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 **Параллельное умножение матриц**

Студент **Саркисов А.С.**

Группа **ИУ7-53Б**

Оценка (баллы) ______________________

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

Москва — 2020 г.

Оглавление

Введение **2**

1 Аналитическая часть **3**

1.1 Алгоритм Винограда 3

1.1.1 Параллельный алгоритм Винограда 4

1.2 Параллельное программирование 4

1.2.1 Организация взаимодействия параллельных потоков 5

1.3 Вывод 5

2 Конструкторская часть **6**

2.1 Схемы алгоритмов 6

2.2 Распараллеливание программы 8

2.3 Вывод 8

3 Технологическая часть **9**

3.1 Выбор ЯП 9

3.2 Сведения о модулях программы 9

3.3 Листинг кода алгоритмов 9

3.4 Вывод 13

4 Исследовательская часть **14**

4.1 Сравнительный анализ на основе замеров времени 14

4.2 Вывод 16

Заключение **17**

Список литературы **17**

Введение

Цель работы: изучение возможности параллельных вычислений и использование такого подхода на практике. Реализация парралельного алгоритма Винограда умножения матриц. В данной лабораторной работе рассматривается алгоритм Винограда и параллельный алгоритм Винограда. Необходимо сравнить зависимость времени работы алгоритма от числа параллельных потоков и размера матриц, провести сравнение стандартного и параллельного алгоритмов.

1 Аналитическая часть

Матрицей А размера $[m * n]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы.[1] Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить. У произведения будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй.

Пусть даны две прямоугольные матрицы А и В размеров $[m * n]$ и $[n * k]$ соответственно. В результате произведение матриц А и В получим матрицу С размера $[m * k]$.

$c_{i,j} = \sum\limits_{r=1}^{n} a_{i,r} \cdot b_{r,j}$ называется произведением матриц А и В [1].

1.1 Алгоритм Винограда

Подход Алгоритма Винограда является иллюстрацией общей методологии, начатой в 1979-х годах на основе билинейных и трилинейных форм, благодаря которым большинство усовершенствований для умножения матриц были получены [2].

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$.

Их скалярное произведение равно (1.1)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.1}$$

Равенство (1.1) можно переписать в виде(1.2)

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{1.2}$$

Менее очевидно, что выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй. Это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.1.1 Параллельный алгоритм Винограда

Трудоемкость алгоритма Винограда имеет сложность $O(nmk)$ для умножения матриц $n1 \times m1$ на $n2 \times m2$. Чтобы улучшить алгоритм, следует распараллелить ту часть алгоритма, которая содержит 3 вложенных цикла.

Вычисление результата для каждой строки не зависит от результата выполнения умножения для других строк. Поэтому можно распараллелить часть кода, где происходят эти действия. Каждый поток будет выполнять вычисления определенных строк результирующей матрицы.

1.2 Параллельное программирование

При использовании многопроцессорных вычислительных систем с общей памятью обычно предполагается, что имеющиеся в составе системы процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти, и время доступа к памяти является одинаковым (при одновременном доступе нескольких процессоров к одному и тому же элементу памяти очередность и синхронизация доступа обеспечивается на аппаратном уровне). Многопроцессорные системы подобного типа обычно именуются симметричными мультипроцессорами (symmetric multiprocessors, SMP).

Перечисленному выше набору предположений удовлетворяют также активно развиваемые в последнее время многоядерные процессоры, в которых каждое ядро представляет практически независимо функциони рующее вычислительное устройство.

Обычный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью – создание новых параллельных методов на основе обычных последовательных программ, в которых или автоматически компилятором, или непосредственно программистом выделяются участки независимых друг от друга вычислений. Возможности автоматического анализа программ для порождения параллельных вычислений достаточно ограничены, и второй подход является преобладающим. При этом для разработки параллельных программ могут применяться как новые алгоритмические языки, ориентированные на параллельное программирование, так и уже имеющиеся языки, расширенные некоторым набором операторов для параллельных вычислений.

Широко используемый подход состоит и в применении тех или иных библиотек, обеспечивающих определенный программный интерфейс (application programming interface, API) для разработки параллельных программ. В рамках такого подхода наиболее известны Windows Thread API. Однако первый способ применим только для ОС семейства Microsoft Windows, а второй вариант API является достаточно трудоемким для использования и имеет низкоуровневый характер [3].

