МГТУ им. Баумана

Лабораторная работа №4

По курсу: "Анализ алгоритмов"

Исследование работы алгоритма Винограда для умножения матриц реализованного при помощи параллельных вычислений

Работу выполнил: Подвашецкий Дмитрий, ИУ7-54Б

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

Москва, 2019

Оглавление

Введение

Матрица - математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы.

Матрицы широко применяются в математике для компактной записи систем линейных алгебраических или дифференциальных уравнений. В этом случае, количество строк матрицы соответствует числу уравнений, а количество столбцов — количеству неизвестных. В результате решение систем линейных уравнений сводится к операциям над матрицами.

Матрицы допускают следующие алгебраические операции:

1. сложение матриц, имеющих один и тот же размер;
2. умножение матриц подходящего размера;
3. умножение матрицы на элемент основного кольца или поля;

Умножение матриц - одна из основных операций над матрицами.

Целью данной лабораторной работы является реализация и изучение алгоритма умножения матриц методом Винограда и стандартного.

Задачами данной лабораторной являются:

1. оптимизировать алгоритм Винограда при помощи распараллеливания вычислений;
2. исследовать поведение функции при обработке различным числом потоков.

1 Аналитическая часть

Операция умножения матриц повсеместно приминяется в математике, физике, программировании и т.д. Для того, чтобы произведение матрицы A, размерами n на m, на матрицу B, размерами u на v, было возможно, необходимо, чтобы n = u.

В алгоритме умножения матриц методом Винограда [1] каждый элемент производной матрицы считается как скалярное произведение. Рассмотрим два вектора:

$$V = \begin{pmatrix} v1 & v2 & v3 & v4 \end{pmatrix}$$
$$W = \begin{pmatrix} w1 & w2 & w3 & w4 \end{pmatrix}$$

Их скалярное произведение равно:

$$V \cdot W = v1w1 + v2w2 + v3w3 + v4w4 \tag{1.1}$$

Выражение (1.1) можно переписать:

$$V \cdot W = (v1 + w2)(v2 + w1) + (v3 + v4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4 \tag{1.2}$$

Можно заметить, что правую часть данного выражения можно высчитать заранее для каждого вектора. Это означает, что, при предварительной обработке векторов мы можем, в дальнейшем, сэкономить 2 операции умножения, за счет 2х лишних операций сложения.

Если при умножении двух матриц произвести обработку строк первой и столбцов второй, то можно добиться большей эффективности по времени.

Вывод

В данном разделе был описан алгоритм Винограда.

2 Конструкторская часть

2.1 Схема алгоритма

На Рисунке 1. изображена схема алгоритма Винограда.

