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»

Отчет
по лабораторной работе № 4

Дисциплина: Анализ алгоритмов

Тема: Алгоритмы параллельного умножения матриц

Студент: Платонова Ольга

Группа: ИУ7-55Б

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

Москва, 2020 г.

Оглавление

Введение

В данной лабораторной работе требуется изучить возможность параллельных вычислений и на их основе реализовать алгоритмы умножения матриц. Так, требуется реализовать две распараллеленные версии алгоритма Винограда и выполнить сравнительный анализ по времени для линейной и распараллеленных реализаций.

1. Аналитическая часть

В данном разделе рассмотрим понятия и формулы, связанные с параллельными вычислениями и умножением матриц с помощью алгоритма Винограда.

1.1. Умножение матриц

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_{ij}, равен сумме произведений элементов i-ой строки матрицы *A* на соответствующие элементы j-ого столбца матрицы *B*.

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \tag{1.1}$$

Замечание

1. Операция умножения двух матриц выполнима тогда и только тогда, когда число столбцов в первом сомножителе равно числу строк во втором; в этом случае говорят, что матрицы *согласованы*.
2. Из факта существования произведения *AB* не следует существование произведения *BA*.[1]

1.2. Алгоритм Винограда

Из определения произведения матриц следует, что каждый элемент результирующей матрицы *C* представляет собой скалярное произведение строки на соответствующий столбец. Работа алгоритма Винограда основана на вычислении двух векторов, скалярное произведение которых позволяет сформировать результирующую матрицу. Рассмотрим эти вектора:

$$V = (v1, v2, v3, v4)$$
$$W = (w1, w2, w3, w4) \tag{1.2}$$

Их скалярное произведение равно:

$$V \bullet W = v1w1 + v2w2 + v3w3 + v4w4 \tag{1.3}$$

Это равенство можно переписать в виде:

$$V \cdot W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4 \quad (1.4)$$

Стоит отметить, что полученное равенство может быть предварительно вычислено. Для этого требуются вектора, которые являются результатом умножения элементов каждой строки первой матрицы, и столбца второй. [2]

1.3. Параллельные вычисления

Параллельные вычисления — способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно).

Существуют различные способы реализации параллельных вычислений. Например, каждый вычислительный процесс может быть реализован в виде процесса операционной системы, либо же вычислительные процессы могут представлять собой набор потоков выполнения внутри одного процесса ОС. Параллельные программы могут физически исполняться либо последовательно на единственном процессоре — перемежая по очереди шаги выполнения каждого вычислительного процесса, либо параллельно — выделяя каждому вычислительному процессу один или несколько процессоров.

Организация параллельного программирования в данной работе будет основываться на создании нескольких потоков.

Поток, нить (thread) – создаваемый операционной системой объект внутри процесса, который выполняет инструкции программы. Процесс может иметь один или несколько потоков, выполняемых в контексте данного процесса. Потоки (нити) позволяют осуществлять параллельное выполнение процессов и одновременное выполнение одним процессом различных частей программы на различных процессорах. [3]

1.4. Вывод

В данном разделе был рассмотрен алгоритм Винограда умножения матриц и разобраны понятия, связанные с параллельным вычислением, требуемые для организации распараллеленных версий алгоритма Винограда.

2. Конструкторская часть

В данном разделе будут рассмотрены схемы для каждой из реализаций алгоритма умножения матриц Винограда.

2.1. Разработка алгоритмов

На рисунке 1 изображена схема *последовательного алгоритма Винограда* умножения матриц.

