МГТУ им. Баумана

Лабораторная работа №4

По курсу: "Анализ алгоритмов"

Параллельное программирование

Работу выполнил: студент группы ИУ7-53Б
Наместник Анастасия

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

Москва, 2020

Оглавление

Введение

Параллельное программирование служит для создания программ, эффективно использующих вычислительные ресурсы за счет одновременного исполнения кода на нескольких вычислительных узлах. Использование параллельного программирования становится наиболее необходимым, поскольку позволяет максимально эффективно использовать возможности многоядерных процессоров и многопроцессорных систем. По ряду причин, включая повышение потребления энергии и ограничения пропускной способности памяти, увеличивать тактовую частоту современных процессоров стало невозможно. Вместо этого производители процессоров стали увеличивать их производительность за счет размещения в одном чипе нескольких вычислительных ядер, не меняя или даже снижая тактовую частоту. Поэтому для увеличения скорости работы приложений теперь следует по-новому подходить к организации кода, а именно - оптимизировать программы под многоядерные системы [4]. В рамках этой лабораторной работы будет рассмотрено параллельное программирование на примере стандартного алгоритма умножения матриц.

Целью данной лабораторной работы является изучение и реализация параллельного программирования для решения задачи умножения матриц, а также сравнительный анализ затрачиваемых временных ресурсов при параллельной и однопоточной реализаций одного и того же алгоритма.

В данной лабораторной работе требуется решить четыре задачи.

1. Изучить основы многопоточного программирования.
2. Придумать два варианта многопоточной реализации стандартного алгоритма умножения матриц.
3. Программно реализовать стандартный алгоритм умножения матриц в однопоточном режиме.

4. Сделать сравнительный анализ по затрачиваемым ресурсам (времени) компьютера на реализацию каждого рассмотренного алгоритма.

1 Аналитическая часть

В данной лабораторной работе метод параллельного программирования будет применен к стандартному алгоритму умножения матриц. Ниже будут представлены теоретические сведения, необходимые для программной реализации этой задачи.

1.1 Стандартный алгоритм умножения матриц

Допустим, имеются матрицы A (формула 1.1) и B (формула 1.2).

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \tag{1.1}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{pmatrix} \tag{1.2}$$

Матрица C = AB будет размерностью $l \times n$, где матрица A размерностью $l \times m$, а матрица B $m \times n$ Тогда каждый элемент матрицы C выражается формулой (1.3)

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{ri} \quad (i = 1, 2, \dots l; j = 1, 2, \dots n) \tag{1.3}$$

Процесс произведения матриц проиллюстрирован рисунком 1.1.

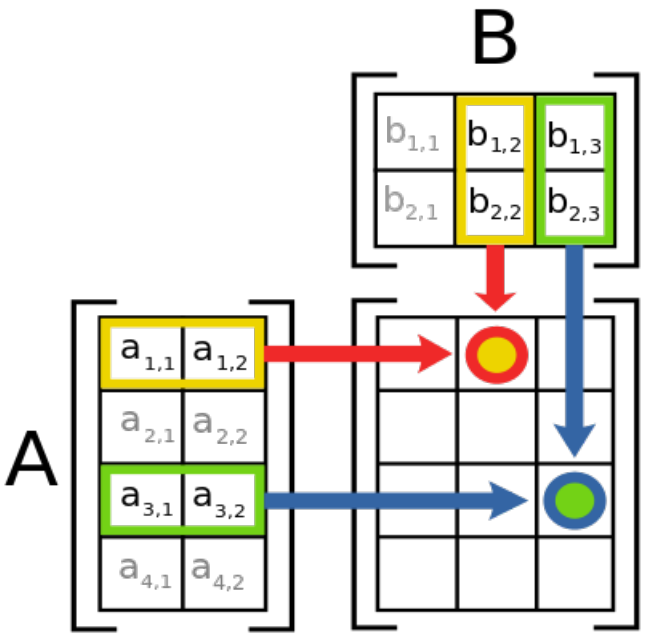

Рис 1.1: Произведение матриц

Важно отметить низкое быстродействие данного алгоритма и достаточно высокую трудоемкость (13NMQ). Следовательно, распараллеливание стандартного алгоритма умножения матриц должно повысить эффективность алгоритма.

1.2 Параллельное программирование

Параллельное программирование - это техника программирования, которая использует преимущества многоядерных или многопроцессорных компьютеров и является подмножеством более широкого понятия многопоточности. Таким образом, Параллельные вычисления - способ организации компьютерных вычислений, при котором программы разрабатываются, как набор взаимодействующих вычислительных процессов, работающих асинхронно и при этом одновременно [4].

1.3 Параллельная реализация стандартного алгоритма умножения матриц

Ввиду того, что в стандартном алгоритме умножения матриц каждая строка и столбец результирующей матрицы считаются независимо, можно выделить два варианта применения многопоточного программирования к данному алгоритму.

1. Распараллеливание по строкам (строки результирующей матрицы вычисляются параллельно).

2. Распараллеливание по столбцам (столбцы результирующей матрицы вычисляются параллельно).

1.4 Вывод

В данном разделе были рассмотрены теоретические сведения о стандартном алгоритме умножения матриц и параллельном программировании, а также были приведены два варианта применения этого подхода к описанному выше алгоритму.

