**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 | | В.А. Иванов |
| | (Группа) | (Подпись, дата) | (И.О. Фамилия) |
| Преподаватель | | | |
| | | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

# Оглавление

## Введение

В данной лабораторной реализуется и оценивается параллельный алгоритм стандартного умножения матриц.

Параллелизм — выполнение нескольких вычислений в различных потоках. Параллельное программирование интересно тем, что 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

## 1.1. Цели и задачи работы

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

Выделены 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.

## 1.2. Математическое описание операции умножения матриц

Умножение матриц - операция над матрицами A$[M * N]$ и B$[N * Q]$ [1]. Результатом операции является матрица C размерами $M * Q$, в которой каждый элемент $c_{i,j}$ задаётся формулой

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

## 1.3. Используемые алгоритмы

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

Параллелизм может быть досигнут за счёт выделения процес-

сов, 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 определение цели и задач работы, описано понятие опереции умножения матриц и описаны используемые алгоритмы.

# 2. Конструкторская часть

Рассмотрим описаные алгоритмы умножения матриц умножения матриц $[M * N]$ и $B[N * Q]$ с результирующей матрицей умножения матриц $C[M * Q]$.

## 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

## 2.2. Алгоритм умножения параллельный по строкам

Как было отмечено выше, вычисление каждого элемента матрицы является независимым. Поэтому возможна следующая параллельная версия данного алгоритма. Пусть производится работа с $T$ потоками. В таком случае, i-й поток будет производить вычисление строк $i$, $i + T$, $i + 2T$, ..., $((M - i)modT) * T + i$.

Схема алгоритма приведена на рисунках 2.2, 2.3

## 2.3. 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 $i$, $i + T$, $i + 2T$, ..., $((Q - i)modT) * T + i$.

Схема алгоритма приведена на рисунках 2.4, 2.5

## 2.4. Требования к программному обеспечению

Для полноценной проверки и оценки алгоритмов необходимо выполнить следующее.

1. 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.
2. Реализовать функцию замера процессорного времени, затраченного функциями.

## 2.5. Заготовки тестов

При проверке алгоритма необходимо будет использовать следующие классы тестов:

- один поток, несколько потоков;
- матрицы размером 1x1;
- пустые матрицы.