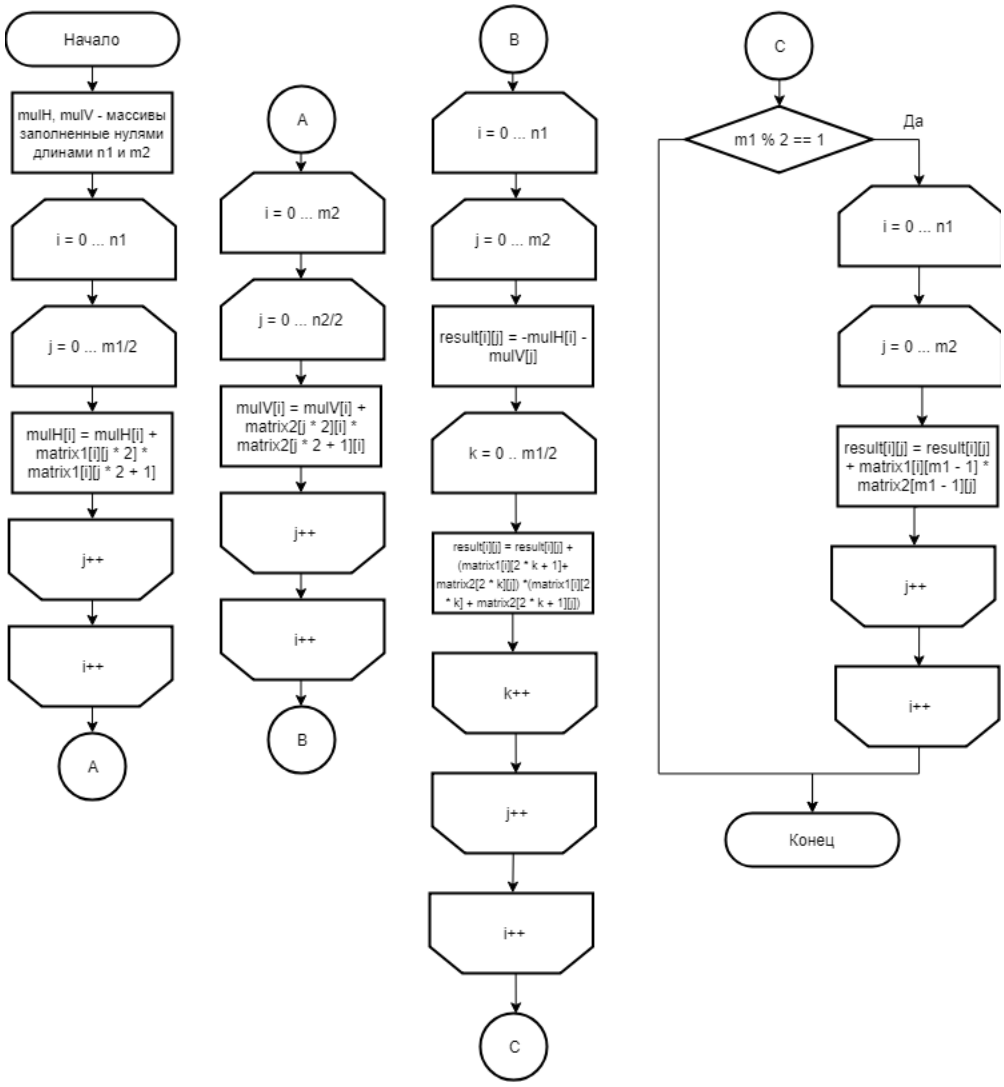

Рисунок 1. Схема алгоритма Винограда.

Вывод

В данном разделе предсталена схема рассматриваемого в данной лабораторной работе алгоритма.

3 Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбрал C++ [2], так как он позволяет реализовать задачу максимально комфортно. Для реализации многопоточности выбрана библиотека <threads> [3].

3.2 Замеры времени

Замер времени работы алгоритмов производился при помощи функций clock() из библиотеки time.h [4].

3.3 Генерация матриц для экспериментов

Все матрицы будут сгенерированы при помощи функции, создающей матрицу заданного размера, полностью заполненную единицами.

3.4 Листинг кода

Листинг 3.1. Реализация многопоточного умножения матриц методом Винограда.

```
1  #include "winmult.h"
2
3  void thrdMultiply(mtr &res, const mtr &A, const mtr &B, int st, int
       end, std::mutex &mtx)
4  {
5      size_t B_rows = B.size(), B_cols = B[0].size();
6
7      bool uneven = (B_rows % 2 != 0 ? true : false);
8
9      for (size_t i = size_t(st); i < size_t(end); i++)
10     {
11         for (size_t j = 0; j < B_cols; j++)
12         {
13             for (size_t k = 0; k < B_rows-1; k+=2)
```

```
14            {
15
16                res[i][j] += (A[i][k] + B[k+1][j])*(A[i][k+1] +
                      B[k][j]);
17
18            }
19
20            if (uneven)
21                res[i][j] += A[i][B_rows-1] * B[B_rows-1][j];
22        }
23    }
24 }
25
26 mtr multiply(mtr &A, mtr &B, size_t thdN)
27 {
28    std::mutex mtx;
29
30    size_t A_rows = A.size(), A_cols = A[0].size();
31    size_t B_rows = B.size(), B_cols = B[0].size();
32    mtr res;
33
34    if (A_cols != B_rows)
35    {
36        return res;
37    }
38
39    std::vector<double> preCalc1;
40    std::vector<double> preCalc2;
41
42    for (size_t i = 0; i < A_rows; i++)
43    {
44        double sum = 0;
45
46        for (size_t j = 0; j < (A_cols % 2 == 0 ? A_cols : A_cols -
              1); j += 2)
47        {
48            sum += A[i][j]*A[i][j+1];
49        }
50
51        preCalc1.push_back(-1*sum);
52    }
53
54    for (size_t i = 0; i < B_cols; i++)
55    {
56        double sum = 0;
57
58        for (size_t j = 0; j < (B_rows % 2 == 0 ? B_rows : B_rows -
```

```
            1); j += 2)
59          {
60              sum += B[j][i]*B[j+1][i];
61          }
62
63          preCalc2.push_back(-1*sum);
64      }
65
66      for (size_t i = 0; i < A_rows; i++)
67      {
68          std::vector<double> tmp;
69          for (size_t j = 0; j < B_cols; j++)
70          {
71              tmp.push_back(preCalc1[i] + preCalc2[j]);
72          }
73
74          res.push_back(tmp);
75      }
76
77      if (thdN > A_rows)
78          thdN = A_rows;
79
80      int step = int(A_rows / thdN);
81
82      std::vector<std::thread> thds;
83
84      int st;
85      int end;
86
87      for (int i = 0; i < int(thdN); i++)
88      {
89          st = step*i;
90          end = step*(i+1);
91          if (i == int(thdN - 1))
92              end = int(A_rows);
93
94          std::thread thread(thrdMultiply, std::ref(res), std::ref(A),
               std::ref(B), st, end, std::ref(mtx));
95          thds.push_back(std::move(thread));
96      }
97
98      for (auto &thd : thds)
99          if (thd.joinable())
100             thd.join();
101
102     return res;
103 }
```

