq = double(li.QuadPart);///1000.0;
11
12     QueryPerformanceCounter(&li);
13     CounterStart = li.QuadPart;
14 }
15
16 double GetCounter()
17 {
18     LARGE_INTEGER li;
19     QueryPerformanceCounter(&li);
20     return double(li.QuadPart-CounterStart)/PCFreq;
21 }
```

Листинг 3.5: функция замера времени на различных размерах матриц

```
1  void beginTimeTest()
2  {
3      int countSize = 5;
```

```
4       int sizes[] = { 32, 100, 250, 500, 1000 };
5
6       char algorithmNames[][100] = {"Classic multiplication",
7       "Parallel by row", "Parallel by column"};
8
9       for (int i = 0; i < countSize; i++)
10      {
11          matrixType a = generateMatrix(sizes[i], sizes[i]);
12          matrixType b = generateMatrix(sizes[i], sizes[i]);
13          double finishTime = 0;
14
15          for (int j = 0; j < 5; j++)
16          {
17              StartCounter();
18              classicMultiplication(a, b, sizes[i],
19                                    sizes[i], sizes[i]);
20              finishTime += GetCounter();
21          }
22
23          finishTime /= 5;
24
25          cout << "\nFor " << algorithmNames[0]
26                           << "\t-> Row - " << sizes[i]
27                           << "\tColumn - " << sizes[i]
28                           << "\tTime - " << finishTime;
29          deleteMatrix(a, sizes[i]);
30          deleteMatrix(b, sizes[i]);
31      }
32
33      matrixType (*algorithms[])(matrixType, matrixType,
34      int, int, int, int) = {multiplicationParallelRow,
35                             multiplicationParallelColumn};
36      int p = 1;
37
```

```
38      int nThreads[] = { 1, 2, 4, 8, 16, 32 };
39
40      for (int k = 0; k < 2; k++)
41      {
42          matrixType (*mult)(matrixType, matrixType,
43                  int, int, int, int) = algorithms[k];
44
45          for (int i = 0; i < countSize; i++)
46          {
47              for (int threads = 0; threads < 6; threads++)
48              {
49                  matrixType a = generateMatrix(sizes[i],
50                                                sizes[i]);
51                  matrixType b = generateMatrix(sizes[i],
52                                                sizes[i]);
53
54                  double finishTime = 0;
55
56                  for (int j = 0; j < 5; j++)
57                  {
58                      StartCounter();
59                      mult(a, b, sizes[i], sizes[i],
60                                           sizes[i],
61                                 nThreads[threads]);
62                      finishTime += GetCounter();
63                  }
64
65                  finishTime /= 5;
66
67                  cout << "\nFor " << algorithmNames[p]
68                              << "\t-> Row - " << sizes[i]
69                              << "\tColumn - " << sizes[i]
70                              << "\tTime - " << finishTime;
71
```

```
72                      p++;
73
74                      deleteMatrix(a, sizes[i]);
75                      deleteMatrix(b, sizes[i]);
76                  }
77              }
78          }
79  }
```

3.5. Вывод

Результатом технологической части стал выбор используемых технических средств реализации и последующая реализация алгоритмов, системы тестов и замера времени работы на языке C++.

4. Исследовательская часть

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.

4.1. Результаты экспериментов

Эксперименты проводились на компьютере со следующими характеристиками:

- ОС - Windows 10, 64bit;
- Процессор - Intel Core i5 7300HQ 2.5GHz, 4 Core 8 Logical Processor
- ОЗУ - 8Gb

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					

4.2. 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 ходе лабораторной работы достигнута поставленная цель: разработка и исследование параллельных алгоритмов умножения матриц. Решены все задачи.

Были изучены и описаны понятия параллелизма и операции умножения матриц. Также были описан и реализованы непараллельный и две параллельные реализации алгоритма умножения матриц. Проведены замеры процессорного времени работы алгоритмов при различном количестве потоков. На основании экспериментов проведён сравнительный анализ.

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.

Список литературы

1. Документация языка C++ 98 [Электронный ресурс], режим доступа: http://www.open-std.org/JTC1/SC22/WG21/, свободный (дата обращения: 14.10.20202).

2. Белоусов И. В. МАТРИЦЫ И ОПРЕДЕЛИТЕЛИ: учебное пособие по линейной алгебре. / Кишинев: 2006/.