1.2.1 Организация взаимодействия параллельных потоков

Потоки исполняются в общем адресном пространстве параллельной программы. Как результат, взаимодействие параллельных потоков можно организовать через использование общих данных, являющихся доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения. В случае же, когда общие данные могут изменяться несколькими потоками, необходимы специальные усилия для организации правильного взаимодействия.

1.3 Вывод

Был рассмотрен алгоритм Винограда и возможность его оптимизации с помощью распараллеливания потоков. Была рассмотрена технология параллельного программирования и организация взаимодействия параллельных потоков.

2 | Конструкторская часть

Требования к вводу: На вход подаются две матрицы
Требования к программе:

- корректное умножение двух матриц;
- при матрицах неправилыных размеров программа не должна аварийно завершаться.

2.1 Схемы алгоритмов

В данной части будет рассмотрена схема алгоритма Винограда (Рис. 2. 1) и выбранный способ распараллеливания.

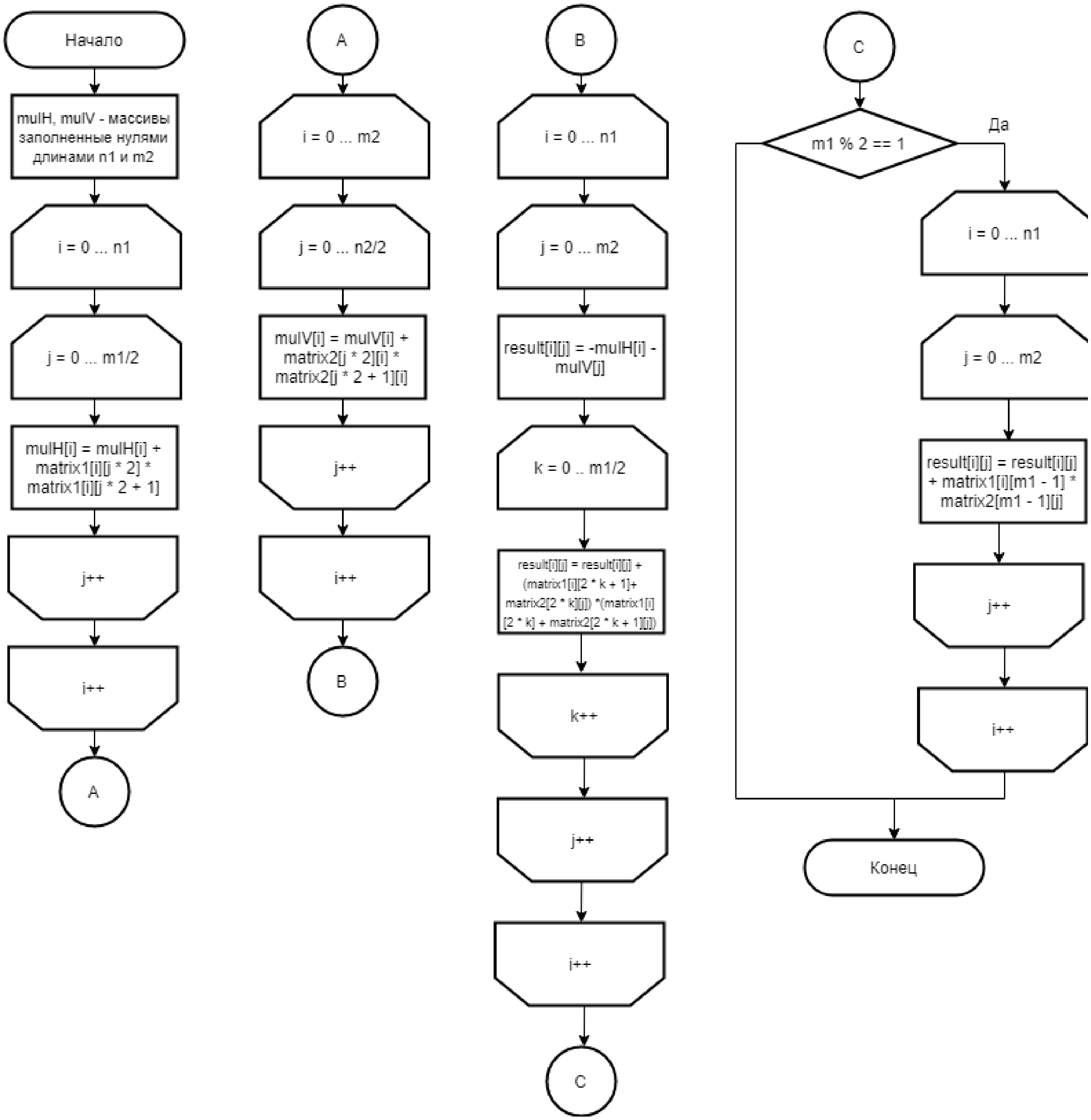

Рис. 2.1: Схема алгоритма Винограда

2.2 Распапараллеливание программы

Распрараллеливание программы должно ускорять время работы. Это достигается за счет реализации в узких участках (напимер в циклах с большим количеством независимых вычилений).

