Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Лабораторная работа №4

по курсу «Анализ алгоритмов»

Многопоточность

Выполнил: Тимонин А.С., гр. ИУ7-52Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

2019 г.

Оглавление

Введение

Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность. Смысл многопоточности - квазимногозадачность на уровне одного исполняемого процесса[2].

В данной работе требуется рассмотреть алгоритм Винограда для умножения матриц в однопоточной и многопоточной реализациях, а также провести сравнительный анализ.

Цель работы: изучение многопоточности и получение практики на примере алгоритма Винограда для перемножения матриц.

Задачи работы:

1. Разработка и реализация алгоритмов.
2. Исследование временных затрат алгоритма.
3. Описание и обоснование полученных результатов.

1. Аналитический раздел

В данном разделе будет описан алгоритм Винограда для перемножения матриц.

1.1 Алгоритм Винограда

Алгоритм Винограда это модифицированная версия классического алгоритма, где часть процессов высчитывается заранее. Эти вычисления позвовлят разбить вычисления алгоритма на потоки. Пусть есть две матрицы A и B размеров nxk, kxm соответственно. Тогда результатом перемножения этих двух матриц будет матрица C размера nxm 1.1.

$$A_{nk} * B_{km} = C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & c_{22} & \cdots & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nm} \end{pmatrix} \tag{1.1}$$

Если посмотреть на результат умножения двух матриц, то можно заметить, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Также, такое умножение позволяет сделать предварительную обработку заранее [3]

Пусть два вектора

$$V = (v1, v2, v3, v4),$$

$$W = (w1, w2, w3, w4)$$

Тогда их скалярное произведение равно:

$$V * W = v1w1 + v2w2 + v3w3 + v4w4$$

Это равенство можно переписать в виде:

$$V * W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4$$

1.2 Многопоточность

Существуют зеленые и нативные потоки. Зеленые потоки - это потоки выполнения, управление которыми вместо операционной системы выполняет виртуальная машина. Программа написанная на языке, поддерживающим зеленые потоки, только эмитирует многопоточность.

На многоядерных процессорах реализация нативных потоков может автоматически назначать работу нексольким процессорам, а реализация зеленых потоков не может назначить работу нескольким процессорам.

Поток выполнения - наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов[1]

1.3 Вывод

В данном разделе был описан алгоритм Винограда перемножени матриц.

2. Конструкторский раздел

В данном разделе будет приведена блок-схема алгоритма Винограда для перемножения матриц, описано для каких частей алгоритма выделялись потоки, и каким образом это было реализовано.

2.1 Разработка алгоритмов

В данном пункте представлена реализация алгоритма Винограда на рис. 2.1.