Вывод

В данном разделе обоснован выбор языка, описаны средства работы с многопоточностью, описан способ замера времени. Также реализован рассматриваемый алгоритм и представлен его литинг.

4 Эксперементальная часть

Исследуем работу алгоритма и сравним время выполнения при обработке матриц размером $100 \times 100, \ldots, 1000 \times 1000$ и $101 \times 101, \ldots, 1001 \times 1001$ с шагом 100 при обработке 1, 2, ... , 64 потоками. Время будем замерять в секундах.

Эксперимент проводоился на следующей системе:

1. Intel(R) Core(TM) i7-3770K
2. 8.00 ГБ ОЗУ

4.1 Сравнительный анализ времени работы алгоритма

4.1.1 Четная размерность

В Таблице 1. - Таблице 7. представлены времена работы алгоритмов в зависимости от размерности при различном количестве потоков.

Таблица 1. Временя работы алгоритма при одном потоке.

Размер	Время (с)
100	0.014000
200	0.089000
300	0.295000
400	0.741000
500	1.506000
600	3.650000
700	5.656000
800	8.845000
900	13.468000
1000	21.052000

Таблица 2. Временя работы алгоритма при двух потоках.

Размер	Время (с)
100	0.018000
200	0.047000
300	0.155000
400	0.384000
500	0.768000
600	1.398000
700	2.627000
800	3.768000
900	6.208000
1000	10.216000

Таблица 3. Временя работы алгоритма при четырех потоках.

Размер	Время (с)
100	0.010000
200	0.027000
300	0.095000
400	0.225000
500	0.502000
600	0.783000
700	1.238000
800	2.041000
900	3.434000
1000	5.190000

Таблица 4. Временя работы алгоритма при восьми потоках.

Размер	Время (с)
100	0.010000
200	0.043000
300	0.086000
400	0.204000
500	0.464000
600	0.717000
700	1.174000
800	1.805000
900	2.983000
1000	4.321000

Таблица 5. Временя работы алгоритма при шестнадцати потоках.

Размер	Время (с)
100	0.015000
200	0.031000
300	0.106000
400	0.232000
500	0.551000
600	0.944000
700	1.310000
800	1.884000
900	3.215000
1000	4.963000

Таблица 6. Временя работы алгоритма при тридцати двух потоках.

Размер	Время (с)
100	0.027000
200	0.040000
300	0.109000
400	0.264000
500	0.529000
600	0.991000
700	1.672000
800	2.211000
900	3.180000
1000	5.12000

Таблица 7. Временя работы алгоритма при шестидесяти четырех потоках.

Размер	Время (с)
100	0.052000
200	0.054000
300	0.127000
400	0.231000
500	0.584000
600	0.853000
700	1.726000
800	2.218000
900	2.995000
1000	4.856000

