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: Многопоточность

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б _____ Хетагуров П.К

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

Преподователь _____ Л.Л. Волкова

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

Введение

В данной лабораторной работе будут рассмотренны и проанализированы параллельные реализации алгоритма Винограда. Проведено сравнение рассмотренных параллельных алгоритмов с последовательной версией алгоритма Винограда.

1 Аналитическая часть

В данном разделе будут поставлены цели и задачи работы, будут рассмотренны основные теоритические сведения связанные с алгоритмами сортировки.

1.1 Цель и задачи работы

Цель работы:

Научиться работать с параллельными вычислениями.

Задачи работы:

1. разработать две параллельные версии алгоритма Винограда;
2. реализовать разработанные алгоритмы и последовательный алгоритм Винограда;
3. провести эксперименты по замеру времени работы разработанных алгоритмов;
4. провести сравнения алгоритмов по затраченному времени.

1.2 Классическое умножение матриц

Операция умножения двух матриц выполнима тогда и только тогда, когда число столбцов в первом сомножителе равно числу строк во втором.

Произведением матрицы A[m×n] на матрицу B[n×k] называется матрица C[m×k] такая, что элемент матрицы C, стоящий в i-ой строке и j-ом столбце, т. е. элемент $c_{i,j}$, равен сумме произведений элементов i-ой строки матрицы A на соответствующие элементы j-ого столбца матрицы B. Т.е определяется формулой (1):

$$c_{i,j} = \sum_{r=1}^{m} a_{ir} b_{rj} \qquad (i = 1, 2, ...l; j = 1, 2, ...n) \tag{1}$$

1.3 Последовательный алгоритм Винограда

Видно, что каждый элемент в результате умножения матриц представляет собой скалярное произведение соответствующих строки и столбца матриц. В алгоритме Винограда происходит некоторая предварительная обработка, позволяющая вычислить часть данных заранее. Заметим, что скалярное произведение двух векторов V и W, например, размерностью 4, можно переписать как (2):

$$V * W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4 \tag{2}$$

В случае матрицы со стороной нечетной длины, необходимо прибавить сделать коррекцию: $result[i, j] += first[i, m1 - 1] * second[m1 - 1, j]$

Видно, что выражение в правой части допускает предварительную обработку, его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй [1].

1.4 Параллельные реализации алгоритма Винограда

Алгоритм Винограда можно разделить на 4 части:

1. подготовка вектора mulH;
2. подготовка вектора mulV;
3. основной цикл;
4. цикл, вносящий поправки в вычисление.

Следовательно, параллельные реализации могут быть следующими:

1. распараллеливание каждой из частей алгоритма, причем первые две могут выполняться одновременно;
2. распараллеливание только основного цикла, так как он занимает наибольшую часть времени выполнения.

1.5 Вывод

В данной части были поставлены задачи и цель работы, рассмотрен классический алгоритм умножения матриц, алгоритм Винограда и способы его распараллеливания.

2 Конструкторская часть

В данном разделе будут рассмотренны схемы алгоритмов, требования к функциональности ПО.

2.1 Требования к ПО

ПО должно иметь два режима работы, выбираемых из меню

1. Режим демонстрации. В этом режиме должен осуществляться ввод матрицы и демонстрация работы на ней всех реализованных алгоритмов.
2. Режим тестирования. В этом режиме должны проводится замеры времени выполнения реализованных алгоритмов в зависимости от количества потоков и размера матриц. Должен осуществляться вывод затраченного процессорного времени на случайным образом сгенерированных данных.

2.2 Схема алгоритма Винограда

На рисунке 1 изображена схема алгоритма Винограда.

