Министерство образования Российской Федерации
Московский Государственный Технический
Университет им. Н.Э. Баумана

Отчет по лабораторной работе №4
По курсу «Анализ алгоритмов»

Тема: «Многопоточное умножение матриц»

Студент: **Медведев А.В.**
Группа: **ИУ7-51**

Преподаватель: **Волкова Л.Л.**

Москва, 2017

Содержание

Постановка задачи

В ходе лабораторной работы предстоит:

1. реализовать алгоритм умножения матриц средствами N потоков;
2. сравнить алгоритм многопоточного умножения матриц с однпоточным;
3. провести эксперименты с замерами времени на матрицах разных размерностей;
4. сделать выводы об эффективности многопоточного умножения матриц.

Алгоритмы

Последовательный базовый алгоритм

Листинг 1: Последовательный базовый алгоритм умножения матриц

```
1  public static double[,] BaseMultiplication(double[,]
       matrix1, double[,] matrix2)
2  {
3      int resRow = matrix1.GetLength(0);
4      int resCol = matrix2.GetLength(1);
5      int kk = matrix2.GetLength(0);
6      double[,] resultmatrix = new double[resRow, resCol];
7
8      for (int i = 0; i < resRow; i++)
9      {
10         for (int j = 0; j < resCol; j++)
11         {
12             for (int k = 0; k < kk; k++)
13             {
14                 resultmatrix[i, j] = resultmatrix[i, j] +
                       matrix1[i, k] * matrix2[k, j];
15             }
16         }
17     }
18
19     return resultmatrix;
20 }
```

Входные данные: $matrix1$ - первая матрица, $matrix2$ - вторая матрица
Выходные данные: $resultmatrix$ - результирующая матрица

Последовательный алгоритм Винограда

Листинг 2: Последовательный алгоритм Винограда умножения матриц

```
public static double[,] VinogradMultiplication(double[,]
    matrix1, double[,] matrix2)
{
    int resRow = matrix1.GetLength(0);
    int resCol = matrix2.GetLength(1);
    int kk = matrix2.GetLength(0);
    double[,] resultmatrix = new double[resRow, resCol];

    double[] rowFactor = new double[resRow];
    double[] colFactor = new double[resCol];

    for (int i = 0; i < resRow; i++)
    {
        rowFactor[i] = matrix1[i, 0] * matrix1[i, 1];
        for (int j = 1; j < kk / 2; j++)
        {
            rowFactor[i] = rowFactor[i] + matrix1[i, 2 * j]
                * matrix1[i, 2 * j + 1];
        }
    }

    for (int i = 0; i < resCol; i++)
    {
        colFactor[i] = matrix2[0, i] * matrix2[1, i];
        for (int j = 1; j < kk / 2; j++)
        {
            colFactor[i] = colFactor[i] + matrix2[2 * j, i]
                * matrix2[2 * j + 1, i];
        }
    }

    for (int i = 0; i < resRow; i++)
    {
        for (int j = 0; j < resCol; j++)
        {
            resultmatrix[i, j] = -rowFactor[i] - colFactor[
                j];
            for (int k = 0; k < kk / 2; k++)
            {
```

```
36                  resultmatrix[i, j] = resultmatrix[i, j] +
37                                       (matrix1[i, 2 * k] +
                                            matrix2[2 * k + 1,
                                            j]) *
38                                       (matrix1[i, 2 * k + 1]
                                             + matrix2[2 * k, j
                                            ]);
39              }
40          }
41      }
42
43      if (kk % 2 == 1)
44      {
45          for (int i = 0; i < resRow; i++)
46          {
47              for (int j = 0; j < resCol; j++)
48              {
49                  resultmatrix[i, j] = resultmatrix[i, j] +
                        matrix1[i, kk - 1] * matrix2[kk - 1, j];
50              }
51          }
52      }
53
54      return resultmatrix;
55  }
```

Входные данные: $matrix1$ - первая матрица, $matrix2$ - вторая матрица
Выходные данные: $resultmatrix$ - результирующая матрица