2 | Конструкторская часть

В данном разделе будут представлены схемы последовательной и параллельной реализаций стандартного алгоритма умножения матриц.

2.1 Стандартный алгоритм произведения матриц

На рисунке 2.2 представлена схема стандартного алгоритма произведения матриц.

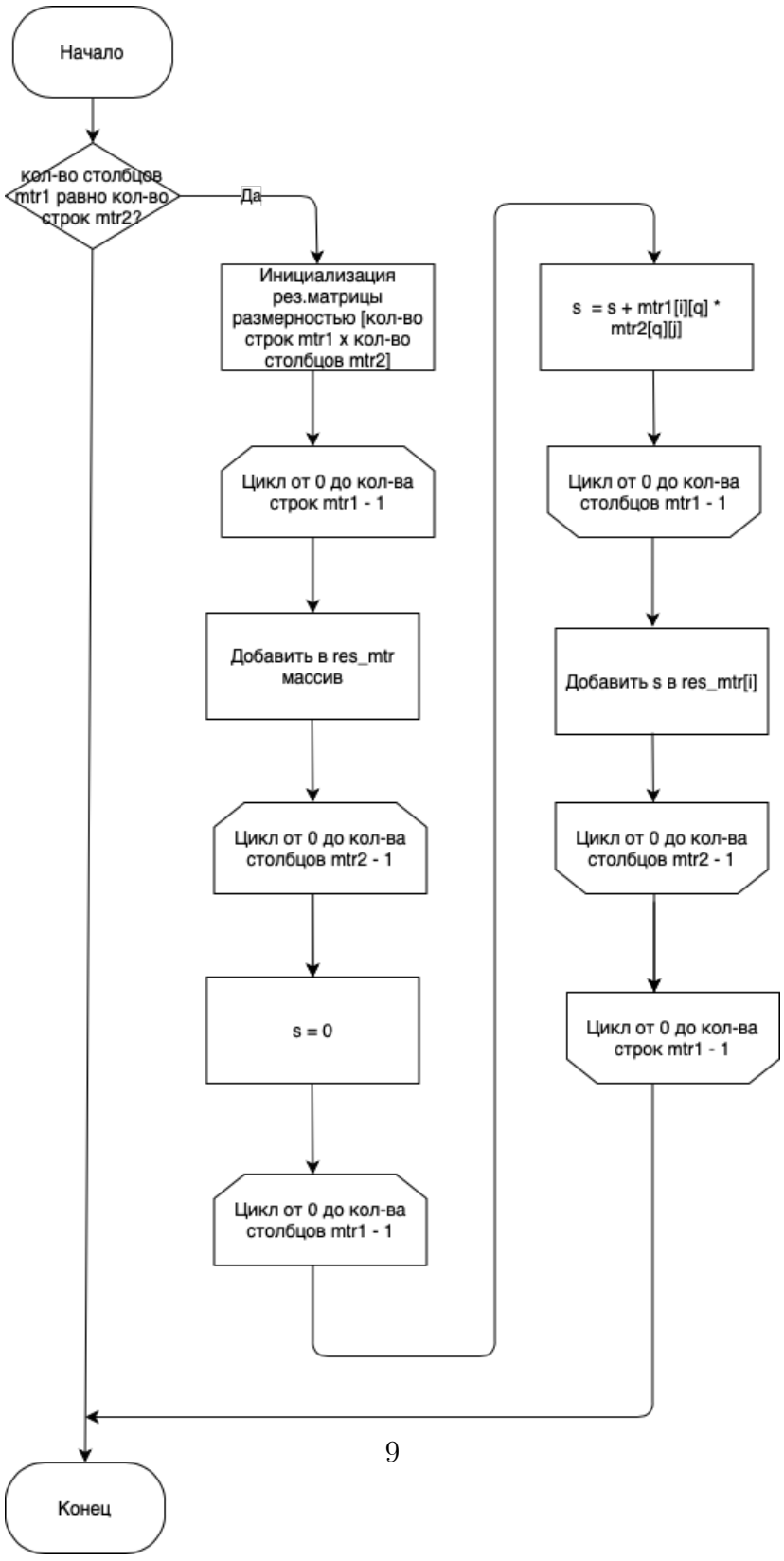
Начало
кол-во столбцов mtr1 равно кол-во строк mtr2?
Да
Инициализация рез.матрицы размерностью [кол-во строк mtr1 x кол-во столбцов mtr2]
Цикл от 0 до кол-ва строк mtr1 - 1
Добавить в res_mtr массив
Цикл от 0 до кол-ва столбцов mtr2 - 1
s = 0
Цикл от 0 до кол-ва столбцов mtr1 - 1
s = s + mtr1[i][q] * mtr2[q][j]
Цикл от 0 до кол-ва столбцов mtr1 - 1
Добавить s в res_mtr[i]
Цикл от 0 до кол-ва столбцов mtr2 - 1
Цикл от 0 до кол-ва строк mtr1 - 1
Конец

Рис 2.1: Схема стандартного алгоритма произведения матриц

2.2 Параллельная реализация при обходе по строкам

На рисунке 2.3 представлена схема главного потока параллельной реализации стандартного алгоритма произведения матриц.

