Федеральное государственное
бюджетное образовательное
учреждение высшего образования
«Московский государственный
технический унимерситет имени
Н.Э. Баумана (национальный
исследовательский университет)»
МГТУ им. Баумана

Лабораторная работа №4

По курсу: "Анализ алгоритмов"

Параллельное умножение матриц

Работу выполнила: Расколотов Д.Ю., ИУ7-54Б

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

Москва, 2020

Оглавление

Введение

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

В ходе лабораторной работы предстоит:

- оптимизировать алгоритм Винограда;
- распараллелить оптимизированный алгоритм Винограда 2-мя способами;
- сравнить зависимость времени работы алгоритма от числа параллельных потоков и размера матриц.

1 Аналитическая часть

Матрицей A размера $[n * m]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая n строк и m столбцов. Числа n и m 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 A и B размеров $[n * m]$ и $[m*k]$ соответственно. В результате произведение матриц A и B получим матрицу C размера $[n * k]$.

$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j}$ называется произведением матриц A и B [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)

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

Кажется, что второе выражение задает больше работы, чем первое: вместо четырех умножений мы насчитываем их шесть, а вместо трех сложений - десять. 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 Оптимизация алгоритма Винограда

В рамках данной лабораторной работы сделано 4 оптимизации:

1. Избавление от деления в условии цикла;
2. Замена $mulH[i] = mulH[i] + \ldots$ на $mulH[i] += \ldots$ (аналогично для $mulV[i]$);
3. Накопление результата в буфер, чтобы не обращаться каждый раз к одной и той же ячейке памяти. Сброс буфера в ячейку матрицы после цикла.
4. Проверка на чётность выполняется сразу же при вычислении буфера.

1.1.2 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, а так же части вычесляемые заранее.

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.

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 специаль-

ные 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. Основываясь на приведённых выше технологиях параллельного программирования и организации взаимодействия параллельных потоков, зная особенности устройства своего компьютера и технологию hyper-T[4], мы получаем из 4-х физических ядер - 8 виртуальных. Исходя из этого можно предположить, что лучшие результаты должны быть получены на 16 потоках.

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 выбранные спросбы распараллеливания.

