Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 5

Многопоточность

Бутолин Александр Алексеевич
Студент группы ИУ7-52Б

2018 г.

Введение

В связи с возрастающей потребностью решать задачи, связанные с обработкой матриц, такие как расчет новых координат тела в пространстве, растет необходимость в эффективных алгоритмах по работе с ними. Перемножение матриц - одна из стандартных и наиболее используемых операций над матрицами, поэтому существует несколько алгоритмов, позволяющих произвести подобные вычисления (в данной работе рассматриваются классический алгоритм и алгоритм Винограда). Важную роль в решении подобных задач играет скорость, с которой производятся вычисления. Одним из способов улучшить данный показатель является многопоточность. В данной работе требуется на примере алгоритмов перемножения матриц изучить и применить на практике метод параллельных вычислений.

Цель работы: изучить алгоритмы умножения матриц (классический и Винограда), а также провести сравнительный анализ с реализацией на потоках. Задачи:

1) реализовать ПО, включающее данные алгоритмы;
2) провести замеры времени выполнения алгоритмов;
3) описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

1. Аналитический раздел

В этом разделе описаны алгоритмы, использованные в данной лабораторной работе.

1.1. Описание алгоритмов

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется их произведением [?]. Рассмотрим стандартный алгоритм перемножения двух матриц. Пусть есть две матрицы A и B размера $a \cdot b$ и $c \cdot d$ соответственно. Тогда, результатом из умножения будет матрица C размером $a \cdot d$, имеющая вид (??):

$$C = \begin{bmatrix} c_1 1 & c_1 2 & ... & c_1 d \\ c_2 1 & c_2 2 & ... & c_2 d \\ ... & & & \\ c_a 1 & c_a 2 & ... & c_a d \end{bmatrix}$$

Каждый элемент матрицы (??) представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Такое выражение можно просчитать заранее [?]. Рассмотрим два вектора:

$$V = (v_1, v_2, v_3, v_4) \tag{1}$$

и

$$W = (w_1, w_2, w_3, w_4) \tag{2}$$

Их скалярное произведение:

$$V * W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4. \tag{3}$$

Это равенство можно переписать в виде:

$$V * W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{4}$$

Несмотря на то, что выражение (??) требует больше вычисления, чем (??), выражение в правой части последнего равенства (??) допускает предварительную обработку[?]. Части этого выражения можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. В этом и заключается алгоритм Винограда [?].

В настоящее время умножение матриц активно применяется при решения задач [?]:

1) касающихся машинного обучения;

2) преобразования координат тела на плоскости или в пространстве.

1.2. Вывод

Исследование предметной области показало, что существует несколько возможных алгоритмов перемножения матриц. Это дает возможность произвести их сравнение для определения их преимуществ и недостатков. Данная задача актуальна, в связи с тем, что операции над матрицами находят обширное применение.

2. Конструкторский раздел

Для реализации алгоритмов были проведены следующие действия.

2.1. Модель вычислений

Введем модель вычислений, используемую при оценки трудоемкости:

1) объявление переменной, массива без определения имеет трудоемкость 0;
2) операторы <знак>=, а также ++ и – имеют трудоемкость 1;
3) условный оператор имеет трудоемкость 0
   а само условие расчитывается по операциям;
4) операции =, +, -, *, / имеют трудоемкость 1;
5) операця индескации имеет трудоемкость 1;
6) оператор цикла имеет трудоемкость 2 + n * (2 + тело цикла)
   для цикла вида for (int i = 0; i < n; i++).

2.2. Схемы алгоритмов

На рисунках ?? и ?? представлены схемы реализуемых алгоритмов, а так же рассчеты трудоемкости каждого из них.

2.3. Сравнительный анализ стандартного алгоритма перемножения матриц и алгоритма Винограда

Как видно из Рис. ?? алгоритм Винограда можно оптимизировать, так как некоторые операции можно вычислять заранее. Если же сравнивать с классическим алгоритмом, можно сделать предположение, что алгоритм Винограда будет работать медленнее, чем стандартный, из-за наличия трех дополнительных циклов. При этом, алгоритм Винограда использует дополнительные массивы, что должно увеличить объем потребляемой памяти.