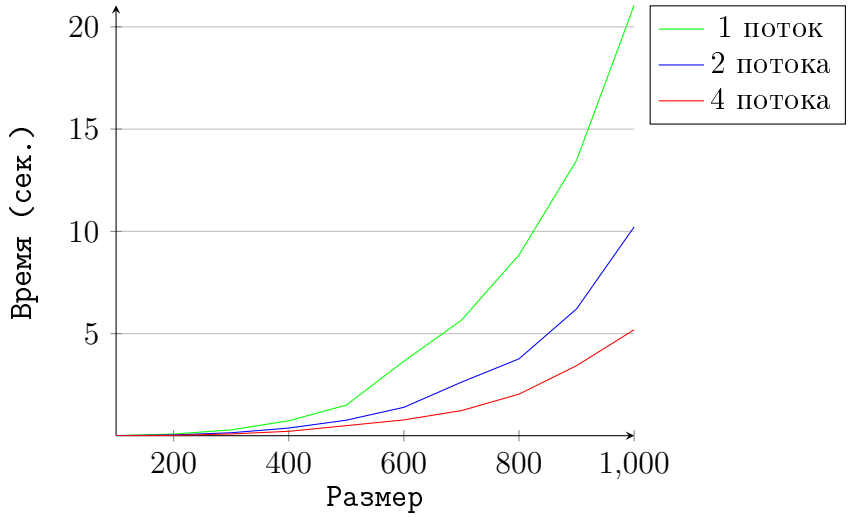

Рисунок 1. График времен работы алгоритма при 1, 2, 4 потоках.

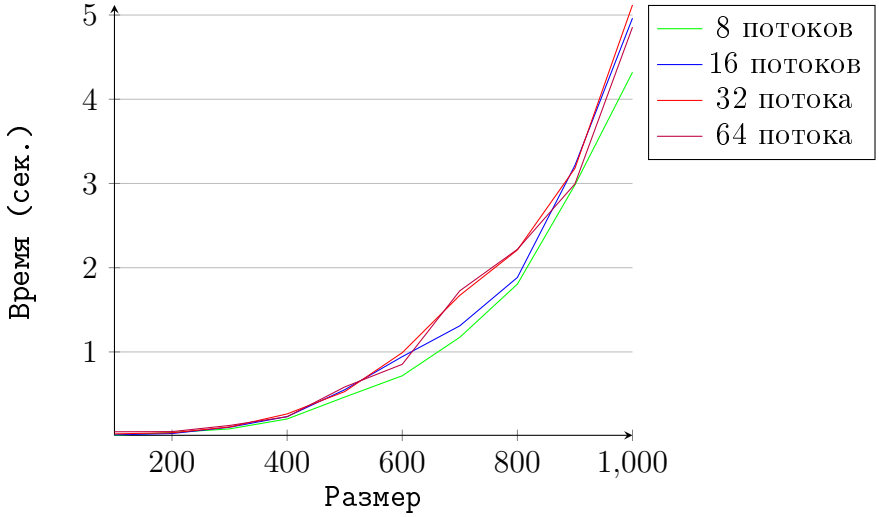

Рисунок 2. График времен работы алгоритма при 8, 16, 32, 64 потоках.

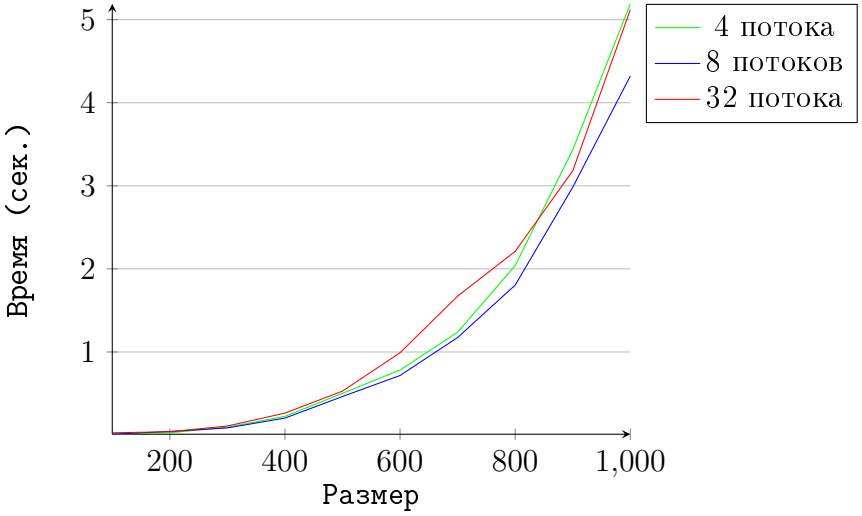

Рисунок 3. График времен работы алгоритма при 4, 8, 32 потоках.

Вывод по данному случаю:

Из Рисунка 3. и Рисунка 2. видно, что при восьми потоках время работы данного алгоритма наименьшее.

Время работы алгоритма при восьми потоках на матрицах размера 1000x1000 составляет 4.32 с, при 800x800 - 1.8 с.

4.1.2 Нечетная размерность

В Таблице 8. - Таблице 14. представлены времена работы алгоритмов в зависимости от размерности при различном количестве потоков.