Рис 2.3: Схема главного потока (обход по строкам)

На рисунке 2.4 представлена схема рабочего потока параллельной реализации стандартного алгоритма произведения матриц.

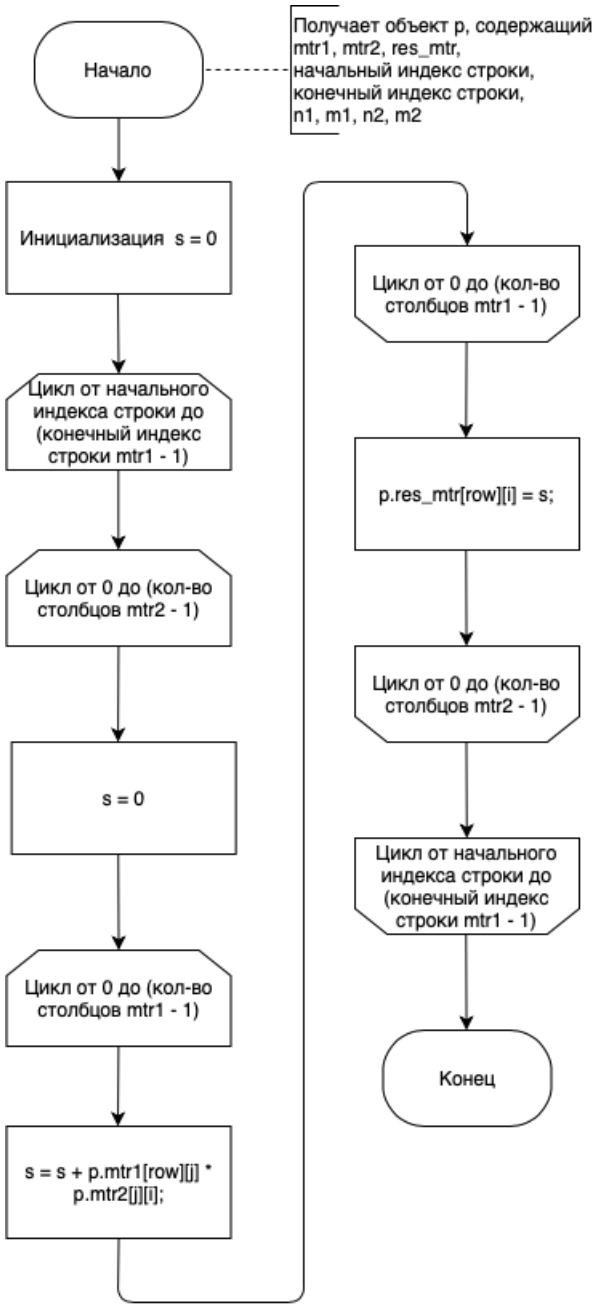

Рис 2.4: Схема рабочего потока (обход по строкам)

2.3 Параллельная реализация при обходе по столбцам

На рисунке 2.5 представлена схема главного потока параллельной реализации стандартного алгоритма произведения матриц.

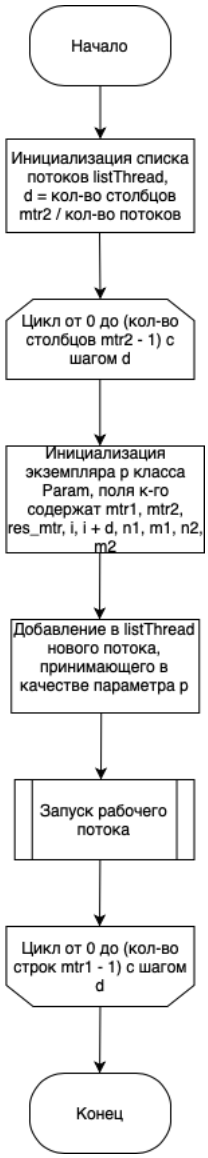

Рис 2.5: Схема главного потока (обход по столбцам)

На рисунке 2.6 представлена схема рабочего потока параллельной реализации стандартного алгоритма произведения матриц.