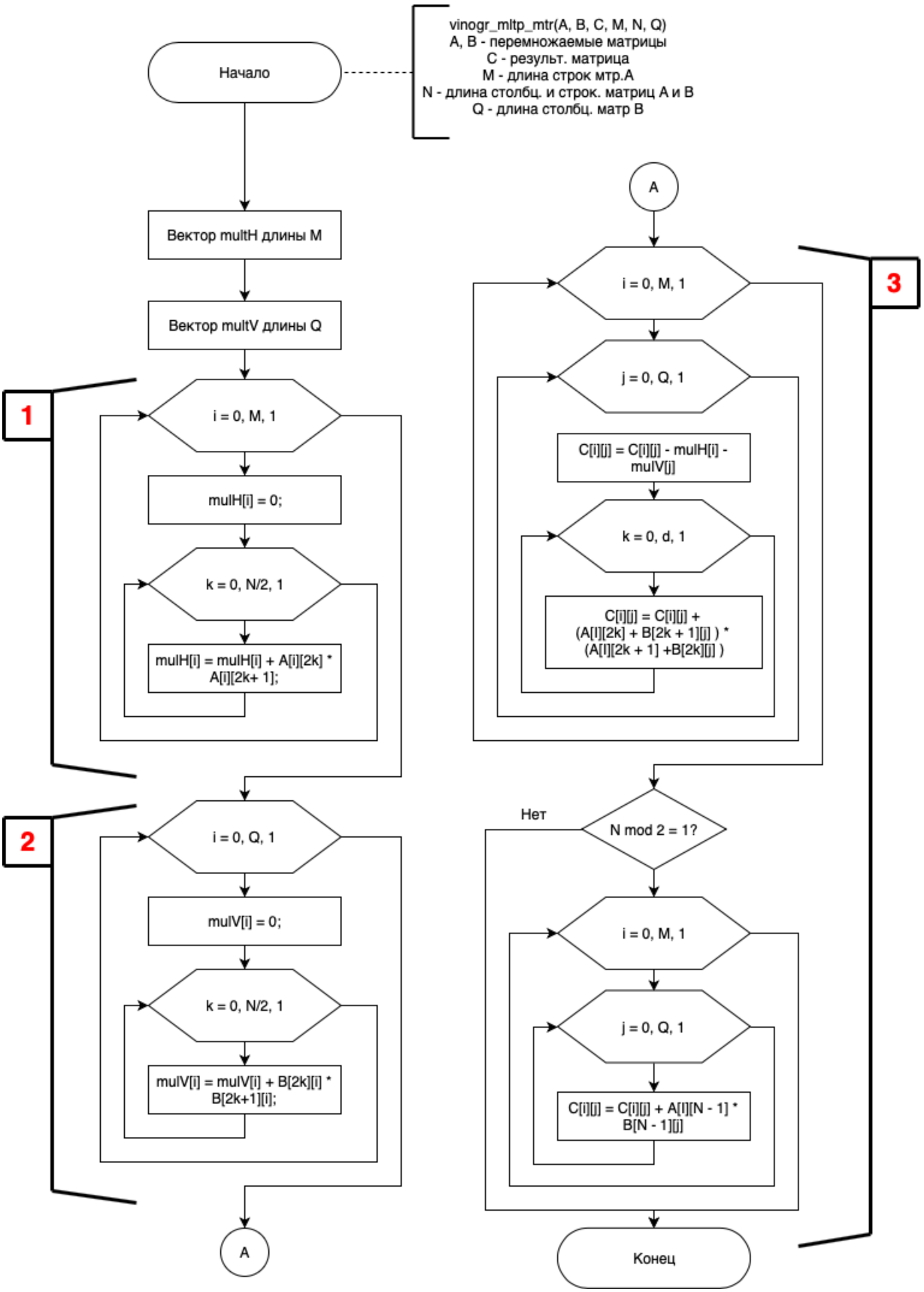

Рис. 2.1: Схема алгоритма Винограда

Для реализации многопоточной версии алгоритма можно выделить 4 основные части

1. Создание и инициализация MulH.
2. Создание и инициализация MulV.
3. Основные и дополнительные(для входных матриц нечетной размерностей) вычисления матрицы произведения.

Части 1, 2 разбиваются по потокам. Поток для 3 части не выделяется, пока не выполнятся 1 и 2 части.

2.2 Вывод

В данном разделе была приведена схема алгоритма, было описано каким образом выделялись потоки в реализованном алгоритме Винограда.

3. Технологический раздел

В данном разделе будут рассмотренны требования к разрабатываемому программному обеспечиванию, средства, использованные в процессе разработки для реализации поставленных задач, а также представлены листинги кода программы.

3.1 Требования к программному обеспечению

Программное обеспечивание должно реализовывать алгоритм Винограда для перемножения матриц в однопоточной и многопоточных реализациях.

3.2 Средства реализации

Для выполнения поставленной задачи был использован язык программирования C++. Среда для разработки XCode. Для измерения процессорного времени была взята функция duration из библиотеки chrono.

Версия компилятора C++: GNU++14 [-std=gnu++14]

3.3 Листинг кода

На основе схема, приведенной в конструкторском разделе, в соответствии с указанными требованиями к реализации с использованием языка C++ было разработано программное обеспечение, содержащее реализации выбранных алгоритмов. В данном пункте приведен листинги 3.1-3.2 реализации алгоритма.

