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»

Отчёт

по лабораторной работе № 4

Название: Параллельное умножение матриц

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б _____ Д.О. Склифасовский

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподователь _____ Л.Л. Волкова

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

Введение **4**

1 Аналитический раздел **5**

1.1 Общие сведения 5

1.2 Алгоритм Винограда 5

1.3 Параллельное программирование 6

1.4 Параллельный алгоритм Винограда 6

1.5 Вывод 7

2 Конструкторский раздел **8**

2.1 Разработка алгоритма 8

2.2 Распараллеливание программы 9

2.3 Вывод 9

3 Технологический раздел **10**

3.1 Общие требования 10

3.2 Средства реализации 10

3.3 Сведения о модулях программы 10

3.4 Листинг кода программы 11

3.5 Вывод 16

4 Экспериментальный раздел **17**

4.1 Описание экспериментов 17

4.2 Технические характеристики устройства, на котором проводились замеры 17

4.3 Примеры работы программы 18

4.4 Анализ времени работы алгоритмов 20

4.5 Вывод 22

5 Заключение **23**

Литература **24**

Введение

Цель работы: изучение возможности параллельных вычислений и использование такого подхода на практике.

В ходе лабораторной работы требуется:

1) выбрать алгоритм для рассмотрения;
2) описать обычную версию;
3) выполнить 2 параллельные версии;
4) запустить эксперименты на каждый при различном числе потоков 1,2,4,8 ...4M, где M - количество логических ядер на компьютере;
5) проверить, всегда ли при росте потоков, время работы снижается;
6) описать главный и рабочий поток схемой;
7) описать точки сборки.

В данной лабораторной работе был выбран алгоритм Винограда. Необходимо сравнить зависимость времени работы алгоритма от числа параллельных потоков и размера матриц, провести сравнение стандартного и параллельного алгоритмов.

1 Аналитический раздел

В данном разделе представлено описание стандартного и параллельного алгоритмов Винограда.

1.1 Общие сведения

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы. Хотя исторически рассматривались, например, треугольные матрицы, в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц.

1.2 Алгоритм Винограда

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее. [1]

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно:

$$V * W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4 \tag{1}$$

Это равенство можно переписать в виде:

$$V*W = (v_1+w_2)(v_2+w_1)+(v_3+w_4)(v_4+w_3)-v_1v_2-v_3v_4-w_1w_2-w_3w_4 \quad (2)$$

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. На практике это означает, что над предварительно обработанными элементами придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.3 Параллельное программирование

Параллельное программирование. Параллельное программирование служит для создания программ, эффективно использующих вычислительные ресурсы за счет одновременного исполнения кода на нескольких вычислительных узлах. Для создания параллельных приложений используются параллельные языки программирования и специализированные системы поддержки параллельного программирования, такие как MPI и OpenMP. Параллельное программирование является более сложным по сравнению с последовательным как в написании кода, так и в его отладки. Для облегчения процесса параллельного программирования существуют специализированные инструменты, например, отладчик TotalView.[2]

1.4 Параллельный алгоритм Винограда

Для того, чтобы добиться большей эффективности алгоритма умножения матриц Винограда, необходимо расспараллелить ту часть алгоритма, которая содержит больше всего вложенных циклов. Вычисление результата для каждой строки не зависит от результата умножения для других строк. Каждый поток будет выполнять вычисления определенных строк результирующей матрицы.

Для проверки эффективности выбранного метода, также можно распараллелить циклы двойной вложенности в начале алгоритма (циклы вычисления векторов mulH и mulV).

1.5 Вывод

Было представлено математическое описание стандартного и параллельного алгоритмов Винограда. Основная идея заключается в том, чтобы распараллелить главный цикл тройной вложенности.

2 Конструкторский раздел

В данном разделе представлены схемы разработанных алгоритмов. Также описывается часть алгоритма, которая будет распараллеливаться.

2.1 Разработка алгоритма

На рисунке 1 изображена схема стандартного алгоритма умножения матриц.