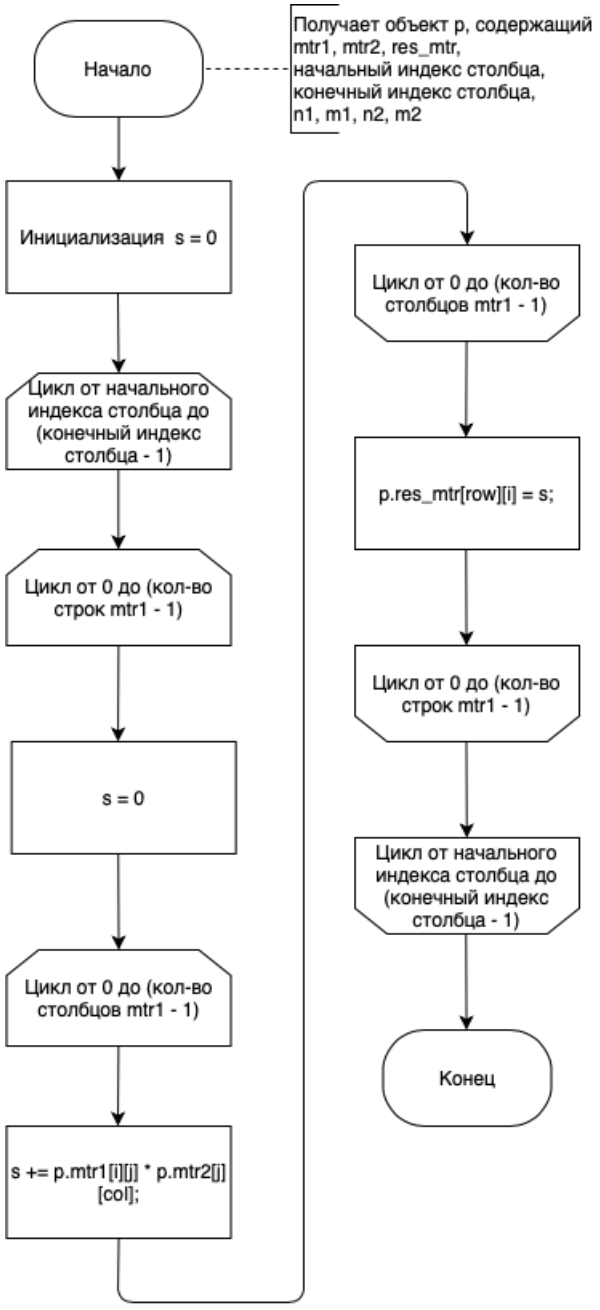

Рис 2.6: Схема рабочего потока (обход по столбцам)

2.4 Вывод

В данном разделе были рассмотрены 5 схем: стандартного алгоритма умножения матриц, главного и рабочего потоков параллельной реализации этого алгоритма при обходе по строкам, главного и рабочего потоков параллельной реализации этого алгоритма при обходе по столбцам.

3 | Технологическая часть

3.1 Выбор языка программирования

В данной лабораторной работе использовался язык программирования - C# [3], так как данный язык программирования поддерживает параллельное программирования, он является нативным. В качестве интегрированной среды разработки использовалась Visual studio [1]. Для замеров времени использовались методы Start() и Stop() класса Stopwatch [2]. Для генерации матрицы использовался метод Next класса Random [5].

3.2 Сведения о модулях программы

Программа состоит из следующих модулей.

- Program.cs - однопоточная реализация.
- ParallelMult.cs - многопоточная реализация.

На листинге 3.1 представлена подпрограмма однопоточной реализации стандартного алгоритма умножения матриц

Листинг 3.1: Подпрограмма однопоточной реализации стандартного алгоритма умножения матриц

```
1 public static int[][] StandMult(int[][] mtr1, int[][] mtr2)
2         {
3             int n1 = mtr1.Length;
4             int n2 = mtr2.Length;
5 
6             if (n1 == 0 || n2 == 0)
```

```
7                 return null;
8
9             int m1 = mtr1[0].Length;
10            int m2 = mtr2[0].Length;
11
12            if (m1 != n2)
13                return null;
14
15            int[][] res\_mtr = new int[n1][];
16            for (int i = 0; i < n1; i++)
17                res\_mtr[i] = new int[m2];
18
19            for (int i = 0; i < n1; i++)
20            {
21                for (int j = 0; j < m2; j++)
22                {
23                    for (int q = 0; q < m1; q++)
24                    {
25                        res\_mtr[i][j] = res\_mtr[i][j] +
                              mtr1[i][q] * mtr2[q][j];
26                    }
27                }
28            }
29
30            return res\_mtr;
31        }
```

На листинге 3.2 представлена подпрограмма главного потока многопоточной реализации стандартного алгоритма умножения матриц при обходе по строкам.

Листинг 3.2: Подпрограмма главного потока многопоточной реализации стандартного алгоритма умножения матриц (обход по строкам)

```
1 public static void ParallelMultFirst(int[][] res_mtr, int
     [][] mtr1, int[][] mtr2, int n1, int m1, int n2, int m2,
      int tc)
2         {
3             List<Thread> listThread = new List<Thread>();
4             int d = n1 / tc;
5             int j = 0;
```

```
6
7               for (int i = 0; i < n1; i += d)
8               {
9                   //Console.WriteLine("Main thread:");
10                  Param p = new Param(res\_mtr, mtr1, mtr2, i
                        , i + d, n1, m1, n2, m2);
11
12                  listThread.Add(new Thread(new
                        ParameterizedThreadStart(ParallelMultRow
                        )));
13                  listThread[j].Start(p);
14                  j += 1;
15               }
16
17              foreach (var elem in listThread)
18              {
19
20                  if (elem.IsAlive)
21                      elem.Join();
22              }
23          }
```

На листинге 3.3 представлена подпрограмма рабочего потока многопоточной реализации стандартного алгоритма умножения матриц при обходе по строкам.

Листинг 3.3: Подпрограмма рабочего потока многопоточной реализации стандартного алгоритма умножения матриц (обход по строкам)

```
1 public static void ParallelMultRow(object obj)
2         {
3             Param p = (Param)obj;
4             int s = 0;
5
6             for (int row = p.start; row < p.end; row++)
7             {
8                 for (int i = 0; i < p.m2; i++)
9                 {
10                    s = 0;
11                    for (int j = 0; j < p.m1; j++)
12                    {
```

```
13                          s += p.mtr1[row][j] * p.mtr2[j][i];
14                      }
15                      p.res\_mtr[row][i] = s;
16                  }
17              }
18          }
```

На листинге 3.4 представлена подпрограмма главного потока многопоточной реализации стандартного алгоритма умножения матриц при обходе по столбцам.