По представленным схемам (раздел ??) можно произвести рассчеты трудоемкости. Таким образом, трудоемкость классического алгоритма равна:

$$T1 = 2 + n \cdot (2 + l \cdot (2 + 2 + m \cdot (2 + 8))) \approx 10 \cdot n \cdot m \cdot l \tag{5}$$

Трудоемкость алгоритма Винограда равна:

$$\begin{gathered} T2 = 2 + n \cdot (2 + 2 + (m/2) \cdot (2 + 12)) + 2 + l \cdot (2 + 2 + (m/2) \cdot (2 + 12)) + \\ 2 + n \cdot (2 + 2 + l \cdot (2 + 7 + 2 + (m/2) \cdot (2 + 23))) + 3 + 2 + \\ n \cdot (2 + 2 + l \cdot (2 + 13)) \approx 12.5 \cdot n \cdot m \cdot l \end{gathered} \tag{6}$$

Трудоемкость оптимизированного алгоритма Винограда равна:

$$\begin{gathered} T3 = 2 + (m/2) \cdot (2 + 2 + 2 + n \cdot (2 + 8) + 2 + l \cdot (2 + 8)) + \\ 2 + n \cdot (2 + 2 + l \cdot (2 + 7 + 2 + (m/2) \cdot (2 + 2 + 16))) + \\ 3 + 2 + 2 + n \cdot (2 + 2 + l \cdot (2 + 10)) \approx 10 \cdot n \cdot m \cdot l \end{gathered} \tag{7}$$

2.4. Вывод

Показанные в конструкторском разделе схемы позволяют реализовать представленные ими алгоритмы на любом языке программирования. Также, проведенный сравнительный анализ двух алгоритмов умножения дал возможность предположить, что алгоритм Винограда будет работать медленнее, потребляя при этом больше памяти, чем классический алгоритм. Была рассчитана трудоемкость каждого алгоритма.

3. Технологический раздел

В данном разделе приведена информация о конкретной реализации приведенных выше алгоритмов, а также исходный код полученных методов.

3.1. Требования к программному обеспечению

К программному обеспечению предъявлены следующие требования:

1) возможность ввода размерностей матриц;

2) возможность вывода результатов алгоритмов;

3) возможность вывода замеров времени, затраченного на работу алгоритмов.

3.2. Средства реализации

Лабораторная работа была выполнена в MonoDevelop на языке C #. Замеры процессорного времени были произведены с помощью встроенного метода TotalProcessorTime с последующим переводом в тики.

3.3. Листинг кода

Ниже приведены листинги реализованных методов.