Листинг 3.1: Реализация алгоритма Винограда

```
1  void Vinograd_n_thread(matrix_type &a, matrix_type &b, matrix_type &c, int n)
2  {
3  vector<int> row(a.n);
4  vector<int> column(b.m);
5
6  vector<thread> threads;
7
8  unsigned int n1 = a.n / 2;
9  zeroing(c.matrix, c.n, c.m);
10
11 double length_part = (double) a.n / n;
12 if (length_part < 1) {
13 length_part = 1;
14 }
15
16 for (int i = 0; i < a.n; i++) {
17 row.push_back(0);
```

```
18 }
19 for (int i = 0; i < b.m; i++) {
20 column.push_back(0);
21 }
22
23 thread thr_mulH(create_mulH, ref(a.matrix), ref(row), 0, ref(a.n), ref(a.m));
24 thread thr_mulV(create_mulV, ref(b.matrix), ref(column), 0, ref(b.m), ref(b.n));
25
26 thr_mulH.join();
27 thr_mulV.join();
28
29 for (int i = 1; i <= n; i++) {
30 threads.push_back(thread(calculate1, ref(a), ref(b), ref(c), ref(row), ref(column), (a.n * (i - 1)) / n, (
       a.n * i) / n));
31 }
32
33 for (int i = 0; i < threads.size(); ++i) {
34 if (threads[i].joinable()) {
35 threads[i].join();
36 }
37 }
38 }
```

Листинг 3.2: Основные вычисления для многопоточной реализации алгоритма Винограда

```
1 void create_mulH(int **&A, vector <int>& row, const unsigned int &M_start, const
       unsigned int &M_end, const unsigned int &N)
2 {
3
4 for (unsigned i = M_start; i < M_end; i++) {
5 //cout << this_thread::get_id()<< endl;
6 for (unsigned k = 0; k < N / 2; k++) {
7 row[i] += A[i][2 * k] * A[i][2 * k + 1];
8 }
9 }
10 }
11
12 void create_mulV(int **&B, vector <int>& column, const unsigned int &Q_start, const
       unsigned int &Q_end, const unsigned int &N)
13 {
14
15 for (unsigned i = Q_start; i < Q_end; i++) {
16 //cout << this_thread::get_id()<< endl;
17 for (unsigned k = 0; k < N / 2; k++) {
18 column[i] += B[2 * k][i] * B[2 * k + 1][i];
19 }
20 }
21 }
22
23 void calculate(int **&A, int **&B, int **&C, vector <int> &row, vector <int> &column,
       const unsigned int &M, const unsigned int &N, const unsigned int &Q)
24 {
25 for (unsigned i = 0; i < M; i++)
26 for (unsigned j = 0; j < Q; j++) {
```

```
27 if (N % 2 == 0)
28 C[i][j] = -row[i] - column[j];
29 else
30 C[i][j] = -row[i] - column[j] + A[i][N - 1] * B[N - 1][j];
31
32 for (unsigned k = 0; k < N / 2; k++) {
33 C[i][j] = C[i][j] + (A[i][k << 1] + B[k << 1 | 1][j]) *
34 (A[i][k << 1 | 1] + B[k << 1][j]);
35 }
36 }
37
38 }
39
40 void calculate1(matrix_type &a, matrix_type &b, matrix_type &c, vector <int> &row, vector <int
       > &column, const unsigned int n_start, unsigned int n_end)
41 {
42 int sum = 0;
43
44 for (unsigned i = n_start; i < n_end; i++) {
45 //cout << this_thread::get_id()<< endl;
46 for (unsigned j = 0; j < b.m; j++) {
47
48 sum = -row[i] - column[j];
49
50 for (unsigned k = 0; k < a.m / 2; k++) {
51 sum += (a.matrix[i][2*k] + b.matrix[2*k+1][j]) *
52 (a.matrix[i][2*k+1] + b.matrix[2*k][j]);
53 }
54
55 if (a.m % 2 == 1)
56 sum += a.matrix[i][a.m - 1] * b.matrix[b.n - 1][j];
57
58 c.matrix[i][j] = sum;
59 }
60 }
61
62 }
```

3.4 Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки, а также был представлены листинги кода реализации алгоритма Винограда.