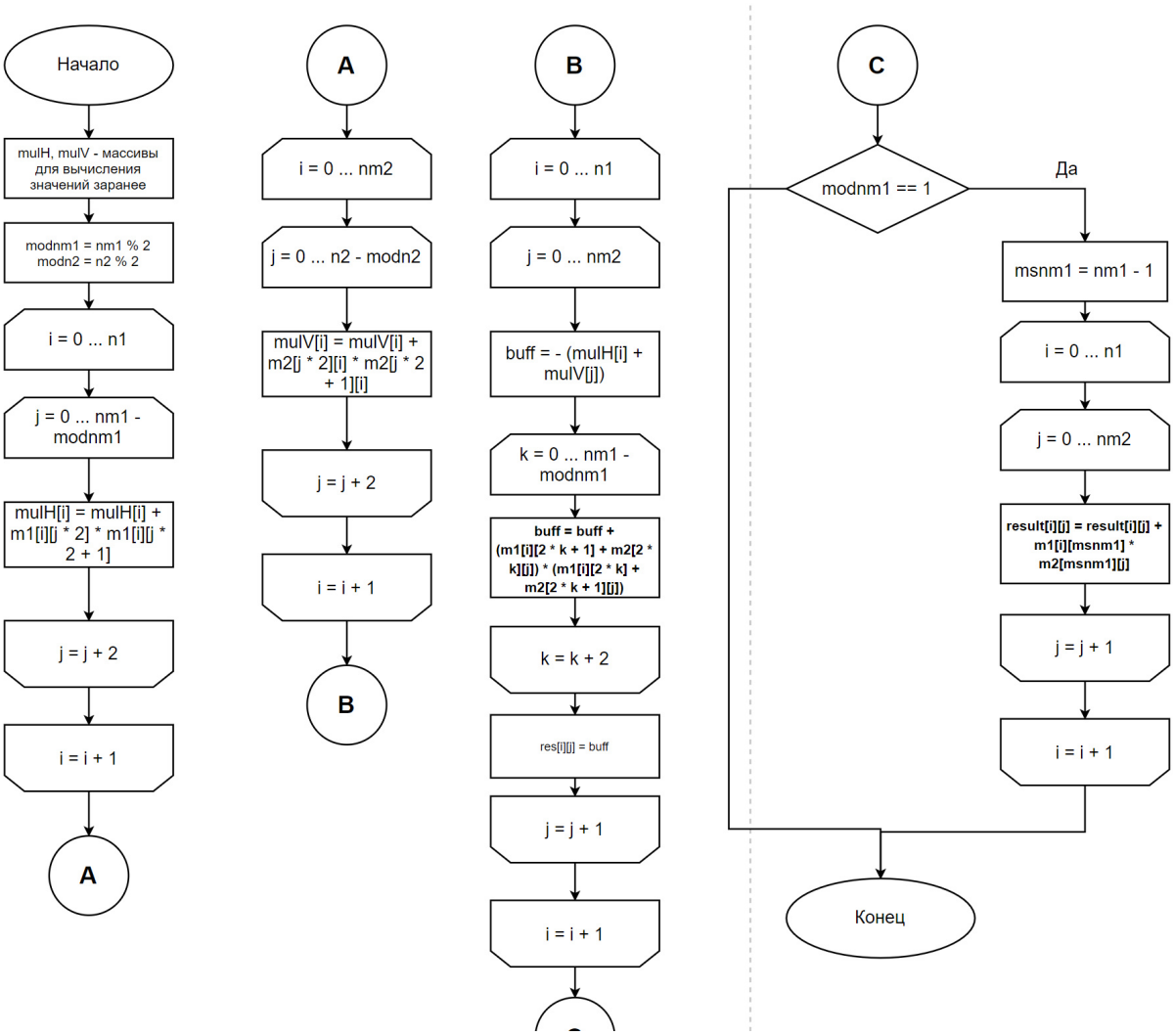

Рис. 2.1: Схема умножения матриц алгоритмом Винограда

2.2 Распараллеливание программы

Распараллеливание 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, а так же участки вычисления дополнительных параметров. Данный блок программы как раз предлагается распараллелить. На (Рис. 2.1) это участки:

- Участок между B и C - являеться тройной цикл поиска результата;
- Двойные циклы поиска дополнительных параметров, такие как участок между A и B.

2.3 Вывод

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

3 | Технологическая часть

Замеры времени были произведены на: Intel(R) Core(TM) i7-10510U, 4 ядра, 8 логических процессоров.

3.1 Выбор ЯП

В качестве языка программирования был выбран C++, так как этот язык поддерживает управление потоками на уровне ОС(незеленые потоки). Средой разработки Visual Studio Code. Время работы алгоритмов было замерено с помощью класса Ctime. Многопоточное программирование было реализовано с помощью класса Thread.

3.2 Листинг кода алгоритмов

Листинг 3.1: Оптимизированный алгоритм Винограда

```
Matrix MultiplicationVinograd(Matrix a, Matrix b)
{
    int n = a.size();
    int ns = a[0].size();

    int m = b.size();
    int ms = b[0].size();

    Matrix mn(n);
    Array mulH(n, 0);
    Array mulV(ms, 0);

    Thread threads;

    for (int i = 0; i < n; i++)
    for (int j = 0; j < ms; j++)
    mn[i].push_back(0);

    int modns = ns % 2;
    int modm = m % 2;

    for (int i = 0; i < n; i++)
    for (int j = 0; j < ns - modns; j += 2)
    mulH[i] = mulH[i] + a[i][j] * a[i][j + 1];

    for (int i = 0; i < ms; i++)
    for (int j = 0; j < m - modm; j += 2)
    mulV[i] = mulV[i] + b[j][i] * b[j + 1][i];

    int minam = ns - 1;
    for (int i = 0; i < n; i++)
    for (int j = 0; j < ms; j++)
    {
        int buff = -mulH[i] - mulV[j];
        for (int k = 0; k < ns / 2; k++)
        buff = buff + (a[i][2 * k + 1] + b[2 * k][j
            ]) * (a[i][2 * k] + b[2 * k + 1][j]);
        if (modns)
        buff = buff + a[i][minam] * b[minam][j];
```

```
39                      mn[i][j] = buff;
40                  }
41
42                  return mn;
43              }
```