## Вывод

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

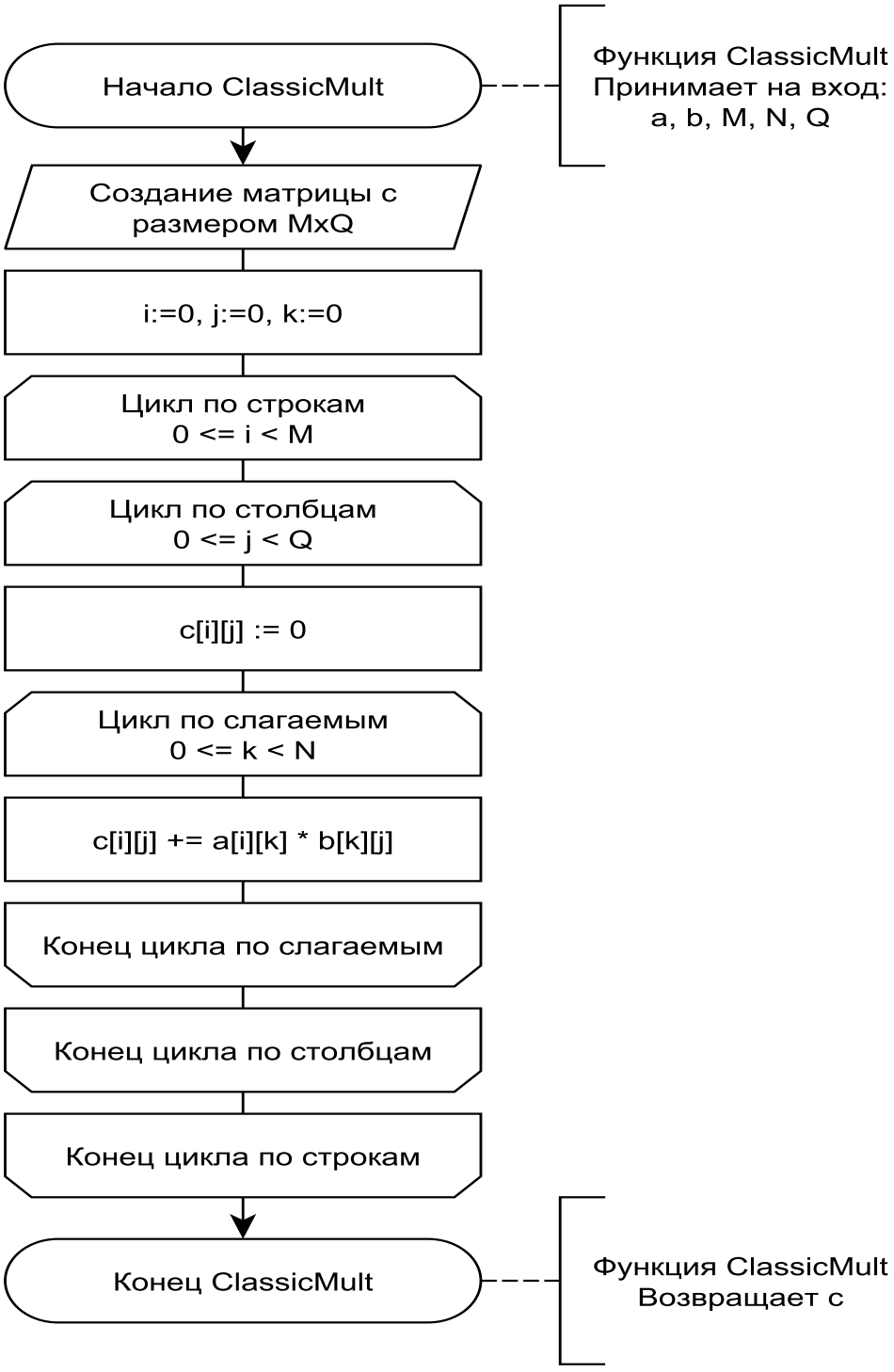


Рис. 2.1 — Стандартный алгоритм умножения