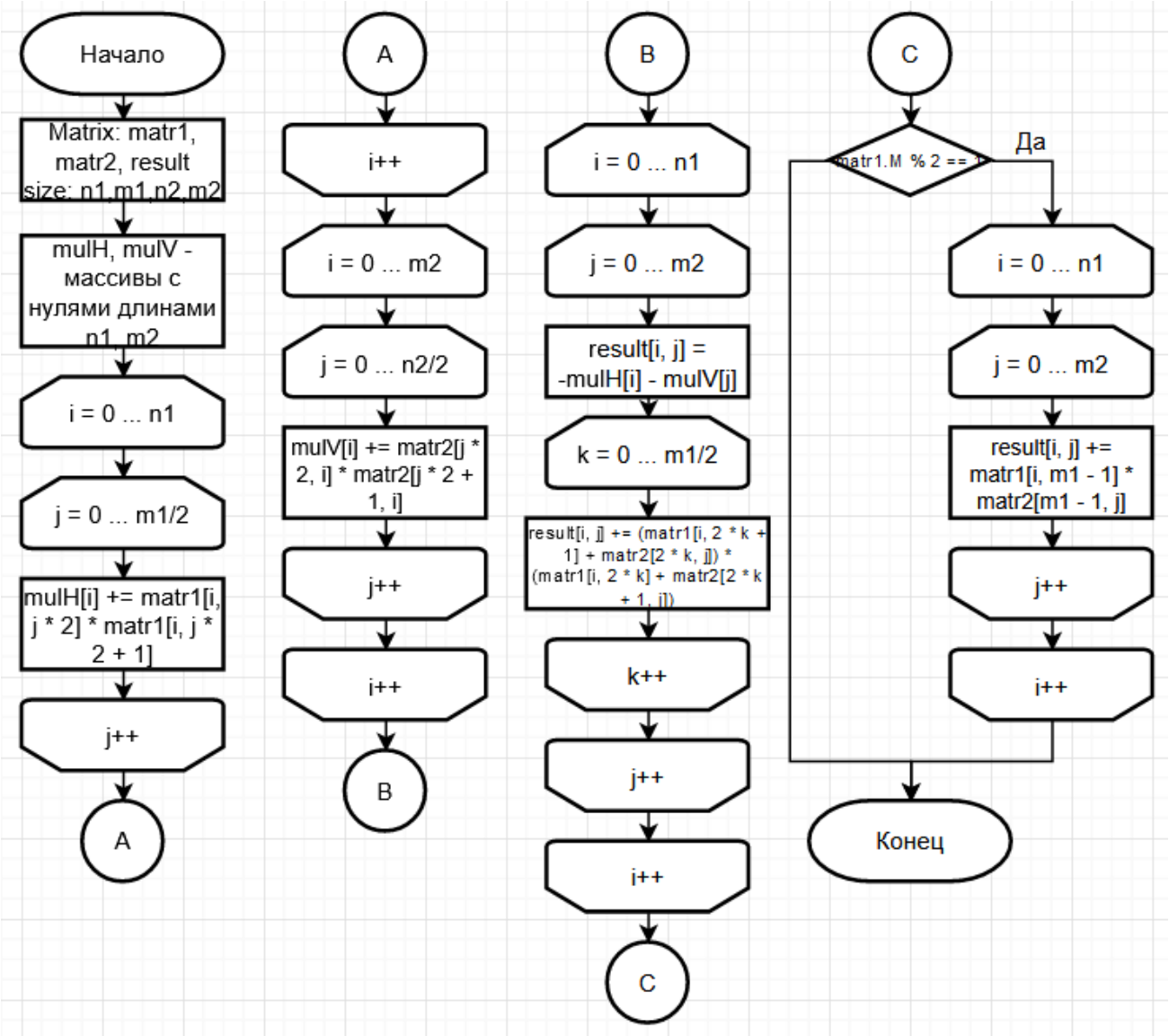

Рисунок 1 – Схема стандартного алгоритма умножения матриц

2.2 Распараллеливание программы

В представленном алгоритме будет распараллеливаться цикл тройной вложенности (участок между B и C). Это должно ускорить время работы алгоритма.