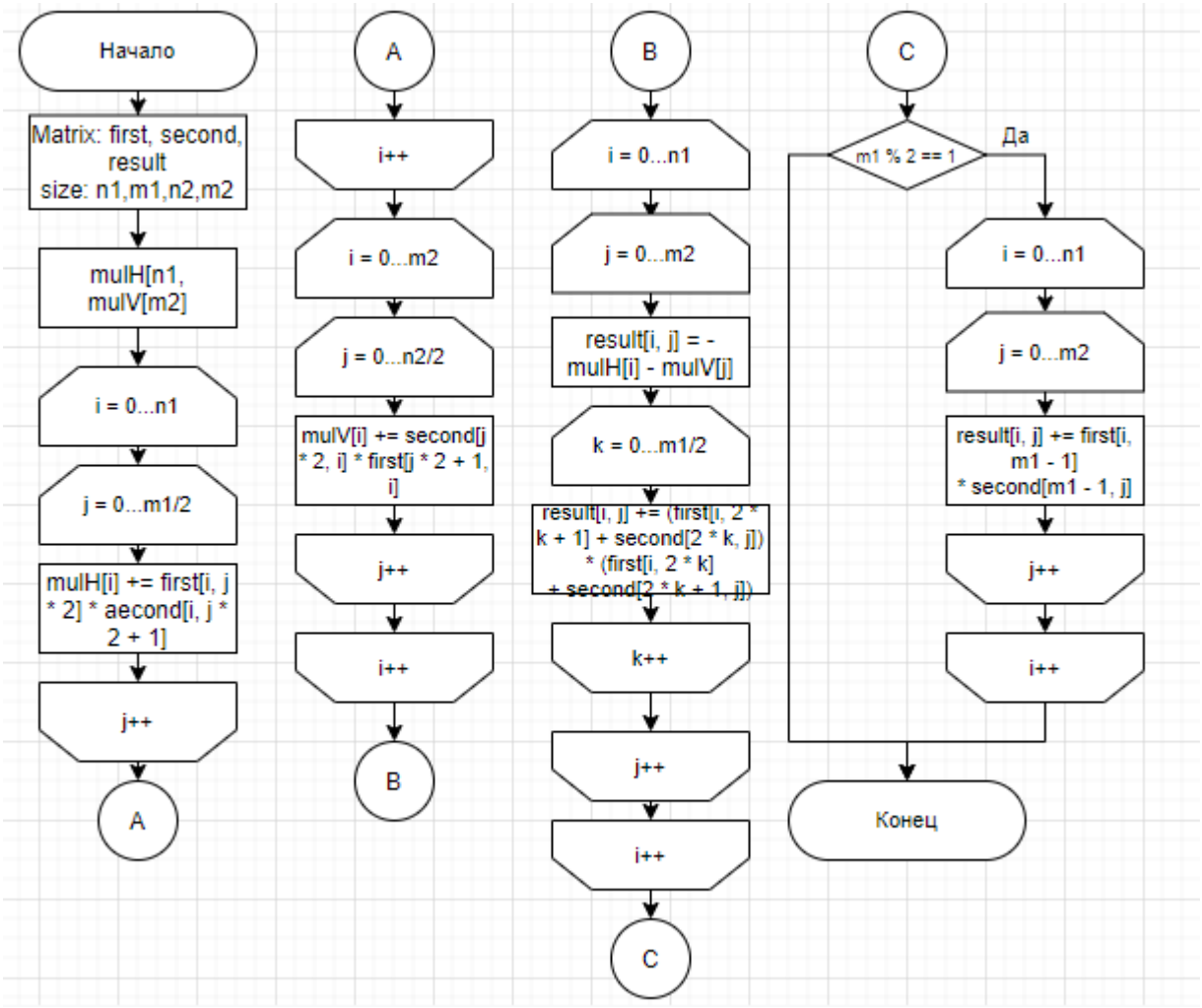

Рисунок 1 – Схема алгоритма Винограда

2.3 Параллельные модификации

В первой параллельной реализации будут распараллелены все части алгоритма, а так же части A и B(расчет массивов) будут выполняться параллельно. Расспаралеливание частей A, B, C, D будет заключаться в делегировании процессу обработки части массива или матрицы, распределения зон ответственности. На рисунке 2 изображена схема данной параллельной реализации.