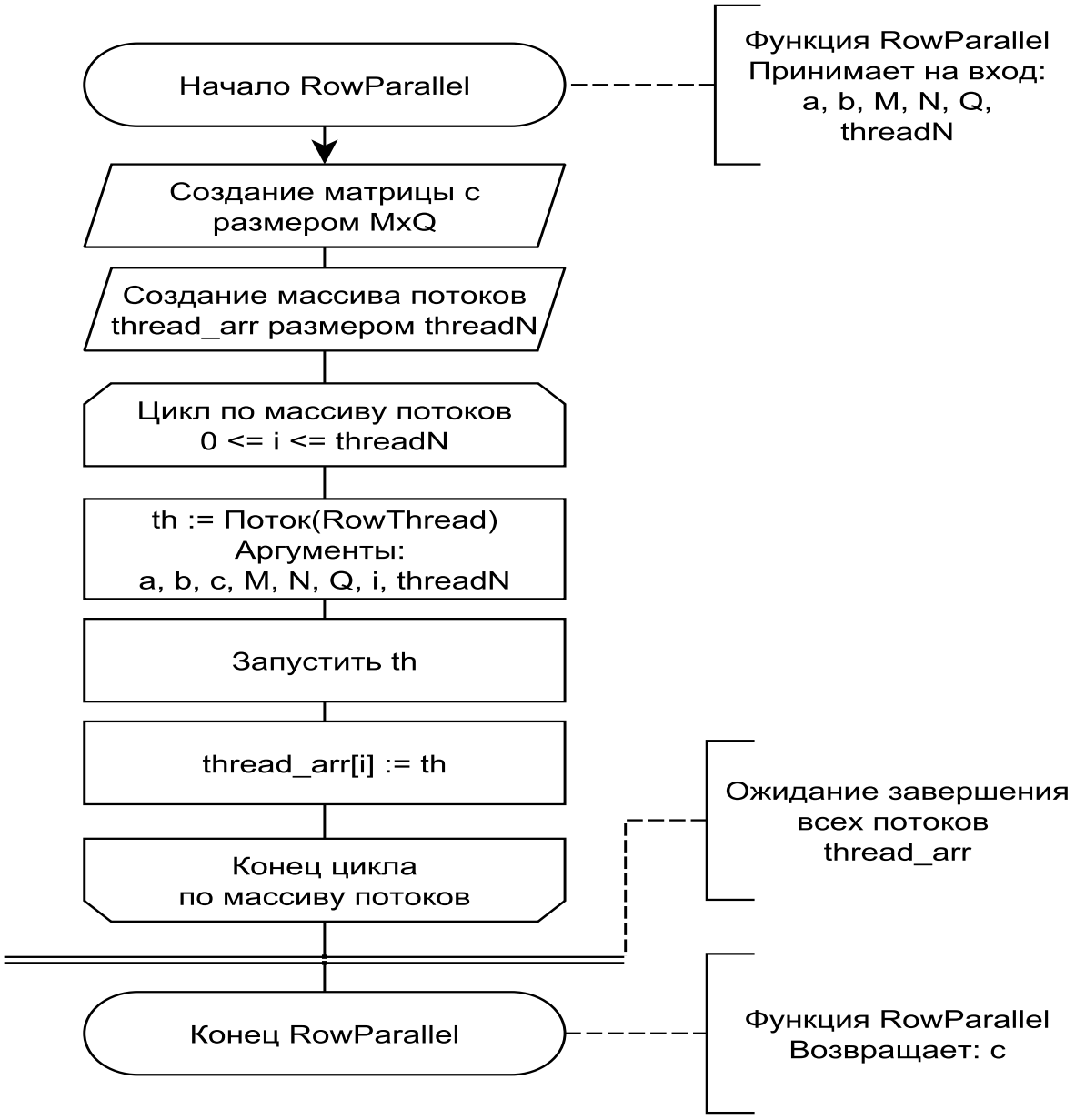


Рис. 2.2 — Алгоритм умножения параллельный по строкам (главный поток)