Листинг 3.2: Распараллеливание двойных циклов в оптимизированном алгоритме Винограда

```
1     void threadMulH(Array &mulH, Matrix a, int n, int ns,
          int modns)
2     {
3         for (int i = 0; i < n; i++)
4         for (int j = 0; j < ns - modns; j += 2)
5         mulH[i] = mulH[i] + a[i][j] * a[i][j + 1];
6     }
7
8     void threadMulV(Array &mulV, Matrix b, int ms, int m,
          int modm)
9     {
10        for (int i = 0; i < ms; i++)
11        for (int j = 0; j < m - modm; j += 2)
12        mulV[i] = mulV[i] + b[j][i] * b[j + 1][i];
13    }
14
15    Matrix ParallelMultiplicationVinograd_1(Matrix a,
          Matrix b)
16    {
17        int n = a.size();
18        int ns = a[0].size();
19
20        int m = b.size();
21        int ms = b[0].size();
22
23        Matrix mn(n);
24        Array mulH(n, 0);
25        Array mulV(ms, 0);
26
27        Thread threads;
28
29        for (int i = 0; i < n; i++)
30        for (int j = 0; j < ms; j++)
31        mn[i].push_back(0);
32
33        int modns = ns % 2;
34        int modm = m % 2;
35
```

```
36          thread thread_MulH(threadMulH, ref(mulH), a, n, ns,
                modns);
37          thread thread_MulV(threadMulV, ref(mulV), b, ms, m,
                modm);
38
39          thread_MulH.join();
40          thread_MulV.join();
41
42          int minam = ns - 1;
43          for (int i = 0; i < n; i++)
44          for (int j = 0; j < ms; j++)
45          {
46              int buff = -mulH[i] - mulV[j];
47              for (int k = 0; k < ns / 2; k++)
48              buff = buff + (a[i][2 * k + 1] + b[2 * k][j]) *
                    (a[i][2 * k] + b[2 * k + 1][j]);
49              if (modns)
50              buff = buff + a[i][minam] * b[minam][j];
51              mn[i][j] = buff;
52          }
53
54          return mn;
55      }
```

Листинг 3.3: Распараллеливание оптимизированного алгоритма Винограда

```
1
2     void threadMultiply(Array mulV, Array mulH, Matrix a,
          Matrix b, Matrix &mn, int start, int end, int ns,
          int ms, int modns)
3     {
4         int minam = ns - 1;
5         for (int i = start; i < end; i++)
6             for (int j = 0; j < ms; j++)
7             {
8                 int buff = -mulH[i] - mulV[j];
9                 for (int k = 0; k < ns / 2; k++)
10                    buff = buff + (a[i][2 * k + 1] + b[2 *
                          k][j]) * (a[i][2 * k] + b[2 * k +
                          1][j]);
11                if (modns)
12                    buff = buff + a[i][minam] * b[minam][j
                          ];
13                mn[i][j] = buff;
14            }
15    }
16
17    Matrix ParallelMultiplicationVinograd_2(Matrix a,
          Matrix b, int thr)
18    {
19        int n = a.size();
20        int ns = a[0].size();
21
22        int m = b.size();
23        int ms = b[0].size();
24
25        Matrix mn(n);
26        Array mulH(n, 0);
27        Array mulV(ms, 0);
28
29        Thread threads;
30
31        for (int i = 0; i < n; i++)
32            for (int j = 0; j < ms; j++)
```

```
33                  mn[i].push_back(0);
34
35          int modns = ns % 2;
36          int modm = m % 2;
37
38          thread thread_MulH(threadMulH, ref(mulH), a, n, ns,
                modns);
39          thread thread_MulV(threadMulV, ref(mulV), b, ms, m,
                modm);
40
41          thread_MulH.join();
42          thread_MulV.join();
43
44          for (int k = 1; k <= thr; k++)
45              threads.push_back(thread(threadMultiply, mulV,
                    mulH, a, b, ref(mn), (k - 1) * n / thr, k *
                    n / thr, ns, ms, modns));
46
47          for (auto &th : threads)
48              th.join();
49
50          return mn;
51      }
```

3.3 Вывод

В данном разделе была рассмотрена структура ПО и листинги кода программы.