Последовательный модифицированный алгоритм Винограда умножения матриц

Листинг 3: Последовательный модифицированный алгоритм Винограда умножения матриц

```
public static double[,] BetterVinogradMultiplication(double
    [,] matrix1, double[,] matrix2)
{
    int resRow = matrix1.GetLength(0); //m
    int resCol = matrix2.GetLength(1); //n

    int n = matrix2.GetLength(0);
    int kk = matrix2.GetLength(0) / 2;
    bool flag = n % 2 == 1;

    double[,] resultmatrix = new double[resRow, resCol];
    double[] rowFactor = new double[resRow];
    double[] colFactor = new double[resCol];

    for (int i = 0; i < resRow; i++)
    {
        rowFactor[i] = matrix1[i, 0] * matrix1[i, 1];
        for (int j = 1; j < kk; j++)
        {
            rowFactor[i] += matrix1[i, 2 * j] * matrix1[i,
                2 * j + 1];
        }
    }


    for (int i = 0; i < resCol; i++)
    {
        colFactor[i] = matrix2[0, i] * matrix2[1, i];
        for (int j = 1; j < kk; j++)
        {
            colFactor[i] += matrix2[2 * j, i] * matrix2[2 *
                j + 1, i];
        }
    }

    double buffer;
    for (int i = 0; i < resRow; i++)
```

```
35      {
36          for (int j = 0; j < resCol; j++)
37          {
38              buffer = (flag ? matrix1[i, n - 1] * matrix2[n
                    - 1, j] : 0);
39              buffer -= rowFactor[i] + colFactor[j];
40              for (int k = 0; k < kk; k++)
41              {
42                  buffer += (matrix1[i, 2 * k] + matrix2[2 *
                        k + 1, j]) *
43                            (matrix1[i, 2 * k + 1] + matrix2
                                [2 * k, j]);
44              }
45
46              resultmatrix[i, j] = buffer;
47          }
48      }
49
50      return resultmatrix;
51  }
```

Входные данные: $matrix1$ - первая матрица, $matrix2$ - вторая матрица
Выходные данные: $resultmatrix$ - результирующая матрица

В современных процессорах линейки Intel Core и некоторых других была реализована технология **hyper-threading**. При включении технологии каждое физическое ядро процессора определяется операционной системой как два логических ядра.

Преимуществами HTT считаются:

- возможность запуска нескольких потоков одновременно (многопоточный код);
- уменьшение времени отклика;
- увеличение числа одновременно обслуживаемых пользователей.

Количество логических ядер на компьютере, на котором выполнялась лабораторная работа, равняется 4:

Параллельный базовый алгоритм

В каждом потоке заполняются $\frac{n}{N}$ строк результирующей матрицы, где n - размерность матрицы (количество ее строк), а N - количество используемых потоков.

Листинг 4: Параллельный базовый алгоритм умножения матриц

```
public static double[,] ParallelBaseMultiplication(double
    [,] matrix1, double[,] matrix2, int nThreads)
{
    int resRow = matrix1.GetLength(0);
    int resCol = matrix2.GetLength(1);
    double[,] resultmatrix = new double[resRow, resCol];
    List<Thread> threads = new List<Thread>();
    int rowsForThread = resRow / nThreads;

    int firstInd = 0;
    for (int i = 0; i < nThreads; i++)
    {
        int lastInd = firstInd + rowsForThread;
        if (i == nThreads - 1)
            lastInd = resRow;

        MultThread tmp = new MultThread(matrix1, matrix2,
            resultmatrix, firstInd, lastInd);
```

```
17          threads.Add(new Thread(new ThreadStart(tmp.Run)));
18          firstInd = lastInd;
19      }
20
21      foreach (Thread thread in threads)
22      {
23          thread.Start();
24      }
25
26      foreach (Thread thread in threads)
27      {
28          thread.Join();
29      }
30
31      //Console.WriteLine("Finish");
32      return resultmatrix;
33  }
```

Входные данные: matrix1 - первая матрица, matrix2 - вторая матрица, CountOfThread - количество потоков
Выходные данные: resultmatrix - результирующая матрица

Класс MultThread - класс, разделяющий работу на потоки

Данные класса MultThread:
firstMatrix - первая матрица;
secondMatrix - вторая матрица;
result - результирующая матрица;
startIndex - с какой строки матрицы поток рассчитывает результат;
endIndex - до какой строки матрицы поток рассчитывает результат.