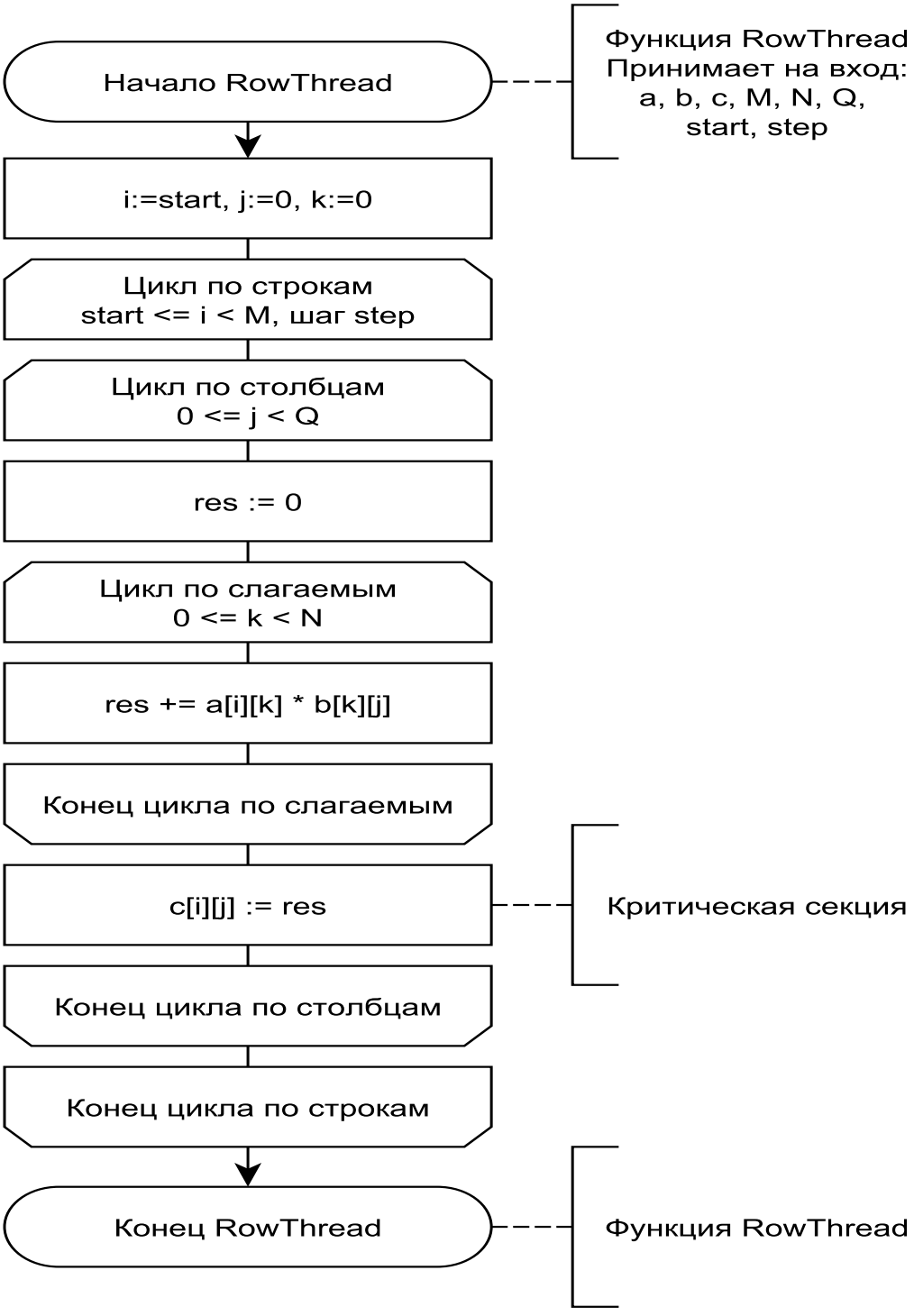


Рис. 2.3 — Алгоритм умножения параллельный по строкам (рабочий поток)