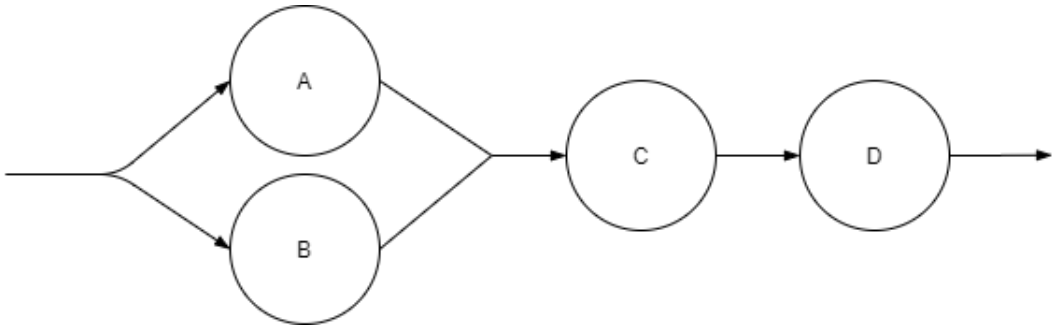

Рисунок 2 – Первый вариант распараллеливания

Во второй параллельной реализации будет распараллелена только часть C(главный цикл). На рисунке 3 изображена схема второй параллельной реализации.

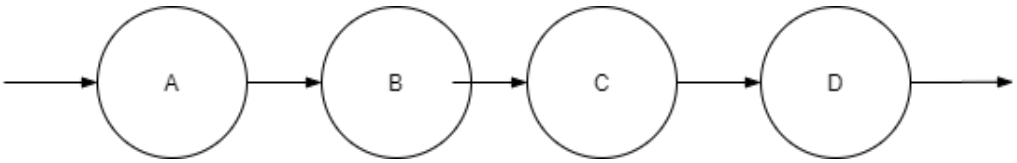

Рисунок 3 – Второй вариант распараллеливания

2.4 Вывод

В данном разделе были рассмотрены схемы алгоритмов и обозначены требования к ПО.

3 Технологическая часть

Ниже будут представлены средства реализации и листинги реализованной программы.

3.1 Средства реализации

Выбранный язык программирования - C#, так как требований по конкретнему языку не выдвигалось и он предоставляет удобные средства распараллеливания [2]. Среда разработки - Visual Studio [3].

Технические характеристики машины, на которой проводились тесты:

- Windows 10 x64;
- 8 ГБ оперативной памяти;
- CPU: AMD FX(tm)-6350 Six-Core Processor 3.90GHz;
- 6 логических ядер.

3.2 Реализации алгоритмов

Ниже представлены листинги реализаций алгоритмов. На листинге 1 представлен алгоритм Винограда.

Листинг 1 – Алгоритм Винограда

```
1  public static Matrix Classic(Matrix first, Matrix second)
2        {
3            Matrix result = new Matrix(0, 0);
4            if (first.M == second.N && first.N != 0 && second.M != 0)
5            {
6                int n1 = first.N;
7                int m1 = first.M;
8                int n2 = second.N;
9                int m2 = second.M;
10
11               result = new Matrix(n1, m2);
12               int[] mulH = new int[n1];
13               int[] mulV = new int[m2];
14
15               for (int i = 0; i < n1; i++)
16               {
17                   for (int j = 0; j < m1 / 2; j++)
18                   {
19                       mulH[i] += first[i, j * 2] * first[i, j * 2 + 1];
20                   }
```

```
                }

                for (int i = 0; i < m2; i++)
                {
                    for (int j = 0; j < n2 / 2; j++)
                    {
                        mulV[i] += second[j * 2, i] * second[j * 2 + 1, i];
                    }
                }

                for (int i = 0; i < n1; i++)
                {
                    for (int j = 0; j < m2; j++)
                    {
                        result[i, j] = -mulH[i] - mulV[j];
                        for (int k = 0; k < m1 / 2; k++)
                        {
                            result[i, j] += (first[i, 2 * k + 1] + second[2 * k, j]) *
                                (first[i, 2 * k] + second[2 * k + 1, j]);
                        }
                    }
                }

                if (m1 % 2 == 1)
                {
                    for (int i = 0; i < n1; i++)
                    {
                        for (int j = 0; j < m2; j++)
                        {
                            result[i, j] += first[i, m1 - 1] * second[m1 - 1, j];
                        }
                    }
                }
            }

            return result;
        }
```

На листинге 2 представлен первый вариант распараллеливания алгоритм Винограда.