Также для проверки эффективности будут распараллеливаться два первых цикла отдельно. Будут построены графики для данной реализации и сравниться время работы алгоритмов.

2.3 Вывод

В данном разделе была рассмотрена схема алгоритма Винограда и описалась часть алгоритма, которая будет распараллеливаться.

3 Технологический раздел

В данном разделе даны общие требования к программе, средства реализации и сама реализация алгоритмов.

3.1 Общие требования

Требования к вводу:

1) вводятся размеры матриц;

2) вводятся (или автоматически генерируются) матрицы.

Требования к программе

1) при вводе неправильных размеров матриц программа не должна завершиться аварийно;

2) должно выполняться корректное умножение матриц.

3.2 Средства реализации

В качестве языка программирования был выбран C#, так как я знаком с данным языком программирования, имею представление о способах тестирования программы.[3]

Средой разработки Visual Studio.[4]

Для замеров процессорного времени используется функция *Stopwatch*.[5][6]

Многопоточное программирование было реализовано с помощью пространства имен System.Threading.[7]

3.3 Сведения о модулях программы

Программа состоит из:

1) Program.cs - главный файл программы, в котором располагается точка входа в программу;

2) Matrix.cs - файл класса Matrix. Класс реализует матрицу размером $n * m$, а также он содержит методы для работы с матрицами;

3) Parameters.cs - файл классов ParametersForFirst и ParametersForSecond, которые используются для передачи данных функциям при распараллеливании циклов;

4) MultMatr.cs - файл класса MultMatr. В нем находится алгоритм Винограда;

5) ParallelMultMatr.cs - файл класса ParallelMultMatr, в котором находятся параллельныи реализации алгоритма Винограда.

3.4 Листинг кода программы

В листинге 1 реализован стандартный алгоритм Винограда.

Листинг 1 – Стандартный алгоритм Винограда

```
1   public static Matrix VinogradMult(Matrix matr1, Matrix matr2)
2   {
3       int n1 = matr1.N;
4       int n2 = matr2.N;
5       int m1 = matr1.M;
6       int m2 = matr2.M;
7       if ((m1 != n2) || n1 == 0 || n2 == 0)
8       {
9           return null;
10      }
11
12      Matrix result = new Matrix(n1, m2);
13      int[] mulH = new int[n1];
14      int[] mulV = new int[m2];
15
16      for (int i = 0; i < n1; i++)
17      {
```

```
18              for (int j = 0; j < m1 / 2; j++)
19              {
20                  mulH[i] += matr1[i, j * 2] * matr1[i, j * 2 + 1];
21              }
22          }
23
24          for (int i = 0; i < m2; i++)
25          {
26              for (int j = 0; j < n2 / 2; j++)
27              {
28                  mulV[i] += matr2[j * 2, i] * matr2[j * 2 + 1, i];
29              }
30          }
31
32          for (int i = 0; i < n1; i++)
33          {
34              for (int j = 0; j < m2; j++)
35              {
36                  result[i, j] = -mulH[i] - mulV[j];
37                  for (int k = 0; k < m1 / 2; k++)
38                  {
39                      result[i, j] += (matr1[i, 2 * k + 1] + matr2[2
                            * k, j]) * (matr1[i, 2 * k] + matr2[2 * k
                          + 1, j]);
40                  }
41              }
42          }
43
44          if (m1 % 2 == 1)
45          {
46              for (int i = 0; i < n1; i++)
47              {
48                  for (int j = 0; j < m2; j++)
49                  {
50                      result[i, j] += matr1[i, m1 - 1] * matr2[m1 -
```

```
                        1, j];
51                  }
52              }
53          }
54
55          return result;
56      }
```