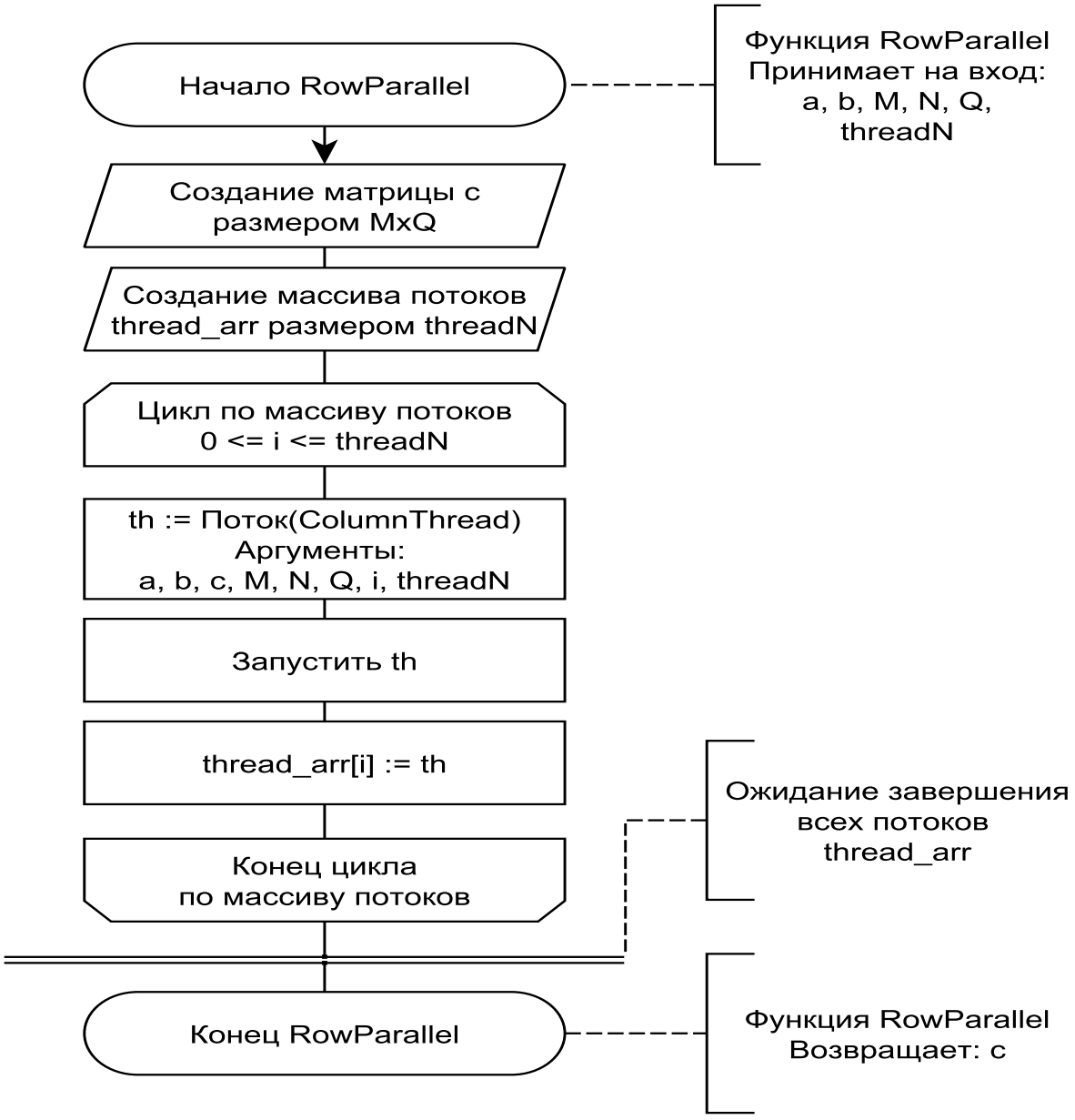


Рис. 2.4 — Алгоритм умножения параллельный по столбцам (главный поток)