Листинг 2 – Первый вариант распараллеливания алгоритм Винограда

```
public static Matrix ParallelFirst(Matrix first, Matrix second, int threadsCount)
```

```
2         {
3             Matrix result = new Matrix(0, 0);
4             if (threadsCount <= 1)
5             {
6                 result = Classic(first, second);
7             }
8             else if (first.M == second.N && first.N != 0 && second.M != 0)
9             {
10                int n1 = first.N;
11                int m1 = first.M;
12                int n2 = second.N;
13                int m2 = second.M;
14
15                result = new Matrix(n1, m2);
16                int[] mulH = new int[n1];
17                int[] mulV = new int[m2];
18
19                Thread[] threads = new Thread[threadsCount];
20
21                int threadsHalfing = threadsCount / 2;
22
23                // mulH
24                int threadWork = threadsHalfing;
25                int step = n1 / threadWork;
26                if (threadWork / n1 >= 1)
27                {
28                    step = 1;
29                    threadWork = n1;
30                    threadsHalfing = threadWork;
31                }
32                int start = 0;
33                for (int i = 0; i < threadWork - 1; i++)
34                {
35                    threads[i] = new Thread(CountMulH);
36                    threads[i].Start(new ParametersForMul(first, mulH, start, start +
                          step, m1));
37                    start += step;
38                }
39                threads[threadWork - 1] = new Thread(CountMulH);
40                threads[threadWork - 1].Start(new ParametersForMul(first, mulH, start,
                      n1, m1));
```

```
41
42                  // MulV
43                  threadWork = (threadsCount - threadsHalfing);
44                  step = m2 / threadWork;
45                  if (threadWork / m2 >= 1)
46                  {
47                      step = 1;
48                      threadWork = m2;
49                  }
50                  start = 0;
51                  for (int i = threadsHalfing; i < threadsHalfing + threadWork - 1; i++)
52                  {
53                      threads[i] = new Thread(CountMulV);
54                      threads[i].Start(new ParametersForMul(second, mulV, start, start +
                            step, n2));
55                      start += step;
56                  }
57                  threads[threadsHalfing + threadWork - 1] = new Thread(CountMulV);
58                  threads[threadsHalfing + threadWork - 1].Start(new ParametersForMul(
                        second, mulV, start, m2, n2));
59
60                  //sync
61                  for (int i = 0; i < threadWork + threadsHalfing; i++)
62                  {
63                      threads[i].Join();
64                  }
65
66                  //Main
67                  step = n1 / threadsCount;
68                  threadWork = threadsCount;
69                  if (threadsCount / n1 >= 1)
70                  {
71                      step = 1;
72                      threadWork = n1;
73                  }
74                  start = 0;
75                  for (int i = 0; i < threadWork - 1; i++)
76                  {
77                      threads[i] = new Thread(CountMain);
78                      threads[i].Start(new ParametersForMain(result, first, second, mulV,
                            mulH, start, start + step, m2, m1));
```

```
79                      start += step;
80                  }
81                  threads[threadWork - 1] = new Thread(CountMain);
82                  threads[threadWork - 1].Start(new ParametersForMain(result, first,
                        second, mulV, mulH, start, n1, m2, m1));
83
84                  // sync
85                  for (int i = 0; i < threadWork; i++)
86                  {
87                      threads[i].Join();
88                  }
89
90                  // end
91                  if (m1 % 2 == 1)
92                  {
93                      start = 0;
94                      for (int i = 0; i < threadWork - 1; i++)
95                      {
96                          threads[i] = new Thread(CountTail);
97                          threads[i].Start(new ParametersForMain(result, first, second,
                                mulV, mulH, start, start + step, m2, m1));
98                          start += step;
99                      }
100                     threads[threadWork - 1] = new Thread(CountTail);
101                     threads[threadWork - 1].Start(new ParametersForMain(result, first,
                            second, mulV, mulH, start, n1, m2, m1));
102
103                     // sync
104                     for (int i = 0; i < threadWork; i++)
105                     {
106                         threads[i].Join();
107                     }
108                 }
109             }
110
111             return result;
112         }
```

На листинге 3 представлен второй вариант распараллеливания алгоритм Винограда.