Листинг 1: Реализация классического алгоритма умножения матриц

```
using System;
namespace lab_05
{
    public class Mult
    {
        int[,] mtr1, mtr2;
        int[,] result;
        int n, m, l;
        int start, finish;
        int[] row_fact, col_fact;
        int coef;
        public Mult(int n, int m, int l)
        {
            this.n = n;
            this.m = m;
            this.l = l;
            mtr1 = new int[n, m];
            mtr2 = new int[m, l];
            row_fact = new int[n];
            col_fact = new int[l];
            result = new int[n, l];
            coef = m / 2;
            result = new int[n, l];

            Fill_mtr();
        }
```

```
27          public Mult(int n, int m, int l, int[,] mtr1, int[,] mtr2)
28          {
29              this.n = n;
30              this.m = m;
31              this.l = l;
32              this.mtr1 = mtr1;
33              this.mtr2 = mtr2;
34              result = new int[n, l];
35          }
36
37          public Mult(int n, int m, int l, int[,] mtr1, int[,] mtr2, int
                coef, int[] row_fact, int[] col_fact)
38          {
39              this.n = n;
40              this.m = m;
41              this.l = l;
42              this.mtr1 = mtr1;
43              this.mtr2 = mtr2;
44              this.row_fact = row_fact;
45              this.col_fact = col_fact;
46              result = new int[n, l];
47              this.coef = coef;
48          }
49
50          public void Print_res()
51          {
52              Console.WriteLine("                                        ");
53              for (int i = 0; i < n; i++)
54              {
55                  for (int j = 0; j < l; j++)
56                  {
57                      Console.Write(result[i, j]);
58                      Console.Write(' ');
59                  }
60                  Console.WriteLine();
61              }
62          }
63
64          public void Print_mtr1()
65          {
66              Console.WriteLine("                1");
67              for (int i = 0; i < n; i++)
68              {
69                  for (int j = 0; j < m; j++)
70                  {
71                      Console.Write(mtr1[i, j]);
72                      Console.Write(' ');
73                  }
74                  Console.WriteLine();
75              }
76          }
```

```
77
78          public void Print_mtr2()
79          {
80              Console.WriteLine("                2");
81              for (int i = 0; i < m; i++)
82              {
83                  for (int j = 0; j < l; j++)
84                  {
85                      Console.Write(mtr2[i, j]);
86                      Console.Write(' ');
87                  }
88                  Console.WriteLine();
89              }
90          }
91
92          public void Fill_mtr()
93          {
94              Random rnd = new Random();
95              for (int i = 0; i < n; i++)
96                  for (int j = 0; j < m; j++)
97                      mtr1[i, j] = rnd.Next(0, 9);
98
99              for (int i = 0; i < m; i++)
100                 for (int j = 0; j < l; j++)
101                     mtr2[i, j] = rnd.Next(0, 9);
102         }
103
104         public void Fill_factors()
105         {
106             for (int i = 0; i < n; i++)
107             {
108                 for (int j = 0; j < coef; j++)
109                     row_fact[i] += mtr1[i, 2 * j + 1] * mtr1[i, 2 * j];
110             }
111
112             for (int i = 0; i < l; i++)
113             {
114                 for (int j = 0; j < coef; j++)