В листинге 2 реализован параллельный алгоритм Винограда

Листинг 2 – Параллельный алгоритм Винограда

```
1   public static Matrix ParallelVinogradMult1(Matrix matr1,
        Matrix matr2, int countOfThreads)
2   {
3       int n1 = matr1.N;
4       int n2 = matr2.N;
5       int m1 = matr1.M;
6       int m2 = matr2.M;
7       if ((m1 != n2) || n1 == 0 || n2 == 0)
8       {
9           return null;
10      }
11
12      Matrix result = new Matrix(n1, m2);
13      int[] mulH = new int[n1];
14      int[] mulV = new int[m2];
15
16      for (int i = 0; i < n1; i++)
17      {
18          for (int j = 0; j < m1 / 2; j++)
19          {
20              mulH[i] += matr1[i, j * 2] * matr1[i, j * 2 + 1];
21          }
22      }
23
```

```
24          for (int i = 0; i < m2; i++)
25          {
26              for (int j = 0; j < n2 / 2; j++)
27              {
28                  mulV[i] += matr2[j * 2, i] * matr2[j * 2 + 1, i];
29              }
30          }
31
32          Thread[] threads = new Thread[countOfThreads];
33          int distribution = n1 / countOfThreads;
34          int startI = 0;
35          for (int i = 0; i < countOfThreads; i++)
36          {
37              int endI = startI + distribution;
38              if (i == countOfThreads - 1)
39              {
40                  endI = n1;
41              }
42              ParametersForFirst param = new ParametersForFirst(
                    result, matr1, matr2, mulV, mulH, startI, endI, m2,
                     m1);
43              threads[i] = new Thread(ComputationLoopForMain);
44              threads[i].Start(param);
45              startI = endI;
46          }
47          foreach (Thread curThread in threads)
48          {
49              curThread.Join();
50          }
51
52          if (m1 % 2 == 1)
53          {
54              for (int i = 0; i < n1; i++)
55              {
56                  for (int j = 0; j < m2; j++)
```

```
57                  {
58                      result[i, j] += matr1[i, m1 - 1] * matr2[m1 -
                            1, j];
59                  }
60              }
61          }
62
63          return result;
64      }
```

В листинге 3 реализован алгоритм Винограда с распараллеленым алгритмом Винограда

Листинг 3 – Распараллеленный главный цикл

```
1       private static void ComputationLoopForMain (object curObj)
2       {
3           ParametersForFirst param = (ParametersForFirst)curObj;
4           int startI  = param.startI,
5               endI    = param.endI,
6               m2      = param.m2,
7               m1      = param.m1;
8           Matrix res  = param.res,
9               matr1   = param.matr1,
10              matr2   = param.matr2;
11          int[] mulH  = param.mulH,
12              mulV    = param.mulV;
13
14          for (int i = startI; i < endI; i++)
15          {
16              for (int j = 0; j < m2; j++)
17              {
18                  res[i, j] = -mulH[i] - mulV[j];
19                  for (int k = 0; k < m1 / 2; k++)
20                  {
21                      res[i, j] += (matr1[i, 2 * k + 1] + matr2[2 *
```

```
                        k, j]) * (matr1[i, 2 * k] + matr2[2 * k +
                        1, j]);
22                  }
23              }
24          }
25      }
```

3.5 Вывод

В данном разделе были даны общие требования к программе, описаны средства реализации, были представлены сведения о модулях программы, а также реализованы алгоритмы умножения матриц (Винограда и параллельный Винограда).

4 Экспериментальный раздел

В данном разделе представлены результаты работы программы и приведен анализ времени работы алгоритмов.

4.1 Описание экспериментов

Было произведено 2 серии замеров:

1) для матриц размером $NxN = 300$ при переменном числе потоков для параллельных реализаций;

2) для матриц размером $NxN, N \in \{100, 200, 300, 400, 500, 600, 700\}$ с постоянным числом потоков для параллельных реализаций.

4.2 Технические характеристики устройства, на котором проводились замеры

1) операционная система Windows 10 64-bit;

2) память 8 ГБ;

3) процессор AMD Ryzen 5 3500U CPU @ 2.1GHz;

4) 8 логических процессоров.