Листинг 3 – Второй вариант распараллеливания алгоритм Винограда

```
1 public static Matrix ParallelSecond(Matrix first, Matrix second, int threadsCount)
```

```
2         {
3             Matrix result = new Matrix(0, 0);
4             if (threadsCount <= 1)
5             {
6                 result = Classic(first, second);
7             }
8             else if (first.M == second.N && first.N != 0 && second.M != 0)
9             {
10                int n1 = first.N;
11                int m1 = first.M;
12                int n2 = second.N;
13                int m2 = second.M;
14
15                result = new Matrix(n1, m2);
16                int[] mulH = new int[n1];
17                int[] mulV = new int[m2];
18
19                for (int i = 0; i < n1; i++)
20                {
21                    for (int j = 0; j < m1 / 2; j++)
22                    {
23                        mulH[i] += first[i, j * 2] * first[i, j * 2 + 1];
24                    }
25                }
26
27                for (int i = 0; i < m2; i++)
28                {
29                    for (int j = 0; j < n2 / 2; j++)
30                    {
31                        mulV[i] += second[j * 2, i] * second[j * 2 + 1, i];
32                    }
33                }
34
35                //Main
36                Thread[] threads = new Thread[threadsCount];
37                int step = n1 / threadsCount;
38                int threadWork = threadsCount;
39                if (threadsCount / n1 >= 1)
40                {
41                    step = 1;
42                    threadWork = n1;
```

```
                }
                int start = 0;
                for (int i = 0; i < threadWork - 1; i++)
                {
                    threads[i] = new Thread(CountMain);
                    threads[i].Start(new ParametersForMain(result, first, second, mulV,
                        mulH, start, start + step, m2, m1));
                    start += step;
                }
                threads[threadWork - 1] = new Thread(CountMain);
                threads[threadWork - 1].Start(new ParametersForMain(result, first,
                    second, mulV, mulH, start, n1, m2, m1));

                // sync
                for (int i = 0; i < threadWork; i++)
                {
                    threads[i].Join();
                }

                // end
                if (m1 % 2 == 1)
                {
                    for (int i = 0; i < n1; i++)
                    {
                        for (int j = 0; j < m2; j++)
                        {
                            result[i, j] += first[i, m1 - 1] * second[m1 - 1, j];
                        }
                    }
                }

                return result;
            }
```

На листинге 4 представлены вспомогательные функции.

Листинг 4 – Вспомогательные функции

```
private static void CountMulH(object paramsObj)
        {
            ParametersForMul paramses = (ParametersForMul)paramsObj;
            int start = paramses.start,
```

```
5                 end = paramses.end,
6                 secondBorder = paramses.secondBorder;
7             Matrix matrix = paramses.matrix;
8             int[] mul = paramses.array;
9
10            for (int i = start; i < end; i++)
11            {
12                for (int j = 0; j < secondBorder / 2; j++)
13                {
14                    mul[i] += matrix[i, j * 2] * matrix[i, j * 2 + 1];
15                }
16            }
17        }
18
19        private static void CountMulV(object paramsObj)
20        {
21            ParametersForMul paramses = (ParametersForMul)paramsObj;
22            int start = paramses.start,
23                end = paramses.end,
24                secondBorder = paramses.secondBorder;
25            Matrix matrix = paramses.matrix;
26            int[] mul = paramses.array;
27
28            for (int i = start; i < end; i++)
29            {
30                for (int j = 0; j < secondBorder / 2; j++)
31                {
32                    mul[i] += matrix[j * 2, i] * matrix[j * 2 + 1, i];
33                }
34            }
35        }
36
37        private static void CountMain(object paramsObj)
38        {
39            ParametersForMain paramses = (ParametersForMain)paramsObj;
40            Matrix result = paramses.result,
41                first = paramses.first,
42                second = paramses.second;
43            int[] mulH = paramses.mulH,
44                mulV = paramses.mulV;
45            int start = paramses.start,
```

```
46                 end = paramses.end,
47                 m2 = paramses.m2,
48                 m1 = paramses.m1;
49 
50             for (int i = start; i < end; i++)
51             {
52                 for (int j = 0; j < m2; j++)
53                 {
54                     result[i, j] = -mulH[i] - mulV[j];
55                     for (int k = 0; k < m1 / 2; k++)
56                     {
57                         result[i, j] += (first[i, 2 * k + 1] + second[2 * k, j]) * (
                                first[i, 2 * k] + second[2 * k + 1, j]);
58                     }
59                 }
60             }
61         }
62 
63         private static void CountTail(object paramsObj)
64         {
65             ParametersForMain paramses = (ParametersForMain)paramsObj;
66             Matrix result = paramses.result,
67                 first = paramses.first,
68                 second = paramses.second;
69             int start = paramses.start,
70                 end = paramses.end,
71                 m2 = paramses.m2,
72                 m1 = paramses.m1;
73 
74             for (int i = start; i < end; i++)
75             {
76                 for (int j = 0; j < m2; j++)
77                 {
78                     result[i, j] += first[i, m1 - 1] * second[m1 - 1, j];
79                 }
80             }
81         }
```