115                     col_fact[i] += mtr2[2 * j + 1, i] * mtr2[2 * j, i];
116             }
117         }
118
119         public void Multiply()
120         {
121             for (int i = start; i < finish; i++)
122                 for (int j = 0; j < l; j++)
123                 {
124                     result[i, j] = 0;
125                     for (int k = 0; k < m; k++)
126                         result[i, j] += mtr1[i, k] * mtr2[k, j];
127                 }
```

```
128         }
129
130         public void Standart_alg()
131         {
132             for (int i = 0; i < n; i++)
133                 for (int j = 0; j < l; j++)
134                 {
135                     result[i, j] = 0;
136                     for (int k = 0; k < m; k++)
137                         result[i, j] += mtr1[i, k] * mtr2[k, j];
138                 }
139         }
140
141         public void Vinograd_mult()
142         {
143             for (int i = start; i < finish; i++)
144             {
145                 for (int j = 0; j < l; j++)
146                 {
147                     result[i, j] = -row_fact[i] - col_fact[j];
148                     for (int r = 0; r < coef; r++)
149                     {
150                         result[i, j] = result[i, j] + (mtr1[i, 2 * r] +
                                mtr2[2 * r + 1, j]) * (mtr1[i, 2 * r + 1] +
                                mtr2[2 * r, j]);
151                     }
152                 }
153             }
154         }
155
156         public void Standart_vinograd_mult()
157         {
158             for (int i = 0; i < n; i++)
159             {
160                 for (int j = 0; j < coef; j++)
161                     row_fact[i] += mtr1[i, 2 * j + 1] * mtr1[i, 2 * j];
162             }
163
164             for (int i = 0; i < l; i++)
165             {
166                 for (int j = 0; j < coef; j++)
167                     col_fact[i] += mtr2[2 * j + 1, i] * mtr2[2 * j, i];
168             }
169
170             for (int i = 0; i < n; i++)
171             {
172                 for (int j = 0; j < l; j++)
173                 {
174                     result[i, j] = -row_fact[i] - col_fact[j];
175                     for (int r = 0; r < coef; r++)
176                     {
```

```
177                         result[i, j] = result[i, j] + (mtr1[i, 2 * r] +
                               mtr2[2 * r + 1, j]) * (mtr1[i, 2 * r + 1] +
                               mtr2[2 * r, j]);
178                     }
179                 }
180             }
181
182             if (m % 2 != 0)
183             {
184                 for (int i = 0; i < n; i++)
185                 {
186                     for (int j = 0; j < l; j++)
187                     {
188                         result[i, j] += mtr1[i, m - 1] * mtr2[m - 1, j];
189                     }
190                 }
191             }
192         }
193
194         public void Set_bounds(int start, int finish)
195         {
196             this.start = start;
197             this.finish = finish;
198         }
199
200         public void Copy(ref int[,] res)
201         {
202             for (int i = start; i < finish; i++)
203                 for (int j = 0; j < l; j++)
204                     res[i, j] = result[i, j];
205         }
206
207         public void Dump_res()
208         {
209             for (int i = 0; i < n; i++)
210             {
211                 for (int j = 0; j < l; j++)
212                     result[i, j] = 0;
213             }
214         }
215
216     }
217 }
```