4.3 Примеры работы программы

На рисунке 2 представлен результат работы алгоритмов.

```
0 - Выход
1 - Проверить умножение матриц
2 - Тестировка
Ввод: 1
Введите первое n:
4
Введите первое m:
4
Введите второе n:
4
Введите второе m:
4
Первая матрица:
6       82      56      34
46      4       28      78
81      12      42      38
8       20      95      17
Вторая матрица:
61      66      0       60
51      42      49      56
97      69      42      52
55      44      89      69
Результат 1:
11850   9200    9396    10210
10016   8568    8314    9822
11717   10420   5734    10338
11658   8671    6483    7713
Результат 2:
11850   9200    9396    10210
10016   8568    8314    9822
11717   10420   5734    10338
11658   8671    6483    7713
Результат 3:
11850   9200    9396    10210
10016   8568    8314    9822
11717   10420   5734    10338
11658   8671    6483    7713
```

Рисунок 2 – Первый результат работы программы

На рисунке 3 представлен второй результат работы алгоритмов.

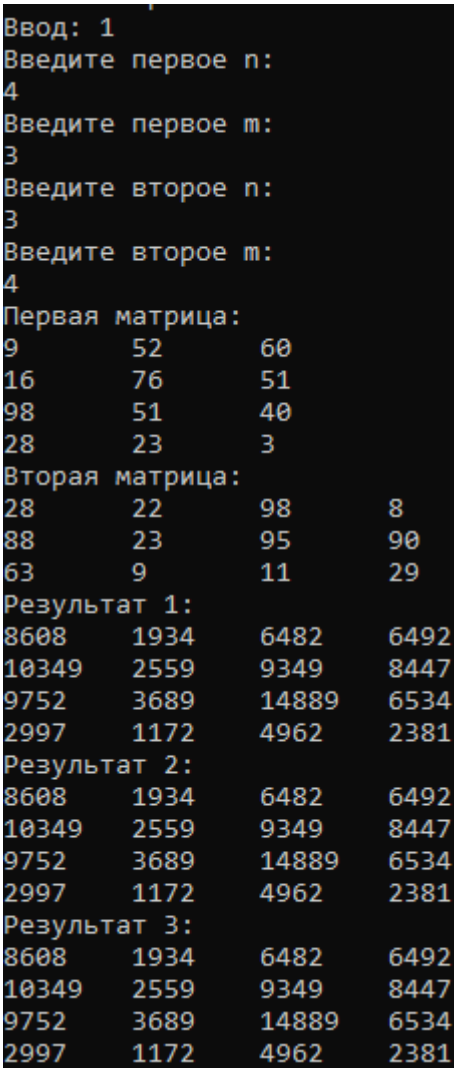

Рисунок 3 – Второй результат работы программы

4.4 Анализ времени работы алгоритмов

На рисунке 4 представлен график зависимости времени работы алгоритмов и количества потоков.

Для передачи в распараллеленную функцию данных в C# необходимо передавать данные только в объекте, изначально создается объект, который хранит необходимые данные, и в функцие данные переприсваиваются переменным. Из-за этого время работы 2 параллельной реализации будет намного больше линейной реализации и 1 параллельной реализации.