4. Экспериментальный раздел

В данном разделе будет приведено экспериментальное исследование временных затрат разработанного программного обеспечения, вместе в подробным сравнительным анализом реализованных алгоритмов на основе экспериментальных данных.

4.1 Сравнительный анализ

Замеры времени выполнялись на квадратных матрицах размеров от 100x100 до 1000x1000 с интервалом в 100 элементов. Также замеры времени проводились над квадратными матрицами нечетной размерностью от 101x101 до 1001x1001 с шагом 100. В табл. 4.1-4.2 и рис. 4.1-4.1 представлены результаты замеров времени в тактах.

Все замеры проводились на процессоре 1,4 GHz Intel Core i5 с памятью 8 ГБ 2133 MHz LPDDR3.

Таблица 4.1: Сравнение времени работы однопоточной и многопоточной версий алгоритма в единицах измерения библиотеки chrono

Размерность матриц	**Поток 1**	**Поток 2**	**Поток 3**	**Поток 4**	**Поток 5**	**Поток 6**
100	4	2	1	1	1	1
200	36	17	7	6	6	6
300	128	56	36	35	27	27
400	334	165	111	89	90	77
500	590	331	205	176	161	148
600	932	535	369	279	250	241
700	1504	857	591	505	465	423
800	2329	1261	815	710	731	675
900	3808	2818	1570	1166	1131	1084
1000	10691	4392	2743	2169	1816	1529