Листинг 5: Класс MultThread

```
1  public class MultThread
2  {
3      private double[,] firstMatrix;
4      private double[,] secondMatrix;
5      private double[,] resultMatrix;
6      private int startIndex;
7      private int endIndex;
8
9      public MultThread(double[,] firstMatrix, double[,]
           secondMatrix, double[,] resultMatrix, int startIndex
```

```
        , int endIndex)
10      {
11          this.firstMatrix = firstMatrix;
12          this.secondMatrix = secondMatrix;
13          this.resultMatrix = resultMatrix;
14          this.startIndex = startIndex;
15          this.endIndex = endIndex;
16
17      }
18
19      public void Run()
20      {
21          Console.WriteLine("Thread {Thread.CurrentThread.
                ManagedThreadId} calc from {startIndex} to {
                endIndex}");
22
23          int colCount= secondMatrix.GetLength(1);
24
25          for (int i = startIndex; i < endIndex; i++)
26          {
27              for (int j = 0; j < colCount; j++)
28              {
29                  double sum = 0;
30                  for (int k = 0; k < secondMatrix.GetLength
                        (0); k++)
31                  {
32                      sum += firstMatrix[i, k] * secondMatrix
                            [k, j];
33
34                  }
35
36                  resultMatrix[i, j] = sum;
37
38              }
39
40          }
41          Console.WriteLine("Thread {Thread.CurrentThread.
                ManagedThreadId} finished.");
42      }
43  }
```

Параллельный алгоритм Винограда

В каждом потоке заполняются $\frac{n}{N}$ строк результирующей матрицы, где n - размерность матрицы (количество ее строк), а N - количество используемых потоков.

При этом векторы $rowVector$ и $columnVector$ заполняются до распараллеливания и передаются в конструктор для каждого из потоков.

Листинг 6: Параллельный мод. алгоритм умножения матриц Винограда

```
1  public static double[,] ParallelVinogradMultiplication(
       double[,] matrix1, double[,] matrix2, int nThreads)
2  {
3      int resRow = matrix1.GetLength(0); //m
4      int resCol = matrix2.GetLength(1); //n
5
6      int n = matrix2.GetLength(0);
7      int kk = matrix2.GetLength(0) / 2;
8
9
10     double[,] resultmatrix = new double[resRow, resCol];
11     double[] rowFactor = new double[resRow];
12     double[] colFactor = new double[resCol];
13
14     for (int i = 0; i < resRow; i++)
15     {
16         rowFactor[i] = matrix1[i, 0] * matrix1[i, 1];
17         for (int j = 1; j < kk; j++)
18         {
19             rowFactor[i] += matrix1[i, 2 * j] * matrix1[i,
                   2 * j + 1];
20         }
21     }
22
23
24     for (int i = 0; i < resCol; i++)
25     {
26         colFactor[i] = matrix2[0, i] * matrix2[1, i];
27         for (int j = 1; j < kk; j++)
28         {
29             colFactor[i] += matrix2[2 * j, i] * matrix2[2 *
```

```
                    j + 1, i];
30              }
31          }
32
33          List<Thread> threads = new List<Thread>();
34          int rowsForThread = resRow / nThreads;
35          int firstInd = 0;
36
37          for (int i = 0; i < nThreads; i++)
38          {
39              int lastInd = firstInd + rowsForThread;
40              if (i == nThreads - 1)
41                  lastInd = resRow;
42
43              MultVinograd tmp = new MultVinograd(matrix1,
                    matrix2, resultmatrix, rowFactor, colFactor,
                    firstInd,
44                  lastInd);
45
46              threads.Add(new Thread(new ThreadStart(tmp.Run)));
47              firstInd = lastInd;
48          }
49
50          foreach (Thread thread in threads)
51          {
52              thread.Start();
53          }
54
55          foreach (Thread thread in threads)
56          {
57              thread.Join();
58          }
59
60          //Console.WriteLine("Finish");
61
62          return resultmatrix;
63      }
```

Класс MultVinograd -предназначен для работы с потоками.