3.3 Вывод

В данном разделе были описаны программные и аппаратные средства реализации, были представлены листинги реализаций алгоритмов.

4 Экспериментальная часть

В данной главе будет представлен пример работы программы, результат экспериментов по замеру времени и произведен сравнительный анализ алгоритмов по затрачиваемому времени.

4.1 Пример работы программы

Пример работы программы представлен на рисунке 4

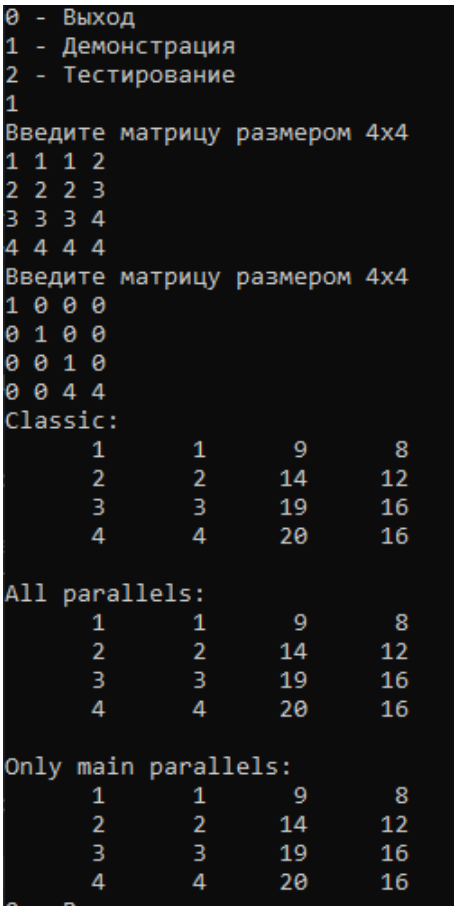

Рисунок 4 – Пример работы программы

4.2 Сравнительный анализ алгоритмов по времени

Эксперимент проводится на матрице размером 500x500. Количество потоков варьируется от 1 до 24 с шагом 2. Количество потоков не влияет на последовательную реализацию алгоритма, поэтому её график не отражает зависимость между скоростью выполнения и количеством потоков, а приведен для сравнения. Результаты представлены на рисунке 5.