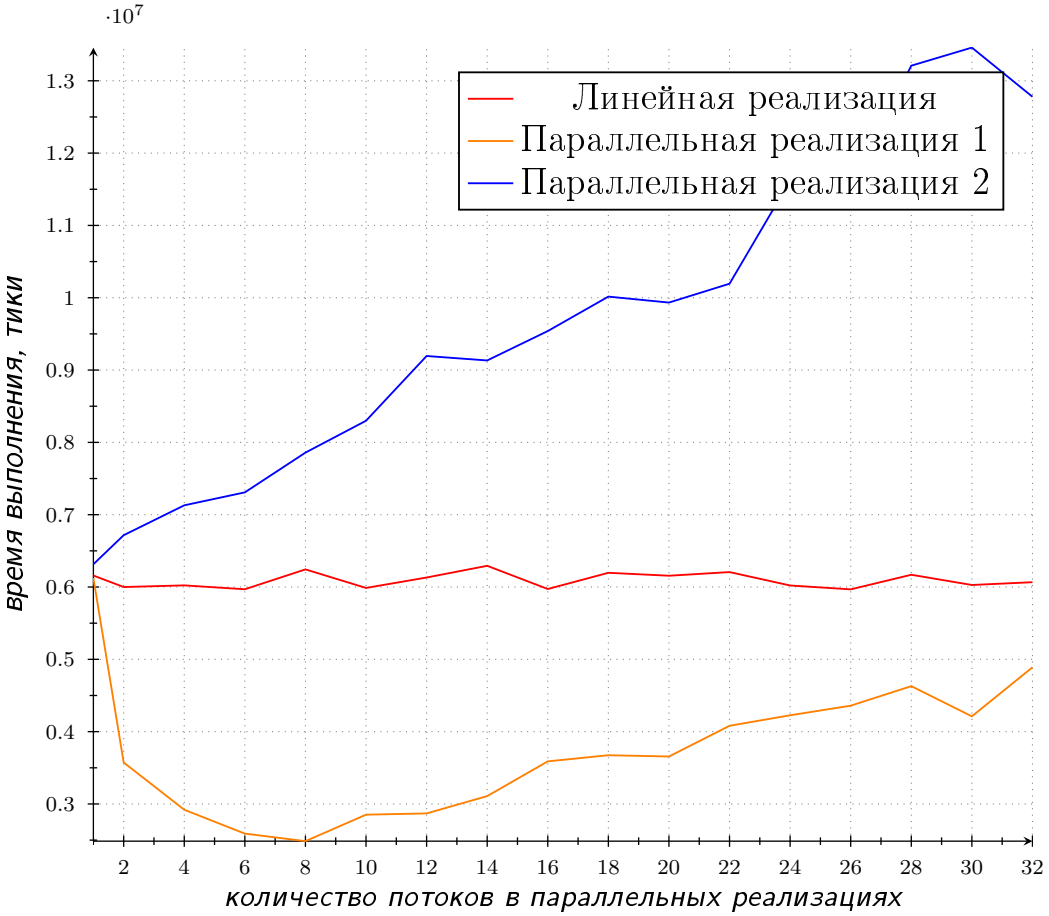

Рисунок 4 – Результаты замеров процессорного времени в первом эксперименте.

На рисунке 5 представлен график зависимости времени работы алгоритмов и размера матриц.

Из-за того, что замеры проводятся с 8 логическими процессорами - эффективнее всего использовать 8 потоков.