Данные класса MultVinogradThread:
firstMatrix - первая матрица;
secondMatrix - вторая матрица;

result - результирующая матрица;
rowVector - массив с заранее посчитанными произведениями;
columnVector - массив с заранее посчитанными произведениями;
startIndex - с какой строки матрицы поток рассчитывает результат;
endIndex - до какой строки матрицы поток рассчитывает результат.

Листинг 7: Класс MultVinograd

```
public class MultVinograd
{
private readonly double[,] _firstMatrix;
private readonly double[,] _secondMatrix;
private readonly double[,] _resultMatrix;
private readonly double[] _rowFactor;
private readonly double[] _colFactor;
private readonly int _startIndex;
private readonly int _endIndex;

public MultVinograd(double[,] firstMatrix, double[,]
    secondMatrix, double[,] resultMatrix, double[] rowFactor
    , double[] colFactor, int startIndex, int endIndex)
{
    _firstMatrix = firstMatrix;
    _secondMatrix = secondMatrix;
    _resultMatrix = resultMatrix;
    _rowFactor = rowFactor;
    _colFactor = colFactor;
    _startIndex = startIndex;
    _endIndex = endIndex;
}

public void Run()
{

    //Console.WriteLine("VinoThread {Thread.CurrentThread.
        ManagedThreadId} calc from {_startIndex} to {
        _endIndex}");


    int colCount = _secondMatrix.GetLength(0);
    int halfRowCount = _secondMatrix.GetLength(0)/2;

    bool flag = colCount % 2 == 1;
```

```
32      for (int i = _startIndex; i < _endIndex; i++)
33      {
34          for (int j = 0; j < colCount; j++)
35          {
36              var buffer = (flag ? _firstMatrix[i, colCount -
                    1] * _secondMatrix[colCount - 1, j] : 0);
37              buffer -= _rowFactor[i] + _colFactor[j];
38              for (int k = 0; k < halfRowCount; k++)
39              {
40                  buffer +=(_firstMatrix[i, 2 * k] +
                        _secondMatrix[2 * k + 1, j]) *
41                              (_firstMatrix[i, 2 * k + 1] +
                                _secondMatrix[2 * k, j]);
42
43              }
44              _resultMatrix[i, j] = buffer;
45          }
46      }
47      Console.WriteLine("Thread {Thread.CurrentThread.
            ManagedThreadId} finished.");
48  }
```

Эксперимент

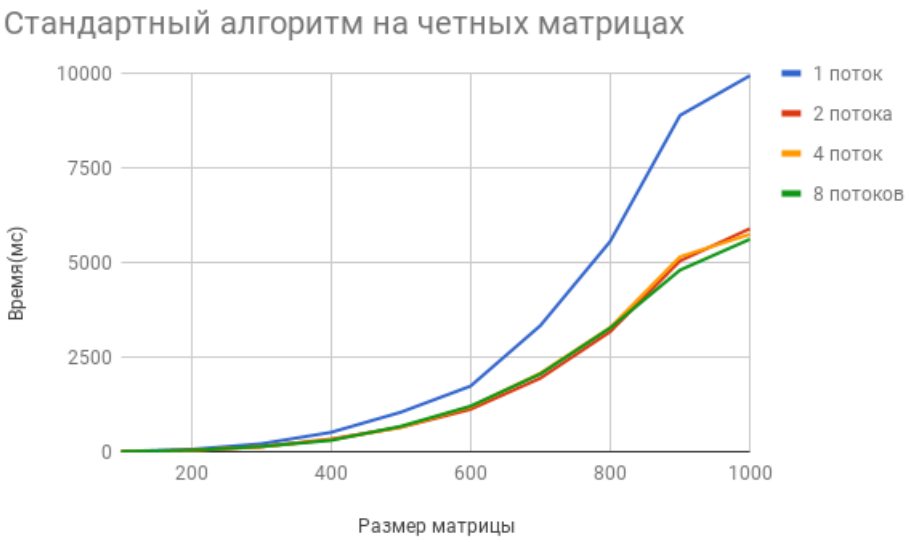
В качестве эксперимента было замерено время работы алгоритмов умножения матриц размерностями от 100*100 до 1000*1000 с шагом 100 средствами 2, 4 и 8 потоков. А так же произведен замер времени работы однопоточного алгоритма. Время измерялось в миллисекундах