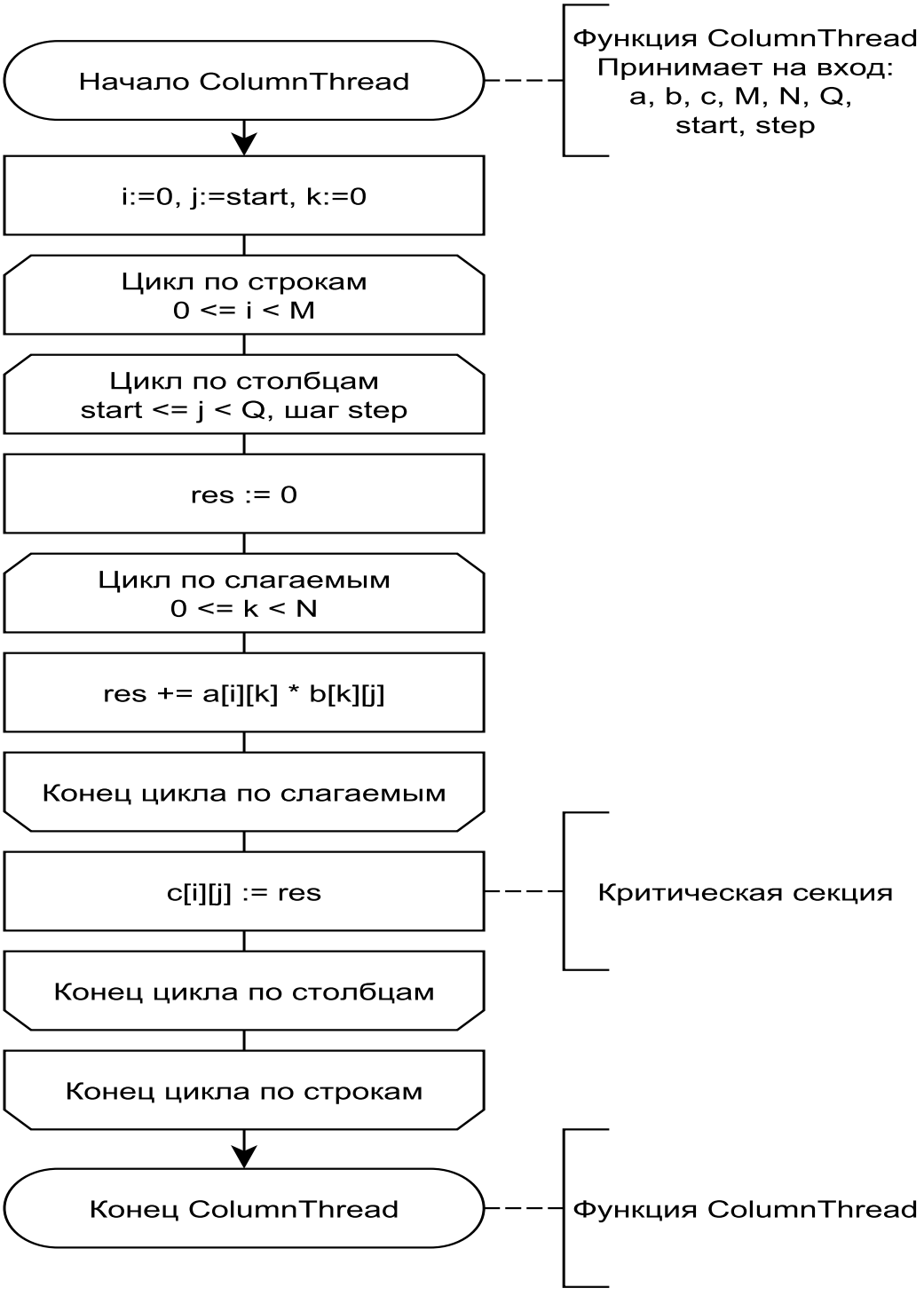


Рис. 2.5 — Алгоритм умножения параллельный по столбцам (рабочий поток)

# 3. Технологическая часть

## 3.1. Выбор языка программирования

В качестве языка программирования был выбран C++[2], так как имеется опыт работы с ним и с библиотеками, позволяющими провести исследование и тестирование программы. Также в языке имеются средства для использования многопоточности[3]. Разработка проводилась в среде Visual Studio 2019[4].

## 3.2. Листинг кода

Реализация алгоритмов умножения матриц представлена на листингах 3.1-3.3

Листинг 3.1 — Функция однопоточного умножения матриц

```
mat_t std_mult(mat_t a, mat_t b, int m, int n, int q, int)
{
  mat_t c = create_mat(m, q);
  for (int i = 0; i < m; i++)
    for (int j = 0; j < q; j++)
    {
      double res = 0;
      for (int k = 0; k < n; k++)
        res += a[i][k] * b[k][j];
      c[i][j] = res;
    }
  return c;
}
```

Листинг 3.2 — Функция умножения матриц (параллельная по строкам)

```
using namespace std;
std::mutex mtx;

```

```
4  void _mult_thread(mat_t& c, mat_t& a, mat_t& b, int m, int n,
       int q, int start, int step)
5  {
6    for (int i = start; i < m; i+= step)
7      for (int j = 0; j < q; j++)
8      {
9        double res = 0;
10       for (int k = 0; k < n; k++)
11         res += a[i][k] * b[k][j];
12       mtx.lock();
13       c[i][j] = res;
14       mtx.unlock();
15     }
16 }
17
18 mat_t std_mult_thread1(mat_t a, mat_t b, int m, int n, int q,
       int thread_n)
19 {
20   mat_t c = create_mat(m, q);
21
22   vector<thread> thread_arr;
23   for (int i = 0; i < thread_n; i++)
24     thread_arr.push_back(thread(_mult_thread, ref(c), ref(a),
       ref(b), m, n, q, i, thread_n));
25
26   for (int i = 0; i < thread_n; i++)
27     thread_arr[i].join();
28
29   return c;
30 }
```

Листинг 3.3 — Функция умножения матриц (параллельная по столбцам)

```
1  using namespace std;
2  std::mutex mtx;
3
4  void _mult_thread2(mat_t& c, mat_t& a, mat_t& b, int m, int n
       , int q, int start, int step)
5  {
6    for (int i = 0; i < m; i++)
7      for (int j = start; j < q; j+=step)
8      {
9        double res = 0;
10       for (int k = 0; k < n; k++)
11         res += a[i][k] * b[k][j];
12       mtx.lock();
13       c[i][j] = res;
14       mtx.unlock();
15     }
16 }
17
18 mat_t std_mult_thread2(mat_t a, mat_t b, int m, int n, int q,
       int thread_n)
19 {
20   mat_t c = create_mat(m, q);
21
22   vector<thread> thread_arr;
23   for (int i = 0; i < thread_n; i++)
24     thread_arr.push_back(thread(_mult_thread2, ref(c), ref(a)
       , ref(b), m, n, q, i, thread_n));
25
26   for (int i = 0; i < thread_n; i++)
27     thread_arr[i].join();
28
29   return c;
30 }
```