Листинг 3.4: Подпрограмма главного потока многопоточной реализации стандартного алгоритма умножения матриц (обход по столбцам)

```
1  public static void ParallelMultSecond(int[][] res_mtr, int
       [][] mtr1, int[][] mtr2, int n1, int m1, int n2, int m2,
        int tc)
2          {
3              int d = m2 / tc;
4              int j = 0;
5
6              List<Thread> listThread = new List<Thread>();
7
8              for (int i = 0; i < m2; i += d)
9              {
10                 Param p = new Param(res\_mtr, mtr1, mtr2, i
                       , i + d, n1, m1, n2, m2);
11
12                 listThread.Add(new Thread(new
                       ParameterizedThreadStart(ParallelMultCol
                       )));
13                 listThread[j].Start(p);
14                 j += 1;
15             }
16             //Console.WriteLine("Main thread:");
17
18             foreach (var elem in listThread)
19             {
20                 if (elem.IsAlive)
21                     elem.Join();
22             }
```

```
23          }
```

На листинге 3.5 представлена подпрограмма рабочего потока многопоточной реализации стандартного алгоритма умножения матриц при обходе по столбцам.

Листинг 3.5: Подпрограмма рабочего потока многопоточной реализации стандартного алгоритма умножения матриц (обход по столбцам)

```
1  public static void ParallelMultCol(object obj)
2          {
3              Param p = (Param)obj;
4              int s = 0;
5
6              for (int col = p.start; col < p.end; col++)
7              {
8                  for (int i = 0; i < p.n1; i++)
9                  {
10                     s = 0;
11                     for (int j = 0; j < p.m1; j++)
12                     {
13                         s += p.mtr1[i][j] * p.mtr2[j][col];
14                     }
15                     p.res\_mtr[i][col] = s;
16                 }
17             }
18         }
```

На листинге 3.6 представлена подпрограмма создания матрицы.

Листинг 3.6: Подпрограмма создания матрицы

```
1  public static int[][] InitMatrix(int n, int m)
2          {
3              int[][] mtr = new int[n][];
4              for (int i = 0; i < n; i++)
5                  mtr[i] = new int[m];
6
7              Console.WriteLine("Enter the matrix: ");
8              for (int i = 0; i < n; i++)
9              {
10                 for (int j = 0; j < m; j++)
11                 {
```

```
12                      mtr[i][j] = Convert.ToInt32(Console.
                            ReadLine());
13                  }
14              }
15              retu
```

На листинге 3.7 представлена подпрограмма рандомной генерации матрицы .

Листинг 3.7: Подпрограмма создания матрицы с использованием метода Next класса Random [5]

```
1  public static int[][] RandomMatrix(int n, int m)
2          {
3              int[][] mtr = new int[n][];
4              for (int i = 0; i < n; i++)
5                  mtr[i] = new int[m];
6
7              for (int i = 0; i < n; i++)
8              {
9                  for (int j = 0; j < m; j++)
10                 {
11                     mtr[i][j] = rand.Next(1, 10);
12                 }
13             }
14             return mtr;
15         }
```

Листинг 3.8: класс Param

```
1  class Param
2      {
3          public int[][] res\_mtr, mtr1, mtr2;
4          public int n1, m1, n2, m2;
5          public int start, end;
6
7          public Param(int[][] res\_mtr, int[][] mtr1, int
               [][] mtr2, int start, int end, int n1, int m1,
              int n2, int m2)
8          {
9              this.res\_mtr = res\_mtr;
10             this.mtr1 = mtr1;
```

```
11              this.mtr2 = mtr2;
12              this.n1 = n1;
13              this.m1 = m1;
14              this.n2 = n2;
15              this.m2 = m2;
16              this.start = start;
17              this.end = end;
18          }
19      }
```

3.3 Тесты

В данном разделе будут представлены таблицы с тестами (таблицы 3.1 - 3.2). Тестировался наилучший случай, когда количество потоков равно числу логических ядер ЭВМ, на которой проводилось тестирование, - 8.

Таблица 3.1: Тестирование результата умножения матриц в параллельной реализации алгоритма при обходе по строкам (количество потоков = 8)

Матрица 1	Матрица 2	Результат
8 5	5 5	Верный
8 10	10 10	Верный
8 5	5 10	Верный
16 2	2 3	Верный
32 5	3 5	Аварийное завершение программы

Примечание: аварийное завершение программы происходит, потому что количество столбцов Матрица1 не совпадает с количеством строк Матрица2.

Таблица 3.2: Тестирование результата умножения матриц в параллельной реализации алгоритма при обходе по столбцам (количество потоков = 8)

Матрица 1	Матрица 2	Результат
5 5	5 8	Верный
10 10	10 8	Верный
10 5	5 8	Верный
3 2	2 16	Верный
5 5	3 16	Аварийное завершение программы

Примечание: аварийное завершение программы происходит, потому что количество столбцов Матрица1 не совпадает с количеством строк Матрица2.

3.4 Вывод

В технологической части были представлены модули программы, листинги кода и тестов к программе, а также обусловлен выбор языка программирования и приведены использовавшиеся в ходе работы инструменты.