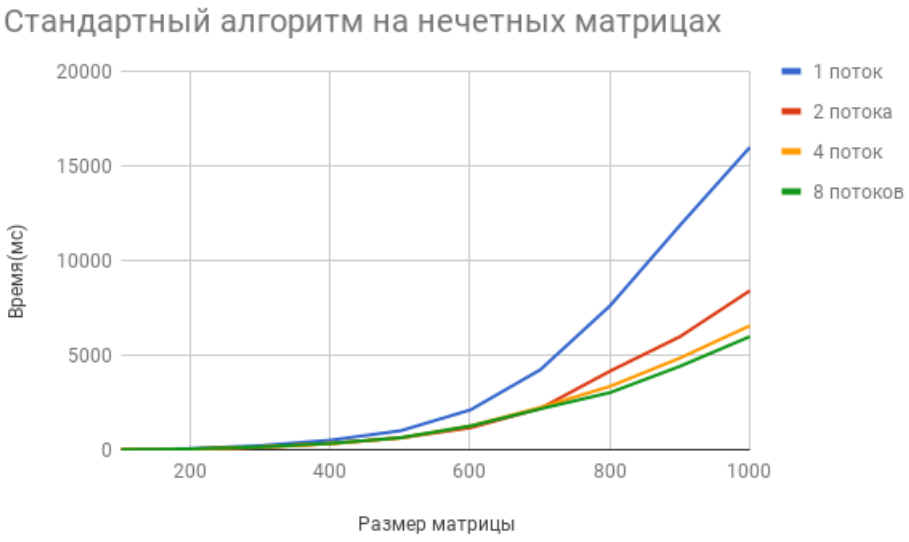
Таблица 1. Результаты эксперимента

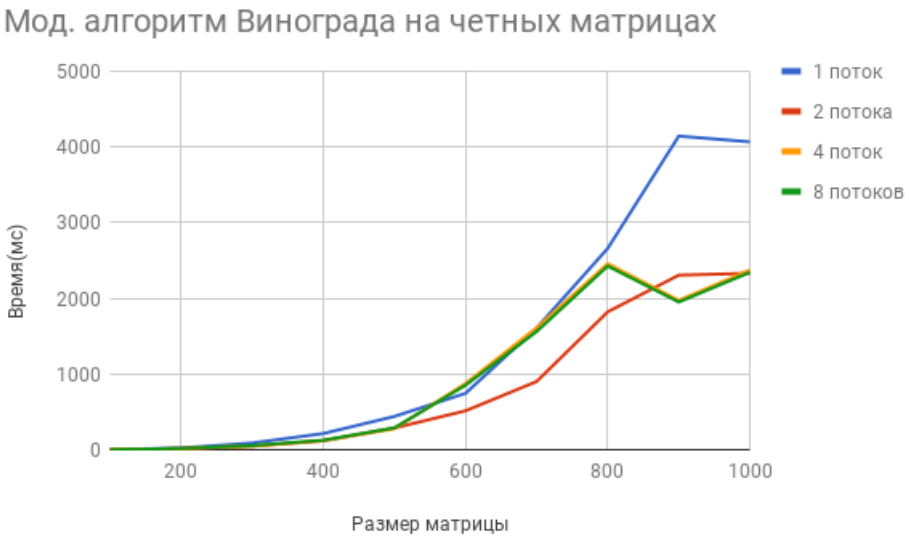
	Станд, (мсек)	**Виногр. (мсек)**	**Мод. (мсек)**	**Станд алгоритм, время (мсек)**				**Алгоритм Винограда, время (мсек)**			
Пот.	0	0	0	1	2	4	8	1	2	4	8
100	4	3	2	7	4	6	7	2	1	2	1
200	43	33	23	61	33	35	39	23	15	15	17
300	156	119	81	209	132	129	137	89	47	49	55
400	404	313	218	511	334	318	299	214	118	120	126
500	820	620	431	1043	644	645	671	440	287	277	289
600	1383	1073	746	1733	1120	1186	1204	745	514	876	852
700	2708	2472	1622	3332	1942	2075	2052	1608	901	1611	1561
800	4514	3673	2615	5556	3170	3281	3269	2659	1821	2457	2426
900	8025	6637	4498	8886	5043	5151	4799	4143	2306	1973	1951
1000	7574	6185	4099	9934	5892	5743	5613	4067	2330	2375	2343