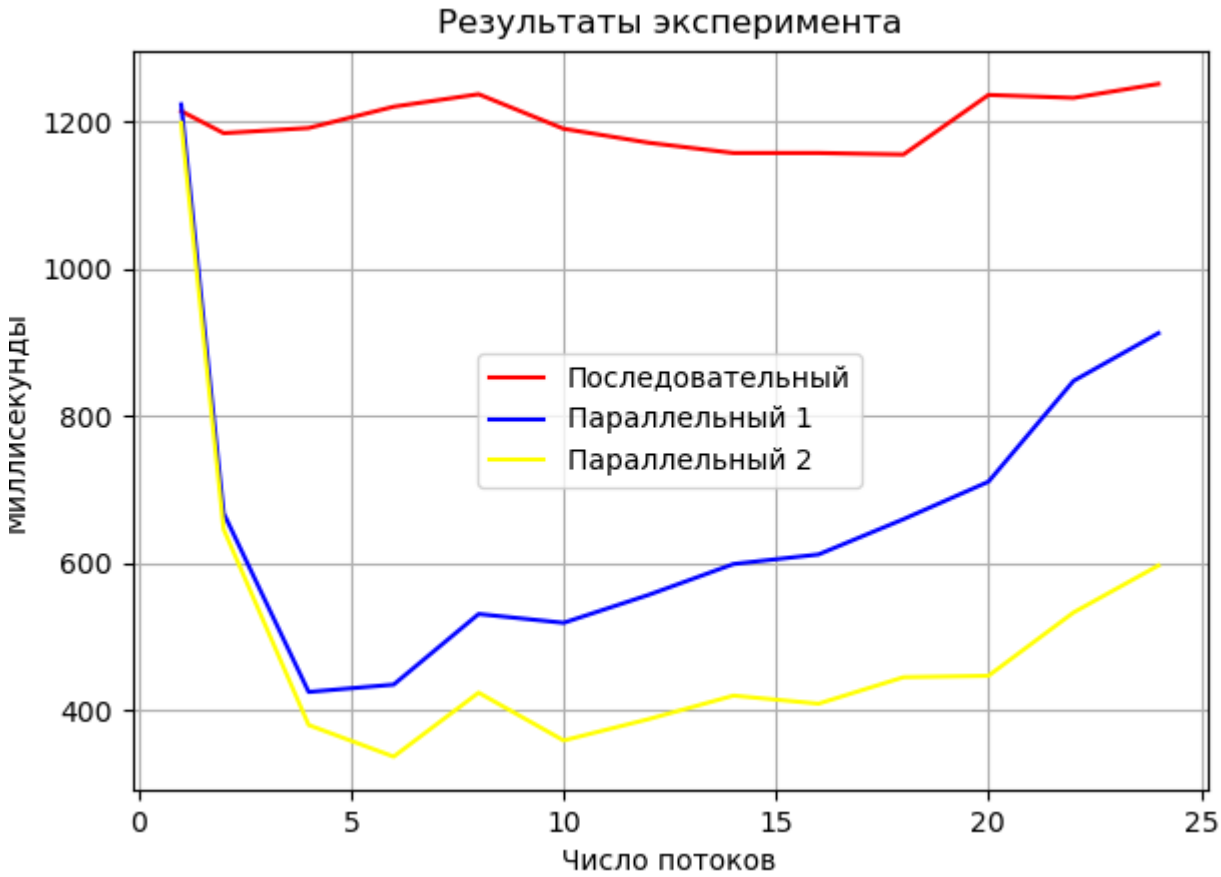

Рисунок 5 – Результаты на разном количестве потоков

На рисунке 6 представлено сравнение алгоритмов на 6 потоках(кроме последовательной реализации) на матрицах со сторонами от 10 до 1000 с шагом 99.