4 | Исследовательская часть

В этом разделе будет проведено несколько экспериментов для сравнительного анализа затрачиваемого времени при выполнении стандартного алгоритма умножения матриц с использованием разного числа потоков. Это позволит оценить эффективность параллельной реализации алгоритма в сравнении с последовательной (однопоточной) и установить зависимость скорости выполнения программы от числа логических ядер используемой ЭВМ.

4.1 Характеристики ЭВМ

- MacBook Pro (Retina, 15-inch, Mid 2014).
- 2,5 GHz Intel Core i7.
- Число логических ядер: 8.

4.2 Результат замеров времени выполнения последовательной и параллельной реализаций стандартного алгоритма умножения матриц при обходе по строкам

Сравним временные показатели работы разных реализаций алгоритма. Так как процедура умножения матриц не является затратной по времени, воспользуемся усреднением массового эксперимента. Для этого установим количество итераций (повторений вызова процедуры) iter и воспользуемся библиотекой методами Start и Stop класса Stopwatch для

замера времени выполнения алгоритма по iter раз. Возьмем iter = 10, а обе матрицы, участвующие в эксперименте, будем получать с помощью рандомной генерации их элементов (метод Next класса Random). Ввиду высокой скорости выполнения вычислений при умножении матриц небольших размеров, для наглядности результатов данного эксперимента возьмем размерность первой матрицы равной [32x1000], а второй - [1000x1000]. При обходе по строкам количество строк результирующей матрицы делится на равные части, которые затем обрабатываются асинхронно и одновременно, каждая одним рабочим потоком. Главный поток запускает рабочие потоки. Эксперимент был проведен со следующим набором потоков: 1, 2, 4, 8, 16, 32. 32 = 4 * 8, где 8 - это количество ядер использовавшейся ЭВМ.

На рисунке 4.1 показана работа однопоточной и многопоточной реализаций стандартного алгоритма умножения матриц при обходе по строкам.

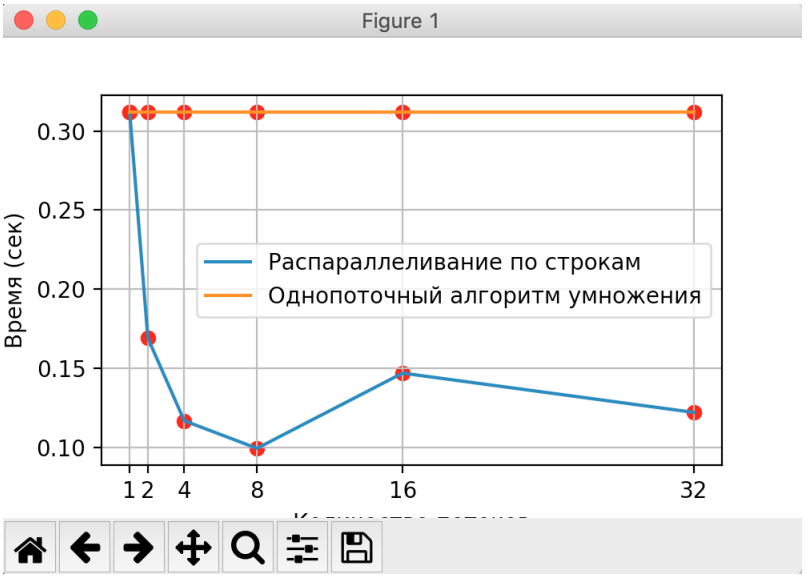

Рис 4.1: Сравнение времени выполнения алгоритмов

В качестве примера в таблице 4.1 представлены временные характеристики однопоточной и многопоточной реализаций стандартного алгоритма умножения матриц при обходе по строкам.

Таблица 4.1: Время выполнения однопоточной и многопоточной

реализаций стандартного алгоритма умножения матриц

Количество потоков	Время (сек.)
1	0.3119
2	0.1691
4	0.1169
8	0.0995
16	0.147
32	0.1222

Как видно из рисунка 4.1 и таблицы 4.1, наибольшая скорость выполнения стандартного алгоритма произведения матриц достигается при параллельной реализации и количестве потоков, равном 8. По сравнению с однопоточной реализацией выигрыш по временной эффективности получается приблизительно в 3 раза. Далее при увеличении числа потоков время выполнения увеличивается, что объясняется очередью потоков (ситуация, когда задач больше, чем потоков) Рассмотрим временные характеристики алгоритма при фиксированном числе потоков (8) и наборе матриц размерностью [32x1000] x [1000x1000], [64x1000] x [1000x1000], ..., [Nx1000] x [1000x1000], где N = 32*6, (рисунок 4.2).