Листинг 2: Реализация алгоритма Винограда перемножения матриц

```
using System;
namespace lab_05
{
    public class Mult
    {
        int[,] mtr1, mtr2;
        int[,] result;
        int n, m, l;
        int start, finish;
        int[] row_fact, col_fact;
        int coef;
        public Mult(int n, int m, int l)
        {
            this.n = n;
            this.m = m;
            this.l = l;
            mtr1 = new int[n, m];
            mtr2 = new int[m, l];
            row_fact = new int[n];
            col_fact = new int[l];
            result = new int[n, l];
            coef = m / 2;
            result = new int[n, l];

            Fill_mtr();
        }
        public Mult(int n, int m, int l, int[,] mtr1, int[,] mtr2)
        {
            this.n = n;
            this.m = m;
            this.l = l;
            this.mtr1 = mtr1;
            this.mtr2 = mtr2;
            result = new int[n, l];
        }

        public Mult(int n, int m, int l, int[,] mtr1, int[,] mtr2, int
           coef, int[] row_fact, int[] col_fact)
        {
            this.n = n;
            this.m = m;
            this.l = l;
            this.mtr1 = mtr1;
            this.mtr2 = mtr2;
            this.row_fact = row_fact;
            this.col_fact = col_fact;
            result = new int[n, l];
            this.coef = coef;
        }

```

```
50          public void Print_res()
51          {
52              Console.WriteLine("                                                  ");
53              for (int i = 0; i < n; i++)
54              {
55                  for (int j = 0; j < l; j++)
56                  {
57                      Console.Write(result[i, j]);
58                      Console.Write(' ');
59                  }
60                  Console.WriteLine();
61              }
62          }
63
64          public void Print_mtr1()
65          {
66              Console.WriteLine("                1");
67              for (int i = 0; i < n; i++)
68              {
69                  for (int j = 0; j < m; j++)
70                  {
71                      Console.Write(mtr1[i, j]);
72                      Console.Write(' ');
73                  }
74                  Console.WriteLine();
75              }
76          }
77
78          public void Print_mtr2()
79          {
80              Console.WriteLine("                2");
81              for (int i = 0; i < m; i++)
82              {
83                  for (int j = 0; j < l; j++)
84                  {
85                      Console.Write(mtr2[i, j]);
86                      Console.Write(' ');
87                  }
88                  Console.WriteLine();
89              }
90          }
91
92          public void Fill_mtr()
93          {
94              Random rnd = new Random();
95              for (int i = 0; i < n; i++)
96                  for (int j = 0; j < m; j++)
97                      mtr1[i, j] = rnd.Next(0, 9);
98
99              for (int i = 0; i < m; i++)
100                 for (int j = 0; j < l; j++)
```

```
101                     mtr2[i, j] = rnd.Next(0, 9);
102             }
103
104             public void Fill_factors()
105             {
106                 for (int i = 0; i < n; i++)
107                 {
108                     for (int j = 0; j < coef; j++)
109                         row_fact[i] += mtr1[i, 2 * j + 1] * mtr1[i, 2 * j];
110                 }
111
112                 for (int i = 0; i < l; i++)
113                 {
114                     for (int j = 0; j < coef; j++)
115                         col_fact[i] += mtr2[2 * j + 1, i] * mtr2[2 * j, i];
116                 }
117             }
118
119             public void Multiply()
120             {
121                 for (int i = start; i < finish; i++)
122                     for (int j = 0; j < l; j++)
123                     {
124                         result[i, j] = 0;
125                         for (int k = 0; k < m; k++)
126                             result[i, j] += mtr1[i, k] * mtr2[k, j];
127                     }
128             }
129
130             public void Standart_alg()
131             {
132                 for (int i = 0; i < n; i++)
133                     for (int j = 0; j < l; j++)
134                     {
135                         result[i, j] = 0;
136                         for (int k = 0; k < m; k++)
137                             result[i, j] += mtr1[i, k] * mtr2[k, j];
138                     }
139             }
140
141             public void Vinograd_mult()
142             {
143                 for (int i = start; i < finish; i++)
144                 {
145                     for (int j = 0; j < l; j++)
146                     {
147                         result[i, j] = -row_fact[i] - col_fact[j];
148                         for (int r = 0; r < coef; r++)
149                         {
150                             result[i, j] = result[i, j] + (mtr1[i, 2 * r] +
                                    mtr2[2 * r + 1, j]) * (mtr1[i, 2 * r + 1] +
```

```
                        mtr2[2 * r, j]);
                }
            }
        }
    }

    public void Standart_vinograd_mult()
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < coef; j++)
                row_fact[i] += mtr1[i, 2 * j + 1] * mtr1[i, 2 * j];
        }

        for (int i = 0; i < l; i++)
        {
            for (int j = 0; j < coef; j++)
                col_fact[i] += mtr2[2 * j + 1, i] * mtr2[2 * j, i];
        }

        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < l; j++)
            {
                result[i, j] = -row_fact[i] - col_fact[j];
                for (int r = 0; r < coef; r++)
                {
                    result[i, j] = result[i, j] + (mtr1[i, 2 * r] +
                       mtr2[2 * r + 1, j]) * (mtr1[i, 2 * r + 1] +
                       mtr2[2 * r, j]);
                }
            }
        }

        if (m % 2 != 0)
        {
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < l; j++)
                {
                    result[i, j] += mtr1[i, m - 1] * mtr2[m - 1, j];
                }
            }
        }
    }

    public void Set_bounds(int start, int finish)
    {
        this.start = start;
        this.finish = finish;
    }
```

```
199
200         public void Copy(ref int[,] res)
201         {
202             for (int i = start; i < finish; i++)
203                 for (int j = 0; j < l; j++)
204                     res[i, j] = result[i, j];
205         }
206
207         public void Dump_res()
208         {
209             for (int i = 0; i < n; i++)
210             {
211                 for (int j = 0; j < l; j++)
212                     result[i, j] = 0;
213             }
214         }
215
216     }
217 }
```