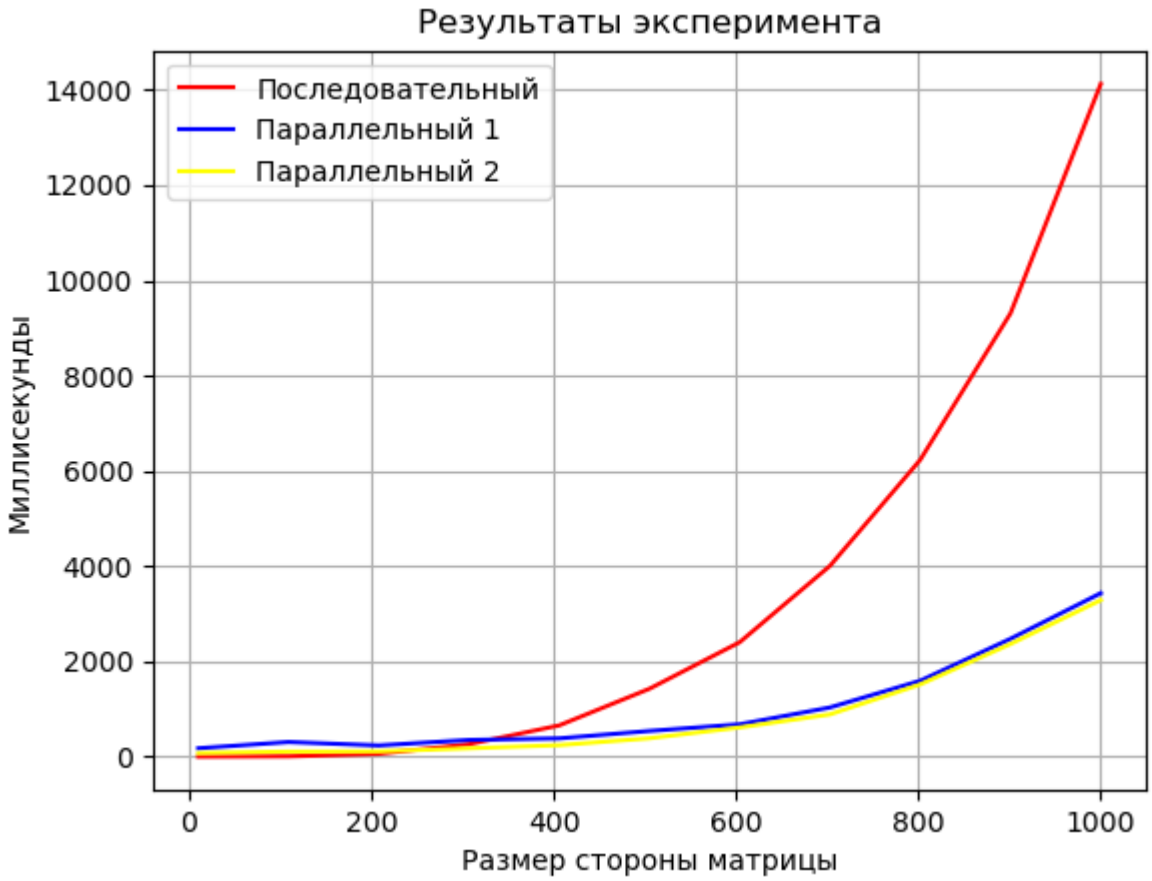

Рисунок 6 – Результаты на разных размерах матриц

4.3 Вывод

Как видно из графиков, самой долгой реализацией является последовательная версия алгоритма. Самой быстрой является первая версия параллельного алгоритма. Вторая версия быстрее последовательной, но уступает в скорости работы второй.

Так же видно, что при увеличении количества потоков от 1 до 6 параллельные версии алгоритмов работают все быстрее, но на количестве потоков больше 6 алгоритмы начинают немного замедляться. Это объясняется тем, что 6 - количество логических ядер машины, на которой проводились тесты, и, при количестве потоков больше 6, реализуется псевдопараллельность, которая занимает больше времени из-за смены контекстов.

Заключение

В данной лабораторной работе были разработаны параллельные алгоритмы Винограда. Они были реализованы, а также были проведены эксперименты по замеру времени работы реализованных алгоритмов и проведены сравнения алгоритмов по результатам эксперимента. Цель работы достигнута, все задачи выполнены.

Список литературы

[1] Умножение матриц [Электронный ресурс]. Режим доступа: (дата обращения - 02.10.2020) Свободный. URL: http://www.algolib.narod.ru/Math/Matrix.html

[2] API.NET. Пространство имен System.Threading [Электронный ресурс]. Режим доступа: (дата обращения - 28.10.2020) Свободный. URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.threading?view=netcore-3.1

[3] Visual Studio [Электронный ресурс]. Режим доступа: (дата обращения - 28.10.2020) Свободный. URL: https://visualstudio.microsoft.com/ru/