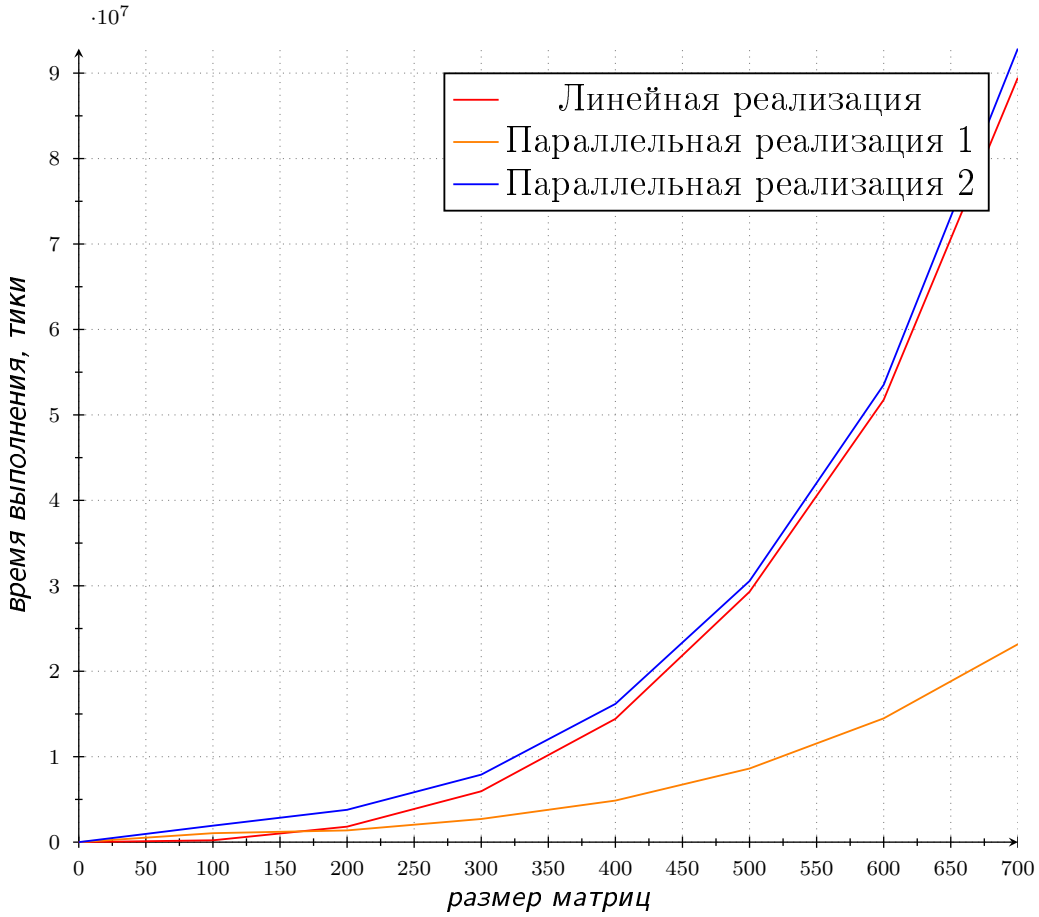

Рисунок 5 – Результаты замеров процессорного времени во втором эксперименте.

4.5 Вывод

Результаты тестирования показывают, что, во-первых, с увеличением количества потоков увеличивается время работы параллельной реализации 2. Линенйая реализация показывает одинаковый результат. А параллельная реализация 2 показывает, что при 8 потоках - она наиболее эффективна, а с увеличением увеличивается время.

Во-вторых, при одинаковом количестве потоков и разных размерах матриц, самым эффективной является вторая параллельная реализация, а первая и линейная работают идентично.

Можно сделать вывод, что самой эффективной реализацией является вторая и, что следует распараллеливать главный цикл (состоящий из 3 вложенных циклов).

5 Заключение

В ходе выполнения лабораторной работы были изучены возможности параллельных вычислений и был использован такой подход на практике. Были даны теоритически оценки алгоритмов умножения матриц. Также сравнили время работы алгоритмов, в результате которого стало понятно, что самой эффективной реализацией была та, когда было необходимо распараллелить главный цикл алгоритма Винограда. Данная реализация быстрее линейной примерно в 2.5 раза.

Литература

1. Умножение матриц. -URL: http://www.algolib.narod.ru/Math/Matrix.html (дата обращения: 24.10.2020). -Текст: электронный.

2. Параллельное программирование. -URL: https://www.viva64.com/ru/t/0038/ (дата обращения: 24.10.2020). -Текст: электронный.

3. Документация по C#. -URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 24.10.2020). -Текст: электронный.

4. Документация по семейству продуктов Visual Studio. -URL: https://docs.microsoft.com/ru-ru/visualstudio/?view=vs-2019 (дата обращения: 01.10.2020). -Текст: электронный.

5. Stopwatch Класс. -URL: https://goo.su/2e99 (дата обращения: 24.10.2020). -Текст: электронный.

6. Под капотом у Stopwatch. -URL: https://habr.com/ru/post/226279/ (дата обращения: 24.10.2020). Текст: электронный.

7. Thread Класс. -URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.thre 3.1 (дата обращения 24.10.2020). -Текст: электронный.