В предложенном алгоритме данным участком будет являться тройной цикл поиска результата. Данный блок программы как раз предлагается распапараллелить. На (Рис. 2.1) это участок между B и C.

2.3 Вывод

В данном разделе была рассмотрена схема алгоритма Винограда и способ ее распапараллеливания.

3 Технологическая часть

Замеры времени были произведены на: 2 GHz Quad-Core Intel Core i5, 4 ядра, 4 логических процессоров.

3.1 Выбор ЯП

В качестве языка программирования был выбран C++ [4] так как этот язык поддерживает управление потоками на уровне ОС. Средой разработки Visual Studio. Время работы алгоритмов было замерено с помощью класса Stopwatch. Многопоточное программирование было реализовано с помощью пространства имен thread.

3.2 Сведения о модулях программы

Программа состоит из:

- main.cpp - главный файл программы, в котором располагается чточка входа в программу и функция замера времени;
- alg.cpp - файл класса alg, в которм находиятся алгоритмы Винограда;

3.3 Листинг кода алгоритмов

Листинг 3.1: Алгоритм Винограда

```
Matrix Vinograd(Matrix A, Matrix B) {
    int N = A.cols();
```

```
    int M = A.rows();
    int Q = B.cols();
    Matrix c(M, Q);

    std::vector<int> MulH(M,0);
    for (int i = 0; i < M; i++) {
        for (int k = 0; k < N-1; k += 2) {
        MulH[i] -= A[i][k] * A[i][k+1]; }
    }
    std::vector<int> MulV(Q,0);
    for (int i = 0; i < Q; i++) {
        for (int k = 0; k < N-1; k += 2) {
            MulV[i] -= B[k][i]*B[k+1][i];
        }
    }
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < Q; j++) {
            int buf = MulH[i] + MulV[j];
            for (int k = 0; k < N-1; k += 2) {
                buf += (A[i][k] + B[k+1][j])*(A[i][k+1] + B
                   [k][j]);
            }
            if (N%2 == 1) {
                buf += A[i][N-1]*B[N-1][j];
            }
            c[i][j] = buf;
        }
    }
    return c;
}
```

Листинг 3.2: Первый шаг алгоритма Винограда

```
void first (std::vector<int> &MulH, Matrix A, int N, int
   start, int end) {
    for (int i = start; i < end; i++) {
        for (int k = 0; k < N-1; k += 2) {
            MulH[i] -= A[i][k] * A[i][k+1];
        }
    }
}
```

Листинг 3.3: Второй шаг алгоритма Винограда

```
void second (std::vector<int> &MulV, Matrix B, int N, int
    start, int end) {
     for (int i = start; i < end; i++) {
          for (int k = 0; k < N-1; k += 2) {
               MulV[i] -= B[k][i]*B[k+1][i];
          }
     }
}
```

Листинг 3.4: Третий шаг алгоритма Винограда

```
void third (Matrix &c, Matrix A, Matrix B, int N, std::
    vector<int> MulH, std::vector<int> MulV, int starti, int
     endi, int startj, int endj) {
     for (int i = starti; i < endi; i++) {
          for (int j = startj; j < endj; j++) {
               int buf = MulH[i] + MulV[j];
               for (int k = 0; k < N-1; k += 2) {
                    buf += (A[i][k] + B[k+1][j])*(A[i][k+1] + B
                       [k][j]);
               }
               if (N%2 == 1) {
               buf += A[i][N-1]*B[N-1][j];
               }
               c[i][j] = buf;
          }
     }
}
```