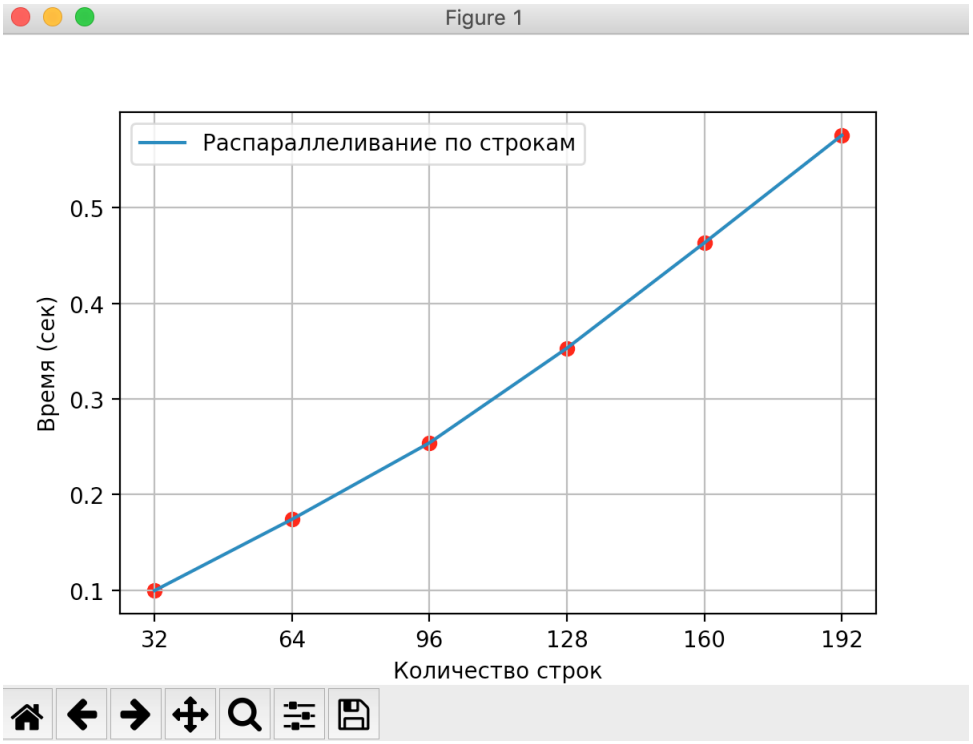

Рис 4.2: Сравнение времени выполнения при разном количестве строк результирующей матрицы и числе потоков, равном 8

Из рисунка 4.2 видно, что время выполнения распараллеленного алгоритма в зависимости от пропорционально увеличивающегося количества обрабатываемых потоками строк растет приблизительной линейно.

4.3 Результат замеров времени выполнения последовательной и параллельной реализаций стандартного алгоритма умножения матриц при обходе по столбцам

Ввиду высокой скорости выполнения вычислений при умножении матриц небольших размеров, для наглядности результатов данного эксперимента возьмем размерность первой матрицы равной [1000x1000], а второй - [1000x32]. При обходе по столбцам количество столбцов результирующей матрицы делится на равные части, которые затем обрабатываются асинхронно и одновременно, каждая одним рабочим потоком. Главный поток запускает рабочие потоки. Эксперимент был проведен со следующим набором потоков: 1, 2, 4, 8, 16, 32. 32 = 4 * 8, где 8 - это количество ядер использовавшейся ЭВМ.

На рисунке 4.3 показана работа однопоточной и многопоточной реализациий стандартного алгоритма умножения матриц при обходе по столбцам.

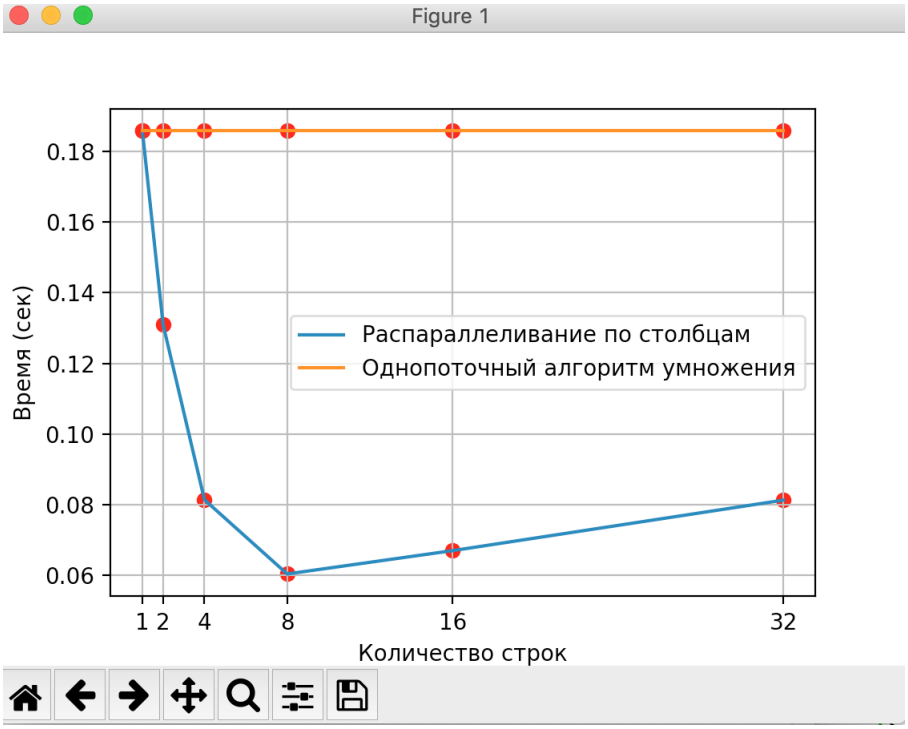

Рис 4.3: Сравнение времени выполнения алгоритмов

В качестве примера в таблице 4.2 представлены временные характеристики однопоточной и многопоточной реализаций стандартного алгоритма умножения матриц при обходе по столбцам.