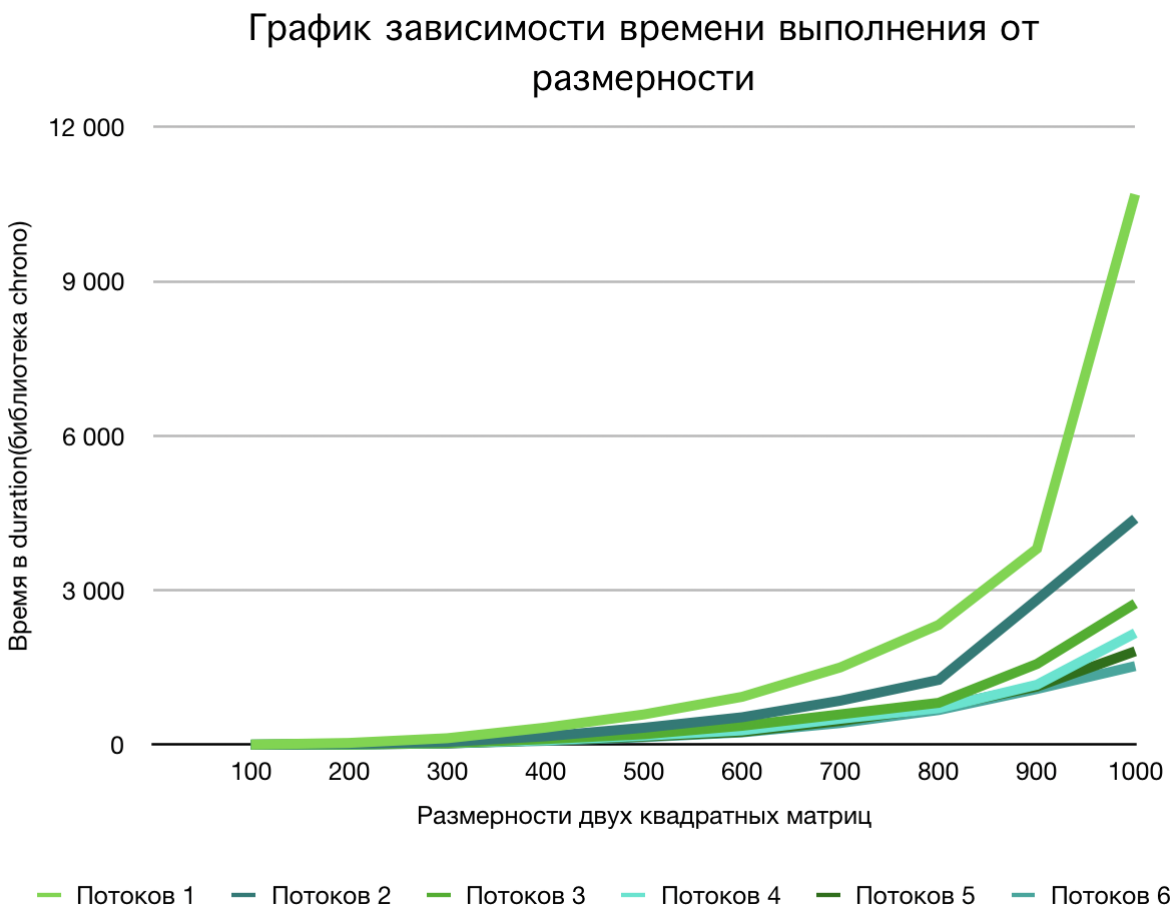

Рис. 4.1: График зависимости времени работы однопоточной и многопоточных версий алгоритма для четной размерности матриц

Таблица 4.2: Сравнение времени работы однопоточной и многопоточной версий алгоритма в единицах измерения библиотеки chrono

Размерность матриц	**Поток 1**	**Поток 2**	**Поток 3**	**Поток 4**	**Поток 5**	**Поток 6**
101	4	2	1	1	1	1
201	24	15	7	11	7	7
301	116	62	37	26	31	30
401	320	153	111	86	84	75
501	531	306	209	163	150	142
601	993	492	333	260	249	242
701	1487	853	556	455	556	428
801	2482	1154	850	699	729	643
901	6480	2810	1562	1093	1175	1362
1001	8949	3892	2645	2179	2260	2054