Листинг 3: Реализация классического алгоритма умножения с использованием параллельных вычислений

```
using System;
using System.Threading;
using System.Collections.Concurrent;

namespace lab_05
{
    public class Threading
    {
        int[,] mtr1, mtr2, result;
        int n, m, l;

        public Threading(int n, int m, int l)
        {
            this.n = n;
            this.m = m;
            this.l = l;
            mtr1 = new int[n, m];
            mtr2 = new int[m, l];
            result = new int[n, l];

            Fill_mtr();
        }

        public void Print_res()
        {
            Console.WriteLine("                                          ");
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < l; j++)
                {
                    Console.Write(result[i, j]);
                    Console.Write(' ');
                }
                Console.WriteLine();
            }
        }

        public void Print_mtr1()
        {
            Console.WriteLine("               1");
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < m; j++)
                {
                    Console.Write(mtr1[i, j]);
                    Console.Write(' ');
                }
                Console.WriteLine();
            }
```

```
50          }
51
52          public void Print_mtr2()
53          {
54              Console.WriteLine("                2");
55              for (int i = 0; i < m; i++)
56              {
57                  for (int j = 0; j < l; j++)
58                  {
59                      Console.Write(mtr2[i, j]);
60                      Console.Write(' ');
61                  }
62                  Console.WriteLine();
63              }
64          }
65
66          public void Dump_res()
67          {
68              for (int i = 0; i < n; i++)
69              {
70                  for (int j = 0; j < l; j++)
71                      result[i, j] = 0;
72              }
73          }
74
75          void Fill_mtr()
76          {
77              Random rnd = new Random();
78              for (int i = 0; i < n; i++)
79                  for (int j = 0; j < m; j++)
80                      mtr1[i, j] = rnd.Next(0, 9);
81
82              for (int i = 0; i < m; i++)
83                  for (int j = 0; j < l; j++)
84                      mtr2[i, j] = rnd.Next(0, 9);
85          }
86
87          public void Thread_mult(int thread_am)
88          {
89              int partition = n / thread_am; //  3=6/2
90              int start = 0, finish = partition + (n - partition * thread_am
                    );//  3 = 3 + (6 - 3 * 2)
91              if (thread_am > n)
92              {
93                  partition = 1;
94                  finish = 1;
95              }
96              int count;
97
98              ConcurrentBag<Thread> thread_list = new ConcurrentBag<Thread
                    >();
```

```
99              ConcurrentBag<Mult> mul_list = new ConcurrentBag<Mult>();
100
101             for (int k = 0; k < thread_am; k++)
102             {
103                 Mult mult = new Mult(n, m, l, mtr1, mtr2);
104                 mult.Set_bounds(start, finish);
105                 Thread thread = new Thread(mult.Multiply);
106                 thread.Start();
107                 thread_list.Add(thread);
108                 mul_list.Add(mult);
109
110                 count = thread_list.Count;
111
112                 for (int i = 0; i < count; i++)
113                 {
114                     thread_list.TryTake(out Thread curr);
115                     curr.Join();
116                 }
117
118                 count = mul_list.Count;
119
120                 for (int i = 0; i < count; i++)
121                 {
122                     mul_list.TryTake(out Mult mul);
123                     mul.Copy(ref result);
124                 }
125
126                 start = finish;
127                 finish += partition;
128                 if (finish > n)
129                     finish = n;
130             }
131         }
132
133         public void Thread_vinograd(int thread_am)
134         {
135             int[] row_fact, col_fact;
136             int coef = m / 2;
137             int partition = n / thread_am;
138             int start = 0, finish = partition + (n - partition * thread_am
                    );
139             if (thread_am > n)
140             {
141                 partition = 1;
142                 finish = partition;
143             }
144
145             int count;
146             row_fact = new int[n];
147             col_fact = new int[l];
148
```

```
149             for (int i = 0; i < n; i++)
150             {
151                 for (int j = 0; j < coef; j++)
152                     row_fact[i] += mtr1[i, 2 * j + 1] * mtr1[i, 2 * j];
153             }
154 
155             for (int i = 0; i < l; i++)
156             {
157                 for (int j = 0; j < coef; j++)
158                     col_fact[i] = col_fact[i] + mtr2[2 * j + 1, i] * mtr2
                        [2 * j, i];
159             }
160 
161             ConcurrentBag<Thread> thread_list = new ConcurrentBag<Thread
                >();
162             ConcurrentBag<Mult> mul_list = new ConcurrentBag<Mult>();
163             for (int k = 0; k < thread_am; k++)
164             {
165                 Mult mult = new Mult(n, m, l, mtr1, mtr2, coef, row_fact,
                    col_fact);
166                 mult.Set_bounds(start, finish);
167                 Thread thread = new Thread(mult.Vinograd_mult);
168                 thread.Start();
169                 thread_list.Add(thread);
170                 mul_list.Add(mult);
171 
172                 count = thread_list.Count;
173 
174                 for (int i = 0; i < count; i++)
175                 {
176                     thread_list.TryTake(out Thread curr);
177                     curr.Join();
178                 }
179 
180                 count = mul_list.Count;
181 
182                 for (int i = 0; i < count; i++)
183                 {
184                     mul_list.TryTake(out Mult mul);
185                     mul.Copy(ref result);
186                 }
187 
188                 start = finish;
189                 finish += partition;
190                 if (finish > n)
191                     finish = n;
192             }
193 
194             if (m % 2 != 0)
195             {
196                 for (int i = 0; i < n; i++)
```

```
197                     {
198                         for (int j = 0; j < l; j++)
199                         {
200                             result[i, j] += mtr1[i, m - 1] * mtr2[m - 1, j];
201                         }
202                     }
203                 }
204             }
205         }
206 }
```