Таблица 8. Временя работы алгоритма при одном потоке.

Размер	Время (с)
101	0.022000
201	0.092000
301	0.324000
401	0.826000
501	1.752000
601	2.821000
701	4.847000
801	9.657000
901	13.795000
1001	21.118000

Таблица 9. Временя работы алгоритма при двух потоках.

Размер	Время (с)
101	0.009000
201	0.062000
301	0.176000
401	0.429000
501	0.999000
601	1.783000
701	2.334000
801	3.616000
901	6.006000
1001	10.169000

Таблица 10. Временя работы алгоритма при четырех потоках.

Размер	Время (с)
101	0.026000
201	0.051000
301	0.097000
401	0.248000
501	0.549000
601	0.915000
701	1.282000
801	1.911000
901	3.202000
1001	5.034000

Таблица 11. Временя работы алгоритма при восьми потоках.

Размер	Время (с)
101	0.010000
201	0.029000
301	0.093000
401	0.217000
501	0.449000
601	0.781000
701	1.209000
801	1.912000
901	2.808000
1001	4.457000

Таблица 12. Временя работы алгоритма при шестнадцати потоках.

Размер	Время (с)
101	0.015000
201	0.035000
301	0.101000
401	0.226000
501	0.486000
601	0.793000
701	1.278000
801	1.790000
901	2.718000
1001	4.252000

Таблица 13. Временя работы алгоритма при тридцати двух потоках.

Размер	Время (с)
101	0.026000
201	0.040000
301	0.116000
401	0.269000
501	0.534000
601	0.934000
701	1.396000
801	1.809000
901	3.021000
1001	4.939000

Таблица 14. Временя работы алгоритма при шестидесяти четырех потоках.

Размер	Время (с)
101	0.054000
201	0.056000
301	0.134000
401	0.263000
501	0.680000
601	0.934000
701	1.773000
801	2.347000
901	3.086000
1001	4.991000

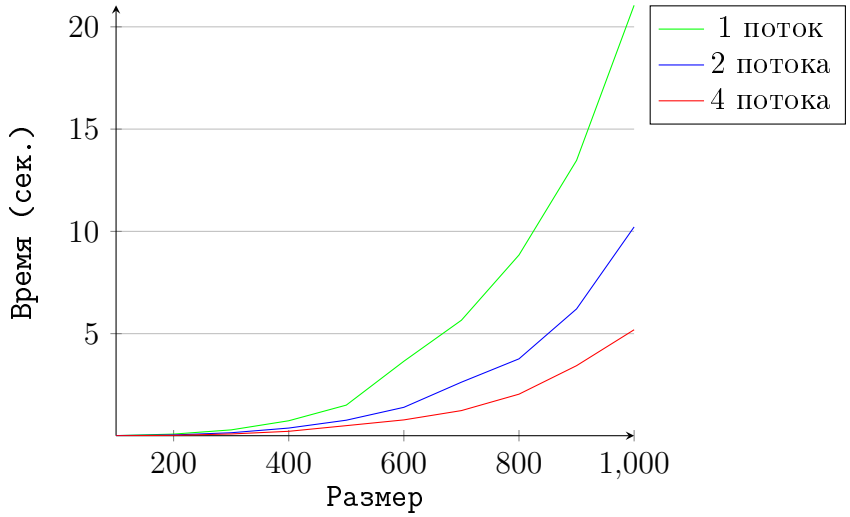

Рисунок 4. График времен работы алгоритма при 1, 2, 4 потоках.