Таблица 4.2: Время выполнения однопоточной и многопоточной реализаций стандартного алгоритма умножения матриц

Количество потоков	Время (сек.)
1	0.1858
2	0.131
4	0.0814
8	0.0604
16	0.067
32	0.0813

Как видно из рисунка 4.3 и таблицы 4.2, наибольшая скорость выполнения стандартного алгоритма произведения матриц достигается при параллельной реализации и количестве потоков, равном 8. По сравнению с однопоточной реализацией выигрыш по временной эффективности также получается приблизительно в 3 раза. При увеличении числа потоков вновь возникает очередь к процессору, и, следовательно, производительность снижается. Рассмотрим временные характеристики алгоритма при фиксированном числе потоков (8) и наборе матриц размерностью [1000x1000] x [1000x32], [1000x1000] x [1000x64], ..., [Mx1000] x [1000x1000], где M = 32 * 6, (рисунок 4.4).

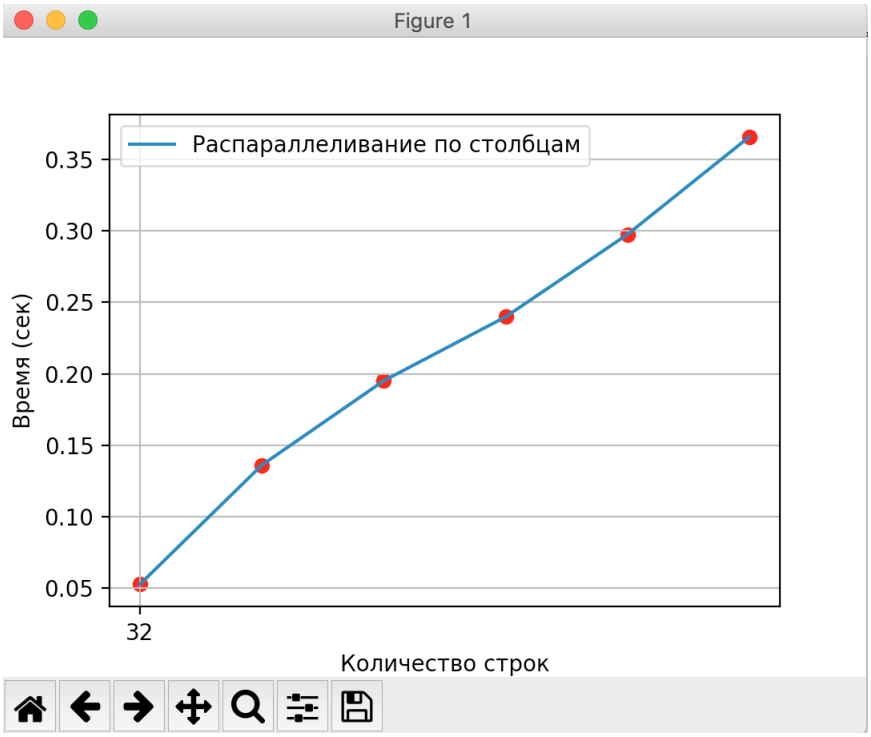

Рис 4.4: Сравнение времени выполнения при разном количестве столбцов результирующей матрицы и числе потоков, равном 8

Из рисунка 4.4 видно, что время выполнения распараллеленного алгоритма в зависимости от пропорционально увеличивающегося количества обрабатываемых потоками строк растет приблизительной линейно.

4.4 Вывод

В ходе эксперимента было установлено, что максимальной эффективности по времени стандартный алгоритм умножения матриц достигает при параллельной реализации с использованием 8 потоков, следовательно, при задействовании всех логических ядер ЭВМ, участвовавшей в эксперименте.

Заключение

В ходе лабораторной работы были реализованы и проанализированы последовательная и параллельная реализации стандартного алгоритма умножения матриц.

В рамках выполнения работы решены следующие задачи.

1. Изучены основы многопоточного программирования.
2. Придуманы два варианта многопоточной реализации стандартного алгоритма умножения матриц: обход по строкам и обход по столбцам.
3. Программно реализован стандартный алгоритм умножения матриц в однопоточном режиме.
4. Сделан сравнительный анализ по затрачиваемым ресурсам (времени) компьютера на реализацию каждого рассмотренного алгоритма.

Литература

[1] Visual Studio [Электронный ресурс], режим доступа:https://visualstudio.microsoft.com/ru/ (дата обращения: 01.10.2020)

[2] Thread.Join Метод [Электронный ресурс], режим доступа:https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.thread.join?view=netcore-3.1 (дата обращения: 01.10.2020)

[3] Документация по C# [Электронный ресурс], режим доступа:https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 02.10.2020)

[4] Академия Microsoft: Основы параллельного программирования с использованием Visual Studio [Электронный ресурс], - режим доступа: https://intuit.ru/studies/courses/4807/1055/lecture/16369 (дата обращения: 02.10.2020)

[5] Random Класс [Электронный ресурс], режим доступа:https://docs.microsoft.com/ru-ru/dotnet/api/system.random?view=netcore-3.1 (дата обращения: 01.10.2020)