## 3.3. Результаты тестирования

Для тестирования написанных функций был создан отдельный файл с ранее описанными классами тестов. Тестирование функций проводилось за счёт сравнения результов функций друг с другом.

Состав тестов приведён в листинге 3.4.

Листинг 3.4 — Модульные тесты

```
#include "tests.h"

using namespace std;

// Сравнение работы функций
bool _cmp_funcs(mat_t a, mat_t b, int m, int n, int q, int
    thread_n)
{
  mat_t c0 = std_mult(a, b, m, n, q, thread_n);
  mat_t c1 = std_mult_thread1(a, b, m, n, q, thread_n);
  mat_t c2 = std_mult_thread2(a, b, m, n, q, thread_n);

  bool flag = cmp_matrix(c0, c1, m, q);
  if (flag)
    flag = cmp_matrix(c1, c2, m, q);

  free_mat(&c0, m, q);
  free_mat(&c1, m, q);
  free_mat(&c2, m, q);
  return flag;
}

// Размер матриц = 1
void _test_size_one()
```

```
24 {
25   mat_t a = create_mat(1, 1);
26   mat_t b = create_mat(1, 1);
27 
28   a[0][0] = 0;
29   b[0][0] = 1;
30   if (!_cmp_funcs(a, b, 1, 1, 1, 1))
31   {
32     std::cout << __FUNCTION__ << " - FAILED\n";
33     return;
34   }
35 
36   a[0][0] = 3;
37   b[0][0] = 4;
38   if (!_cmp_funcs(a, b, 1, 1, 1, 16))
39   {
40     std::cout << __FUNCTION__ << " - FAILED\n";
41     return;
42   }
43 
44   free_mat(&a, 1, 1);
45   free_mat(&b, 1, 1);
46 
47   std::cout << __FUNCTION__ << " - OK\n";
48 }
49 
50 // Пустые матрицы
51 void _test_void()
52 {
53   mat_t a = random_matrix(3, 2);
54   mat_t b = void_matrix(2, 1);
55   if (!_cmp_funcs(a, b, 3, 2, 1, 1))
56   {
57     std::cout << __FUNCTION__ << " - FAILED\n";
```

```
58      return;
59    }
60    free_mat(&a, 3, 2);
61
62    a = void_matrix(3, 2);
63    if (!_cmp_funcs(a, b, 3, 2, 1, 1))
64    {
65      std::cout << __FUNCTION__ << " - FAILED\n";
66      return;
67    }
68    free_mat(&a, 3, 2);
69    free_mat(&b, 2, 1);
70    std::cout << __FUNCTION__ << " - OK\n";
71  }
72
73  // Сравнение работы при различном количестве потоков
74  void _test_threads()
75  {
76    mat_t a = random_matrix(50, 50);
77    mat_t b = random_matrix(50, 50);
78
79    for (int i = 1; i <= 16; i++)
80    {
81      if (!_cmp_funcs(a, b, 50, 50, 50, i))
82      {
83        std::cout << __FUNCTION__ << " - FAILED\n";
84        break;
85      }
86    }
87
88    free_mat(&a, 50, 50);
89    free_mat(&b, 50, 50);
90
91    std::cout << __FUNCTION__ << " - OK\n";
```

```
92 }
93
94 void run_tests()
95 {
96   _test_size_one();
97   _test_void();
98   _test_threads();
99 }
```

Все тесты пройдены успешно.

## 3.4. Оценка времени

Для замера процессорного времени исполнения функции используется функция QueryPerformanceCounter библиотеки windows.h[5]. Проведение измерений производится в функциях, приведённых в листинге 3.3.