Листинг 3.5: Функция распараллеливания алгоритма Винограда

```
Matrix Mul(Matrix A, Matrix B, int tcount) {
     int N = A.cols();
     int M = A.rows();
     int Q = B.cols();
     Matrix c(M, Q);
     std::vector<int> MulH(M,0);
     std::vector<int> MulV(Q,0);
```

```
    thread *th[tcount];
    int n = tcount/2;
    for (int i = 0; i < n; i++) {
        th[i] = new thread(first, ref(MulH), A, N, i*M/n, (
            i+1)*M/n);
    }
    int m = tcount - n;
    for (int i = 0; i < m; i++) {
        th[n+i] = new thread(first, ref(MulV), B, N, i*M/m,
            (i+1)*M/m);
    }
    for (int i = 0; i < tcount; i++) {
        th[i]->join();
    }
    thread *th3[tcount];
    for (int i = 0; i < tcount; i++) {
        th3[i] = new thread(third,
        ref(c),
        A, B, N,
        MulH, MulV,
        i*M/tcount, (i+1)*M/tcount,
        0, Q);
    }
    for (int i = 0; i < tcount; i++) {
        th3[i]->join();
    }
    for (int i = 0; i < tcount; i++) {
        delete th[i];
        delete th3[i];
    }
    return c;
}
```

Листинг 3.6: Функция распараллеливания алгоритма Винограда

```
Matrix Mul(Matrix A, Matrix B, int tcount) {
    int N = A.cols();
    int M = A.rows();
    int Q = B.cols();
    Matrix c(M, Q);
    std::vector<int> MulH(M,0);
```

```
    std::vector<int> MulV(Q,0);
    thread *th[tcount];
    int n = tcount/2;
    for (int i = 0; i < n; i++) {
        th[i] = new thread(first, ref(MulH), A, N, i*M/n, (
            i+1)*M/n);
    }
    int m = tcount - n;
    for (int i = 0; i < m; i++) {
        th[n+i] = new thread(first, ref(MulV), B, N, i*M/m,
             (i+1)*M/m);
    }
    for (int i = 0; i < tcount; i++) {
        th[i]->join();
    }
    thread *th3[tcount];
    for (int i = 0; i < tcount; i++) {
        th3[i] = new thread(third,
        ref(c),
        A, B, N,
        MulH, MulV,
        i*M/tcount, (i+1)*M/tcount,
        0, Q);
    }
    for (int i = 0; i < tcount; i++) {
        th3[i]->join();
    }
    for (int i = 0; i < tcount; i++) {
        delete th[i];
        delete th3[i];
    }
    return c;
}
```

3.4 Вывод

В данном разделе были рассмотрены основные сведения о модулях программы, листинг кода.

4 Исследовательская часть

4.1 Сравнительный анализ на основе замеров времени

Был проведен замер времени работы алгоритма на 1, 2, 4, 8, 16, 32 и 64 потоков для матриц разного размера.