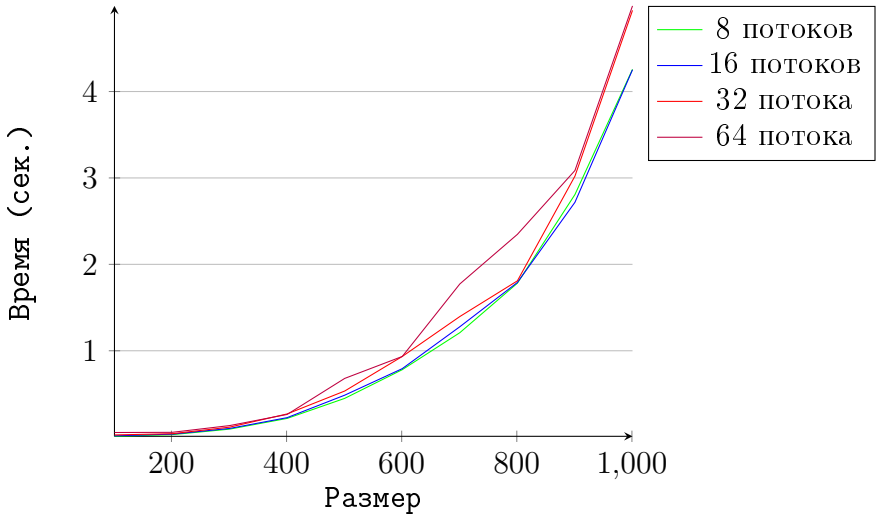

Рисунок 5. График времен работы алгоритма при 8, 16, 32, 64 потоках.

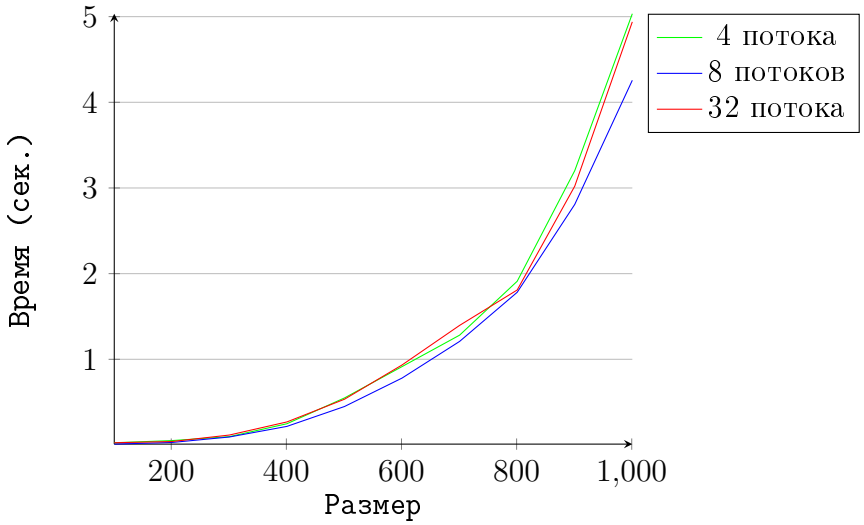

Рисунок 6. График времен работы алгоритма при 4, 8, 32 потоках.

Вывод по данному случаю:

Из Рисунка 5. и Рисунка 6. видно, что при восьми потоках время работы данного алгоритма наименьшее.

Время работы алгоритма при восьми потоках на матрицах размера 1001x1001 составляет 4.45 с, при 800x800 - 1.91 с.

Вывод

В данной главе проведет эксперимент по измерению времени работы рассматриваемого алгоритма в зависимости от количества потоков. Рассматривались матрицы четного и нечетного размера.

В результате сделан вывод, что для матриц четного размера, также как и для матриц нечетного размера, наилучшая производительность достигается при количестве потоков равном восьми. При таком количестве потоков время умножения матриц 800x800 равно 1.8 с, 801x801 - 1.91 с, 1000x1000 - 4.32с, 1001x1001 - 4.45 с.

При работе с одним потоком время умножения матриц 801x801 в 5 раз больше, матриц 800x800 в 7 раз больше, 1001x1001 в 4.74 раза больше, 1000x1000 в 4.73 раза больше.

Заключение

В ходе выполнения данной лабораторной был работы реализован оптимизированный, при помощи распараллеливания вычислений, алгоритм Винограда.

Также проведено исследование поведения алгоритма при различном количестве потоков, на основании проведенного эксперимента. В этом эксперементе матрицы 100x100, 200x200, ... , 1000x1000 и 101x101, 201x201, ... , 1001x1001 перемножались сами с собой, при этом использовалось разное количество потоков (1, 2, 4, .. , 64).

В результате этого исследования сделан вывод, что наилудшая производительность достигается при восьми потоках. При таком количестве потоков получилось ускорить время работы, по сравнению с одним потоком, в 5 раз для 801x801, в 7 раз для 800x800, в 4.74 раза для 1001x1001 и в 4.73 раза для 1000x1000.

Литература

[1] Умножение матриц. URL: http://www.algolib.narod.ru/Math/Matrix.html.

[2] std::thread. URL: https://ru.cppreference.com/w/cpp/thread/thread.

[3] <ctime> (time.h). URL: http://www.cplusplus.com/reference/ctime/.

[4] Стандарт языка C++11 согласно ISO. URL: https://isocpp.org/std/the-standard.