Листинг 3.5 — Функции замера процессорного времени работы функции

```
1  void test_time(mult_f f, int thread_n)
2  {
3    int n = 1000;
4    mat_t a = random_matrix(n, n);
5    mat_t b = random_matrix(n, n);
6    mat_t c;
7
8    int count = 0;
9    start_counter();
10   while (get_counter() < 3.0 * 1000)
11   {
12     c = f(a, b, n, n, n, thread_n);
13     free_mat(&c, n, n);
14     count++;
```

```
15    }
16    double t = get_counter() / 1000;
17
18    start_counter();
19    for (int i = 0; i < count; i++)
20    {
21      c = create_mat(n, n);
22      free_mat(&c, n, n);
23    }
24    t -= get_counter() / 1000;
25
26    cout << "Выполнено " << count << " операций за " << t << "
      секунд" << endl;
27    cout << "Время: " << t / count << endl;
28
29    free_mat(&a, n, n);
30    free_mat(&b, n, n);
31  }
32
33  void experiments_series(vector<int>& a)
34  {
35    for (int i : a)
36    {
37      cout << "==========================================" << endl
      ;
38      cout << "\Количествоп потоков: " << i << endl;
39
40      cout << "Стандартное умножениемногопоточно( по строкам)" <<
      endl;
41      test_time(std_mult_thread1, i);
42      cout << "Стандартное умножениемногопоточно( по столбцам)" <<
      endl;
43      test_time(std_mult_thread2, i);
44
```

```
45      cout << "\Стандартноеn умножениемногопоточно( по строкам)" <<
        endl;
46      test_time(std_mult_thread1, i);
47      cout << "Стандартное умножениемногопоточно( по столбцам)" <<
        endl;
48      test_time(std_mult_thread2, i);
49
50      cout << "==========================================" << endl
        ;
51    }
52    cout << "Стандартное умножение однопоточно()" << endl;
53    test_time(std_mult, 1);
54  }
```

## Вывод

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

# 4. Исследовательская часть

## 4.1. План экспериментов

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: $1, 2, 4, 8, 16, 32$

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

## 4.2. Результат экспериментов

По результатам измерений процессорного времени можно составить таблицы 4.1, 4.2, 4.3, 4.4, 4.5

Таблица 4.1 — Результат измерений процессорного времени, размер 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 — Результат измерений процессорного времени, размер 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 — Результат измерений процессорного времени, размер 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 — Результат измерений процессорного времени, размер 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 — Результат измерений процессорного времени, размер 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 | | | | | |

Эксперименты проводились на компьютере с характеристиками:

- ОС - Windows 10, 64 бит;
- Процессор - Intel Core i7 8550U (1800 МГц, 4 ядра, 8 логи-

ческих процессоров);

- Объем ОЗУ: 8 ГБ.

## Вывод

По результатам экспериментов можно заключить следующее.

- При относительно небольшом размере матриц (менее 100x100) использование потоков для уменьшения времени исполнения нецелесообразно, так как накладнымые расходы времени на управление потоками и mutex-ами больше, чем выигрыш от параллельного выполнения вычислений.
- Наиболее быстродейственно алгоритм действует на 8 потоках, что равно количеству логических процессоров на испытуемом компьютере.
- 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.
- 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%.

- 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 количества логических ядер машины.

## Заключение

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

Были изучены и описаны понятия параллелизма и операции умножения матриц. Также были описан и реализованы непараллельный и две параллельные версии параллельного 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 быстродейственным является использование количества потоков, которое равно количеству логических ядер компьютера. 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. Белоусов И. В. МАТРИЦЫ И ОПРЕДЕЛИТЕЛИ: учебное пособие по линейной алгебре. / Кишинев: 2006/.

2. Документация языка C++ 98 [Электронный ресурс]. Режим доступа: http://www.open-std.org/JTC1/SC22/WG21/, свободный (дата обращения: 14.10.2020)

3. Документация библиотеки std (раздел о потоках) [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/thread, свободный (дата обращения: 23.10.2020).

4. Документация среды разработки Visual Studio 2019 [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2019, свободный (дата обращения: 14.10.2020)

5. QueryPerformanceCounter function [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter, свободный (дата обращения: 29.09.2020).