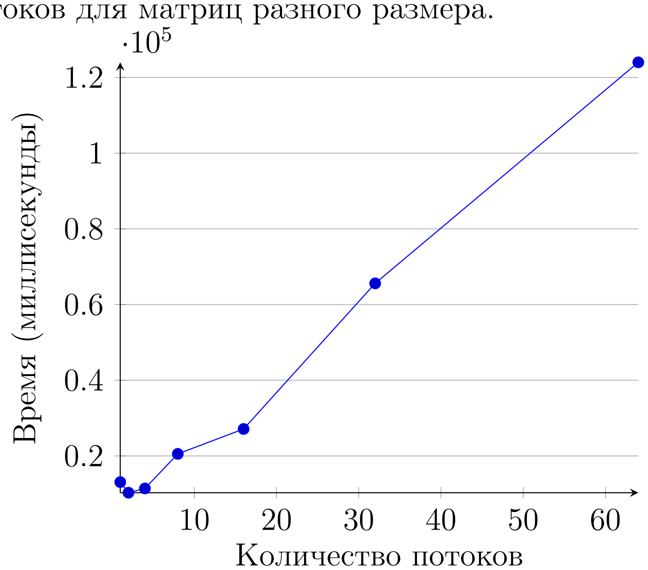

Рис. 4.1: Сравнение времени работы алгоритма для матрицы 100 на 100

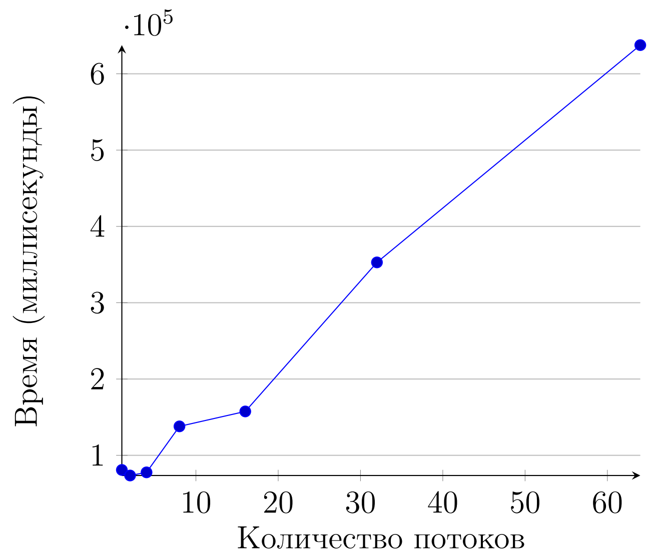

Рис. 4.2: Сравнение времени работы алгоритма для матрицы 200 на 200

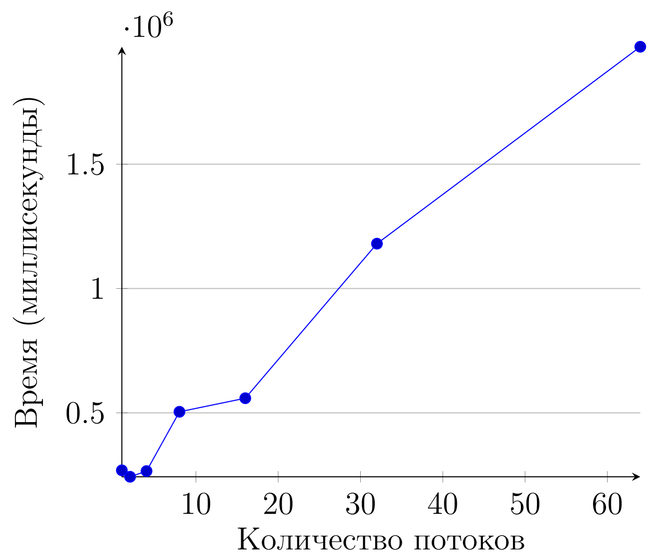

Рис. 4.3: Сравнение времени работы алгоритма для матрицы 300 на 300

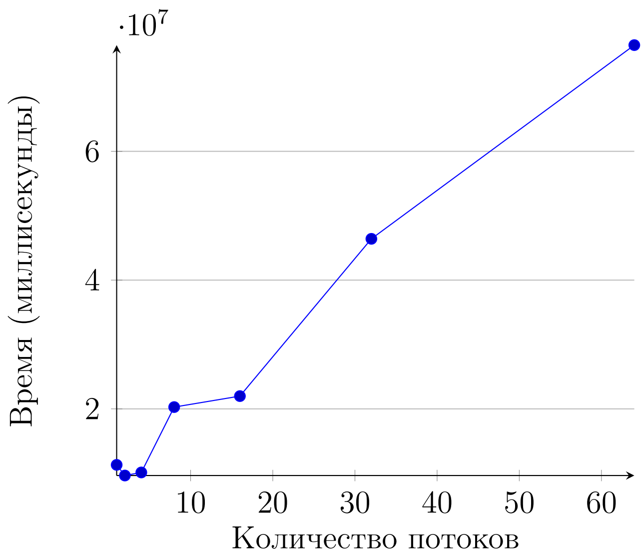

Рис. 4.4: Сравнение времени работы алгоритма для матрицы 1000 на 1000

4.2 Вывод

По результатам исследования на графиках видно, что алгоритм на любом количестве элементов матрицы улучшает свою производительность до тех пор, пока количество потоков не превысит количество логических потоков. Работа алгоритма на двух потоках в среднем на 8 - 9 процентов быстрее, чем на одном, в свою очередь работа на четырёх потоках достигает разницы 3 - 4 процента в этом же сравнении. После подключения более 4 потоков скорость алгоритма сильно ухудшается, заметна деградация времени в размере 80 - 85 процентов при переходе с 4 до 8 потоков, после чего скорость уменьшается в среднем на 40 процентов при переходе к последующем значениям числа потоков.

Заключение

В ходе лабораторной работы я изучил возможности параллельных вычислений и использовал такой подход на практике. Реализовал алгоритм Винограда умножения матриц с помощью параллельных вычислений.

Был проведен эксперимент для поиска оптимального количества потоков, в результате которого оказалось, что эффективнее всего использовать количество потоков равное количеству логических потоков.

Литература

[1] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16

[2] Le Gall, F. (2012), "Faster algorithms for rectangular matrix multiplication Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), pp. 514–523

[3] Константин Баркалов, Владимир Воеводин, Виктор Гергель. Intel Parallel Programming [Электронный ресурс], - режим доступа https://www.intuit.ru/studies/courses/4447/983/lecture/14925

[4] Руководство по языку C++[Электронный ресурс], - режим доступа: https://docs.microsoft.com/ru-ru/dotnet/cpp/