4 Исследовательская часть

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Был проведен замер времени работы каждого из алгоритмов.

В первом эксперименте я решил сравнить эффективность распараллеливания оптимизированного алгоритма Винограда 1-ым способом и оптимизированный алгоритм Винограда. В первом случае производились замеры распараллеливания двойных циклов поиска дополнительных параметров.

Первый эксперимент производился на квадратных матрицах с чётными разамерами от 100 x 100 до 1000 x 1000 с шагом 100. Сравним результаты распараллеливания:

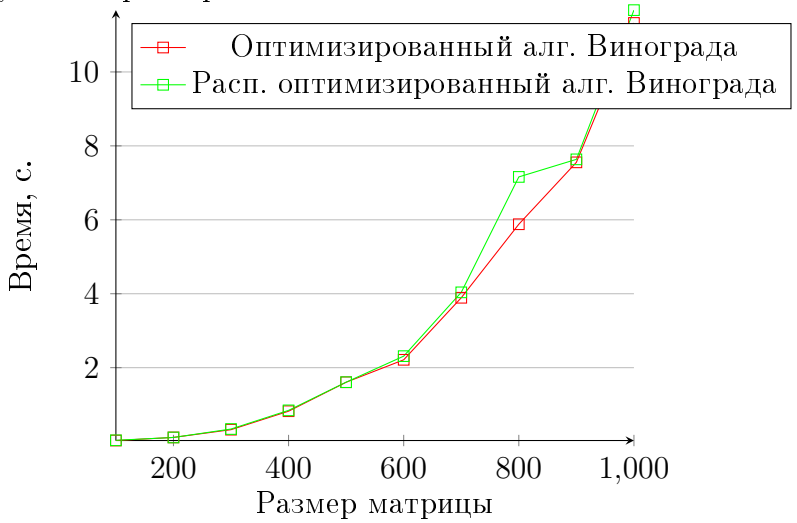

Рис. 4.1: Сравнение времени работы алгоритмов при четном размере

матрицы

Второй эксперимент производился для оценки эффективности многопоточности для второго способа распараллеливания, когда поданы квадратные матрицы с чётным размером 700x700, кол-во потоков от 2 до 128 с шагом 2^n. В этом случае производились замеры распараллеливания двойных циклов и тройного цикла поиска результата.
Сравним результаты выполнения:

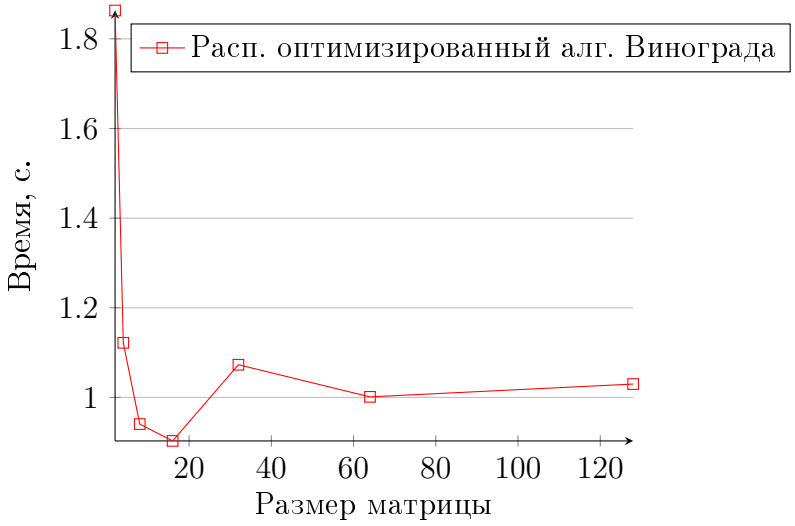

Рис. 4.2: Сравнение времени работы алгоритма распараллеливания для разных потоков