Листинг 4: Реализация алгоритма Винограда умножения с использованием параллельных вычислений

```
using System;
using System.Threading;
using System.Collections.Concurrent;

namespace lab_05
{
    public class Threading
    {
        int[,] mtr1, mtr2, result;
        int n, m, l;

        public Threading(int n, int m, int l)
        {
            this.n = n;
            this.m = m;
            this.l = l;
            mtr1 = new int[n, m];
            mtr2 = new int[m, l];
            result = new int[n, l];

            Fill_mtr();
        }

        public void Print_res()
        {
            Console.WriteLine("                                          ");
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < l; j++)
                {
                    Console.Write(result[i, j]);
                    Console.Write(' ');
                }
                Console.WriteLine();
            }
        }

        public void Print_mtr1()
        {
            Console.WriteLine("                1");
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < m; j++)
                {
                    Console.Write(mtr1[i, j]);
                    Console.Write(' ');
                }
                Console.WriteLine();
            }
```

```
50          }
51
52          public void Print_mtr2()
53          {
54              Console.WriteLine("                2");
55              for (int i = 0; i < m; i++)
56              {
57                  for (int j = 0; j < l; j++)
58                  {
59                      Console.Write(mtr2[i, j]);
60                      Console.Write(' ');
61                  }
62                  Console.WriteLine();
63              }
64          }
65
66          public void Dump_res()
67          {
68              for (int i = 0; i < n; i++)
69              {
70                  for (int j = 0; j < l; j++)
71                      result[i, j] = 0;
72              }
73          }
74
75          void Fill_mtr()
76          {
77              Random rnd = new Random();
78              for (int i = 0; i < n; i++)
79                  for (int j = 0; j < m; j++)
80                      mtr1[i, j] = rnd.Next(0, 9);
81
82              for (int i = 0; i < m; i++)
83                  for (int j = 0; j < l; j++)
84                      mtr2[i, j] = rnd.Next(0, 9);
85          }
86
87          public void Thread_mult(int thread_am)
88          {
89              int partition = n / thread_am; //  3=6/2
90              int start = 0, finish = partition + (n - partition * thread_am
                    );//  3 = 3 + (6 - 3 * 2)
91              if (thread_am > n)
92              {
93                  partition = 1;
94                  finish = 1;
95              }
96              int count;
97
98              ConcurrentBag<Thread> thread_list = new ConcurrentBag<Thread
                    >();
```

```
99              ConcurrentBag<Mult> mul_list = new ConcurrentBag<Mult>();
100
101             for (int k = 0; k < thread_am; k++)
102             {
103                 Mult mult = new Mult(n, m, l, mtr1, mtr2);
104                 mult.Set_bounds(start, finish);
105                 Thread thread = new Thread(mult.Multiply);
106                 thread.Start();
107                 thread_list.Add(thread);
108                 mul_list.Add(mult);
109
110                 count = thread_list.Count;
111
112                 for (int i = 0; i < count; i++)
113                 {
114                     thread_list.TryTake(out Thread curr);
115                     curr.Join();
116                 }
117
118                 count = mul_list.Count;
119
120                 for (int i = 0; i < count; i++)
121                 {
122                     mul_list.TryTake(out Mult mul);
123                     mul.Copy(ref result);
124                 }
125
126                 start = finish;
127                 finish += partition;
128                 if (finish > n)
129                     finish = n;
130             }
131         }
132
133         public void Thread_vinograd(int thread_am)
134         {
135             int[] row_fact, col_fact;
136             int coef = m / 2;
137             int partition = n / thread_am;
138             int start = 0, finish = partition + (n - partition * thread_am
                    );
139             if (thread_am > n)
140             {
141                 partition = 1;
142                 finish = partition;
143             }
144
145             int count;
146             row_fact = new int[n];
147             col_fact = new int[l];
148
```

```
149             for (int i = 0; i < n; i++)
150             {
151                 for (int j = 0; j < coef; j++)
152                     row_fact[i] += mtr1[i, 2 * j + 1] * mtr1[i, 2 * j];
153             }
154 
155             for (int i = 0; i < l; i++)
156             {
157                 for (int j = 0; j < coef; j++)
158                     col_fact[i] = col_fact[i] + mtr2[2 * j + 1, i] * mtr2
                        [2 * j, i];
159             }
160 
161             ConcurrentBag<Thread> thread_list = new ConcurrentBag<Thread
                >();
162             ConcurrentBag<Mult> mul_list = new ConcurrentBag<Mult>();
163             for (int k = 0; k < thread_am; k++)
164             {
165                 Mult mult = new Mult(n, m, l, mtr1, mtr2, coef, row_fact,
                    col_fact);
166                 mult.Set_bounds(start, finish);
167                 Thread thread = new Thread(mult.Vinograd_mult);
168                 thread.Start();
169                 thread_list.Add(thread);
170                 mul_list.Add(mult);
171 
172                 count = thread_list.Count;
173 
174                 for (int i = 0; i < count; i++)
175                 {
176                     thread_list.TryTake(out Thread curr);
177                     curr.Join();
178                 }
179 
180                 count = mul_list.Count;
181 
182                 for (int i = 0; i < count; i++)
183                 {
184                     mul_list.TryTake(out Mult mul);
185                     mul.Copy(ref result);
186                 }
187 
188                 start = finish;
189                 finish += partition;
190                 if (finish > n)
191                     finish = n;
192             }
193 
194             if (m % 2 != 0)
195             {
196                 for (int i = 0; i < n; i++)
```

```
197                     {
198                         for (int j = 0; j < l; j++)
199                         {
200                             result[i, j] += mtr1[i, m - 1] * mtr2[m - 1, j];
201                         }
202                     }
203                 }
204             }
205         }
206 }
```