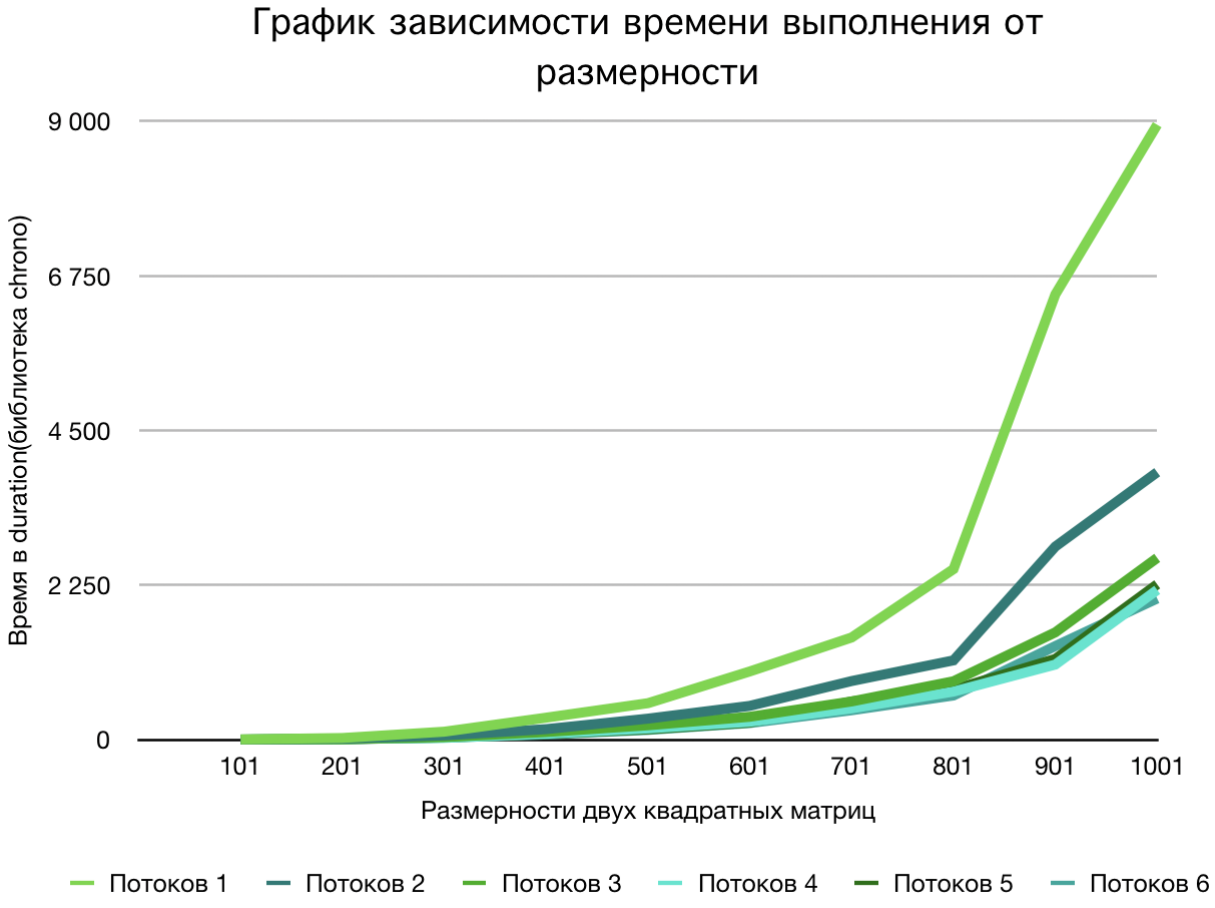

Рис. 4.2: График зависимости времени работы однопоточной и многопоточных версий алгоритма для нечетной размерности матриц

Из данных графиков видно, что присутствие хотя бы двух потоков в несколько раз эффективнее, в сравнению с однопоточной реализацией алгоритма. Разница по времени выполнения в многопоточной реализации алгоритма не зависит от четной или нечетной

Таблица 4.3: Время работы программы для 7 и 8 потоков

Размерность матриц	**Поток 7**	**Поток 8**
100	1	1
200	7	6
300	26	26
400	70	66
500	132	126
600	218	210
700	410	370
800	595	581
900	1133	937
1000	1356	1482

размерности матриц. Самое опитмальное количество потоков для реализации данного алгоритма - 4, из-за того что на восьми потоках разниц не существенна.

4.2 Вывод

В данном разделе было проведено исследование однопоточной и многопоточных версий алгоритма. Приведены графики зависимостей времени работы алгоритма от размерности матриц.

Среди всех версий алгоритма самыми лучшим оказались версии, где задействовалось не менее четырех потоков. Многопоточная версия алгоритмов показала хороший реазультат относительно однопоточной версии: восьмипоточная реализация алгоритма эффективней однопоточной на 400% при размерности матриц равной 1000.

5. Заключение

В ходе выполнения данной лабораторной работы были изучены и реализованы различные алгоритмы перемножения матриц. В аналитической части было приведено описание алгоритмов. В конструкторской части были представлены блок-схемы алгоритмов. Также был выполнен расчет сложности алгоритмов. В экспериментальной части проведен сравнительный анализ временных затрат, после которого была выявлена ощутимая эффективность многопоточного программирования по отношению к стандартному, однопоточному.

На матрицы размерности 1000x1000 результаты оказались следующими:

- Эффективность 1 потока к 2 потокам составила 240%.
- Эффективность 1 потока к 4 потокам составила 490%.
- Эффективность 1 потока к 6 потокам составила 700%.
- Эффективность 1 потока к 8 потокам составила 720%.

Литература

[1] Энтони Уильямс Параллельное программирование на C++ в действии. Практика разработки многопоточных программ.

[2] Многопоточность [Электронный ресурс]. - Режим доступа: https://ru.bmstu.wiki/Многопоточность, свободный. (Дата обращения: 20.11.2019 г.)

[3] Многопоточность [Письменный ресурс]. - Семинар по анализу алгоритмов. (Дата обращения: 19.11.2019 г.)