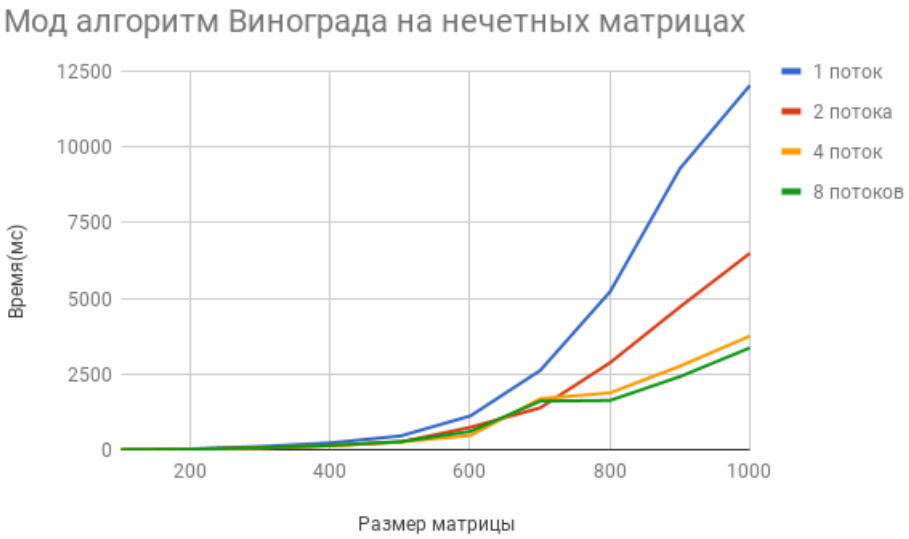
Таблица 2. Результаты эксперимента

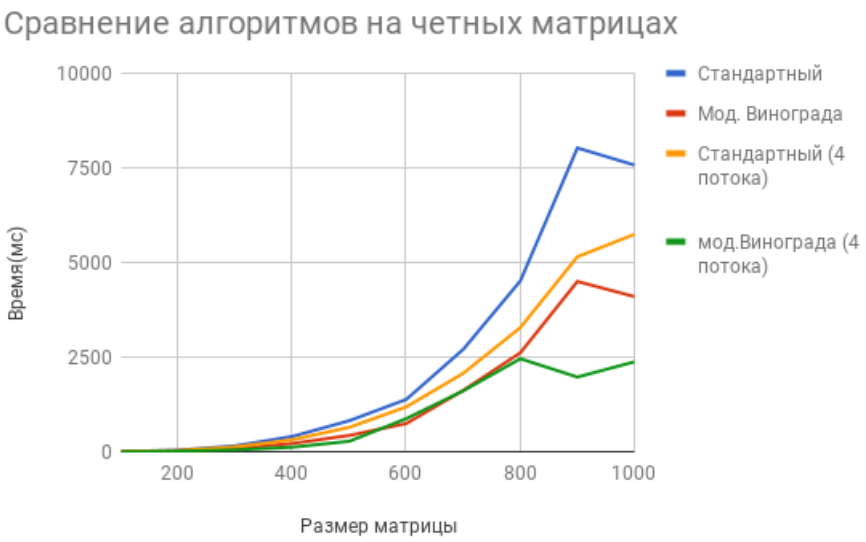
	Станд, (мсек)	**Виногр. (мсек)**	**Мод. (мсек)**	**Станд. алгоритм, время (мсек)**				**Алгоритм Винограда, время (мсек)**			
Пот.	0	0	0	1	2	4	8	1	2	4	8
101	5	3	2	7	4	6	5	2	1	1	2
201	48	37	26	61	35	38	48	28	16	16	20
301	176	140	103	217	131	134	149	106	63	55	66
401	404	317	226	508	326	325	336	228	138	130	143
501	796	622	450	1004	627	625	637	455	256	257	267
601	1757	1410	1112	2106	1160	1254	1254	1118	742	470	605
701	3678	3179	2643	4228	2181	2237	2159	2621	1384	1683	1606
801	6954	6269	5663	7614	4157	3350	3020	5223	2877	1878	1629
901	10409	9182	8137	11860	5976	4846	4411	9287	4717	2759	2415
1001	15091	13807	12856	15981	8403	6552	5977	12031	6487	3755	3364