3.4. Вывод

Основываясь на схемах, полученных в конструкторском разделе, было реализовано три алгоритма перемножения матриц. Программа была написана на языке C # в среде MonoDevelop. Были выполнены поставленные для ПО требования, а именно осуществлен ввод размерностей матриц и вывод результатов работы алгоритмов.

4. Экспериментальная часть

При реализации алгоритмов была произведена проверка правильности работы.

4.1. Примеры работы

Были проведены тесты всех алгоритмов для определения правильности их работы.
Тест 1. Умножение.
Матрицы для перемножения:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 10 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 4 & 0 \\ 5 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

Ожидаемый результат:

$$C = \begin{bmatrix} 12 & 7 & -5 \\ -2 & 6 & -10 \\ 0 & 0 & 0 \end{bmatrix}$$

Результаты, где (??) - классический алгоритм и (??) - алгоритм Винограда:

$$Classic = \begin{bmatrix} 12 & 7 & -5 \\ -2 & 6 & -10 \\ 0 & 0 & 0 \end{bmatrix} \quad (8) \qquad Winogr = \begin{bmatrix} 12 & 7 & -5 \\ -2 & 6 & -10 \\ 0 & 0 & 0 \end{bmatrix} \quad (9)$$

Тест 2. Умножение на единичную матрицу.
Матрицы для перемножения:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 4 \\ 5 & 0 \end{bmatrix}$$

Ожидаемый результат:

$$C = \begin{bmatrix} 2 & 4 \\ 5 & 0 \end{bmatrix}$$

Результаты, где (??) - классический алгоритм и (??) - алгоритм Винограда:

$$Classic = \begin{bmatrix} 2 & 4 \\ 5 & 0 \end{bmatrix} \quad (10) \qquad Winogr = \begin{bmatrix} 2 & 4 \\ 5 & 0 \end{bmatrix} \quad (11)$$

Тест 3. Умножение на нулевую матрицу.
Матрицы для перемножения:

$$A = \begin{bmatrix} -1 & 10 \\ 2 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Ожидаемый результат:

$$C = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Результаты, где (??) - классический алгоритм и (??) - алгоритм Винограда:

$$Classic = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (12) \qquad Winogr = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (13)$$

Тест 4. Умножение матриц 1x1.
Матрицы для перемножения:

$$A = \begin{bmatrix} -1 \end{bmatrix} \qquad B = \begin{bmatrix} 2 \end{bmatrix}$$

Ожидаемый результат:

$$C = \begin{bmatrix} -2 \end{bmatrix}$$

$$Classic = \begin{bmatrix} -2 \end{bmatrix} \quad (14) \qquad Winogr = \begin{bmatrix} -2 \end{bmatrix} \quad (15)$$

Результаты, где (??) - классический алгоритм и (??) - алгоритм Винограда:

4.2. Постановка эксперимента

Замеры времени производились на различной нечетной размерности матрицы от 11 до 1011 с шагом 100 для разного числа потоков (1, 2, 4, 8, 16). Замеры процессорного времени были произведены с помощью встроенного метода TotalProcessorTime с последующим переводом в тики.

4.3. Сравнительный анализ на основе экспериментальных данных

По произведенным замерам были построены следующие графики.

Были также произведены замеры времени выполнения многопоточных реализаций, для определения зависимости времени выполнения от числа потоков. Выявленная зависимость представлена на рисунке

4.4. Вывод

Произведенные эксперименты показали, что параллельные вычисления в реализации классического алгоритма улучшают показатели, в то время как в реализации алгоритма Винограда ухудшают их, предположительно из-за времени создания объектов, копирования исходных данных и выполнения последнего цикла в основном потоке после завершения всех потоков. Также был произведен анализ зависимости времени выполнения методов от числа потоков.

Заключение

В ходе лабораторной работы были реализованы два алгоритма перемножения матриц: классический и алгоритм Винограда. Для обоих алгоритмов была предложена оптимизация, подразумевающая параллельные вычисления. Был произведен анализ временных показателей для всех реализаций. Результаты данного анализа, произведенного в экспериментальной части, показали, что параллельные вычисления улучшают работу классического алгоритма, в то время как распараллеленный алгоритм Винограда не улучшается в данной реализации, так как для каждого потока происходит копирование всех исходных данных, а последний цикл, предусмотренный алгоритмом для корректировки результата на нечетных размерностях матриц не модет быть распараллелен. Копирования элементов можно избежать, используя указатели, но это делает код менее безопасным и требует контроля памяти, так как автоматический сборщик мусора не контролирует указатели. Также было показано, что при совпадении числа потоков и числа логических ядер методы показывают самые лучшие результаты, после чего их показатели начинают ухудшаться.

Список литературы

[1] Coppersmith and Shmuel Winograd. «Journal of Symbolic Computation» - M.: Доклады Академий Наук СССР, 1965.

[2] «Алгоритм Копперсмита — Винограда» [Электронный ресурс]. Journal of Symbolic Computation. – Режим доступа: http://ru.math.wikia.com/wiki/Алгоритм Копперсмита — Винограда, свободный.

[3] Корн Г., Корн Т. Алгебра матриц и матричное исчисление // Справочник по математике. — 4-е издание. — М: Наука, 1978.

[4] ETANIT.COM: Сайт о программировании. [Электронный ресурс]. URL:https://metanit.com/sharp/tutorial/11.1.php (дата обращения: 012.02.2019)