Третий эксперимент производился для оценки эффективности второго способа распараллеливания, когда поданы квадратные матрицы с чётным размером от 100 x 100 до 1000 x 1000 с шагом 100 для 8-ми потоков.
Сравним результаты выполнения:

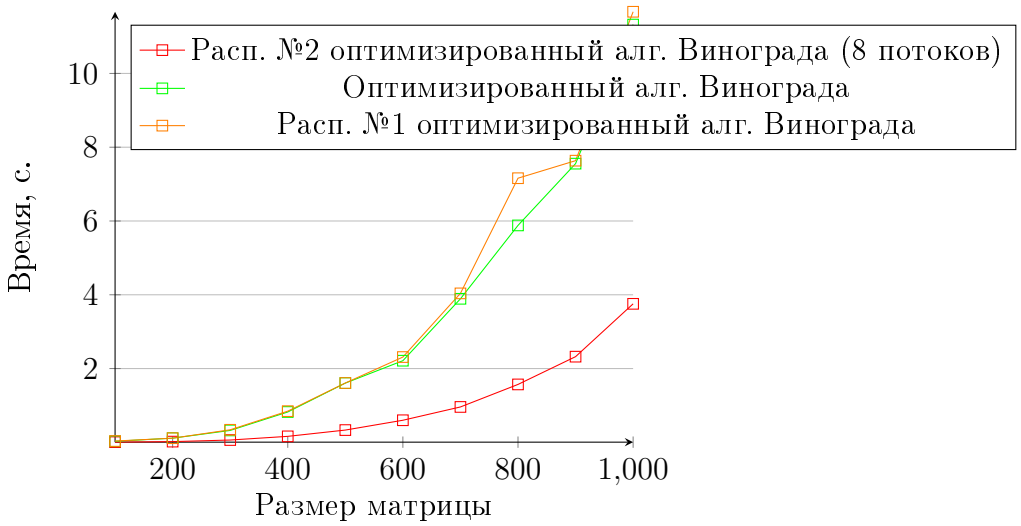

Рис. 4.3: Сравнение времени работы алгоритмов распараллеливания на 16 потоках.

4.2 Вывод

По результатам тестирования, основываясь на эксперименте №1 (рис. 4.1), можно понять, что нужды в распараллеливании двойных циклов поиска дополнительных параметров - нет. Так как это менее эффективно простого оптимизированного алг. Винограда из-за затрат времени на выделение потоков и работы с ними.
Что касательно последних двух экспериментов (рис. 4.2 и 4.3), то существенно выделяется разница в эффективности распараллеливания тройного цикла поиска результата, особенно на 16 потоках.

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.

Были проведены эксперименты для поиска оптимального способа распараллеливания и сравнения их эффективности.

Также был проведен анализ целесообразности распараллеливания циклов поиска mulH mulV, в результате которого оказалось, что при таком подходе разница результатов при увеличении размера матрицы ухудшается на 1% - 2%.

Ввиду результатов предыдущих исследовиний следующий эксперимент проводился с использованием более продуманной оптимизацией распараллеливания, распараллеливание было произведено как для главного цикла поиска результата, так и для циклов mulH и mulV. Было произведено сравнение работы оптимизированного алгоритма Винограда и параллельной реализации при увеличении количества потоков. Выяснилось, что при увеличение потоков до 16 сокращает время работы на 70% по сравнению с однопоточной реализацией. Однако дальнейшее увеличение количества потоков не дает значительного выигрыша по времени, а наоборот ухудшаются показатели.

Цель достигнута и все задачи выполнены

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] Алгоритмы умножения матриц: http://www.algolib.narod.ru/Math/Matrix.html

[4] Hyper-T: https://en.wikipedia.org/wiki/Hyper-threading