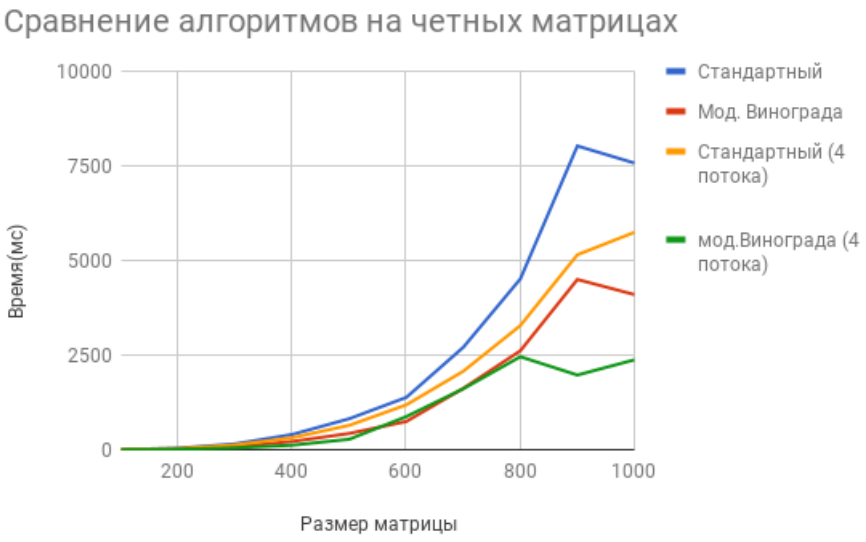

Стандартный алгоритм на четных матрицах
10000
7500
5000
2500
0
Время(мс)
200
400
600
800
1000
Размер матрицы
1 поток
2 потока
4 поток
8 потоков


Стандартный алгоритм на нечетных матрицах
20000
15000
10000
5000
0
Время(мс)
200
400
600
800
1000
Размер матрицы
1 поток
2 потока
4 поток
8 потоков


Мод. алгоритм Винограда на четных матрицах
5000
4000
3000
2000
1000
0
Время(мс)
200
400
600
800
1000
Размер матрицы
1 поток
2 потока
4 поток
8 потоков


Мод алгоритм Винограда на нечетных матрицах
12500
10000
7500
5000
2500
0
Время(мс)
200
400
600
800
1000
Размер матрицы
1 поток
2 потока
4 поток
8 потоков


Сравнение алгоритмов на четных матрицах
10000
7500
5000
2500
0
Время(мс)
200
400
600
800
1000
Размер матрицы
Стандартный
Мод. Винограда
Стандартный (4 потока)
мод.Винограда (4 потока)


Сравнение алгоритмов на четных матрицах
10000
7500
5000
2500
0
Время(мс)
200
400
600
800
1000
Размер матрицы
Стандартный
Мод. Винограда
Стандартный (4 потока)
мод.Винограда (4 потока)

Выводы из эксперимента

В результате эксперимента было выявлено, что использование потоков повышает эффективность алгоритма умножения матриц. В данной лабораторной работе был выбран способ, который заключается в разбиении задачи на N независимых подзадач умножения матрицы размерностью $\frac{n}{N}$ на матрицу размерностью n, где N - количество потоков, n - размерность исходных квадратных матриц. Подобная методика построения параллельных методов решения сложных задач является одной из основных в параллельном программировании и широко используется в практике. Параллельные методы матричного умножения приводят к равномерному распределению вычислительной нагрузки между процессорами.

Заключение

В ходе лабораторной работы были реализованы алгоритмы умножения матриц средствами 2,4 и 8 потоков, произведено сравнение алгоритмов многопоточного умножения матриц с однопоточным, а также проведены эксперименты с замерами времени на матрицах разных размерностей.