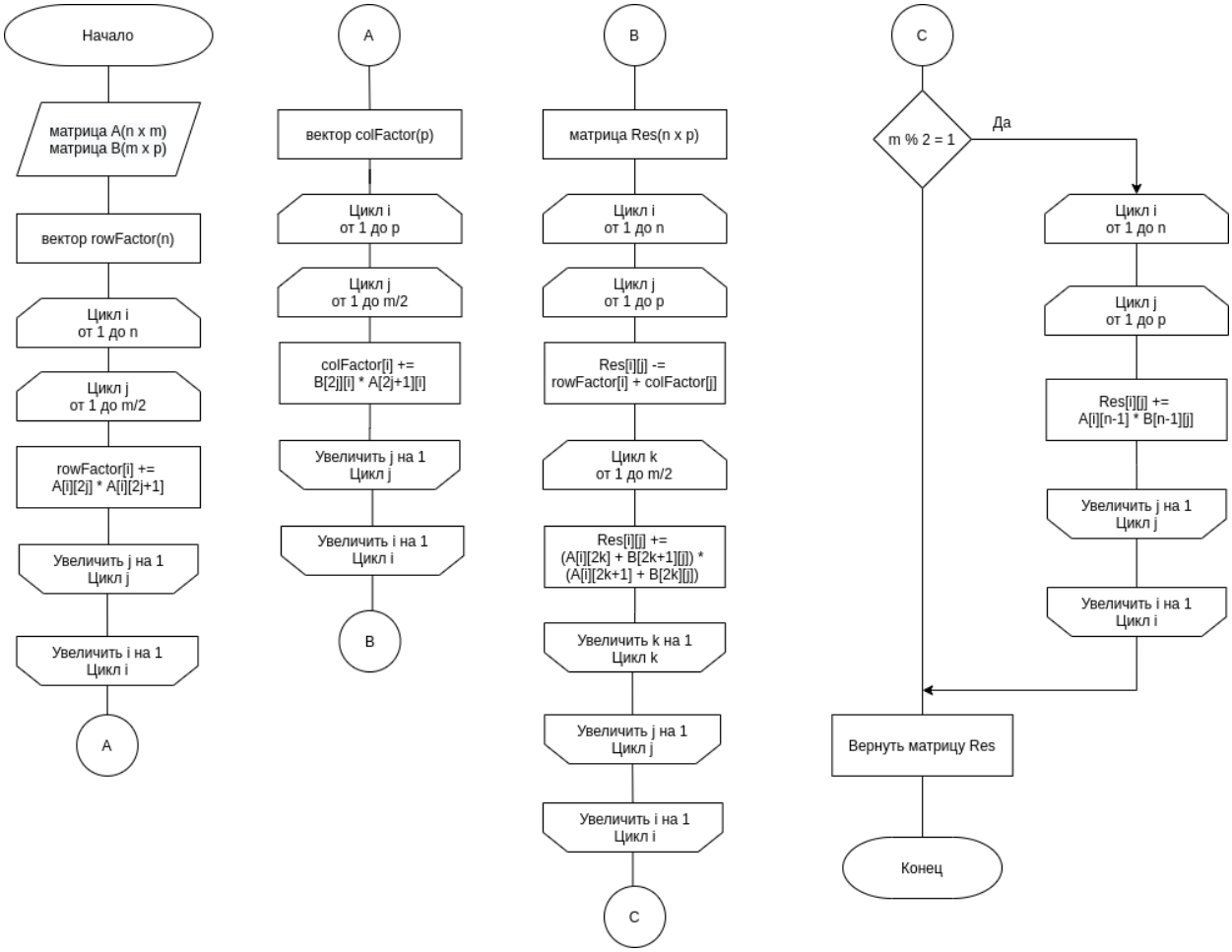

Рисунок 1. Схема последовательного алгоритма Винограда.

2.2. Схемы параллельных вычислений

С целью изучения параллельных вычислений следует рассмотреть следующие реализации алгоритма Винограда.

1. Распараллеливание участка кода, отвечающего за заполнение двух векторов rowFactor и colFactor.

   Поскольку вычисление для каждой из строк первого (соответственно, столбцов второго) вектора не зависит от результатов вычислений для других строк (столбцов), применение параллельных вычислений для данного этапа алгоритма будет корректным. Таким образом, предлагается распараллелить участок кода, соответствующий участкам Начало-A и A-B. (рисунок 1)

2. Распараллеливание участка кода, отвечающего за заполнение результирующей матрицы.

   Аналогично пункту 1, вычисление элемента результирующей матрицы для каждой строки не зависит от результатов вычислений для других строк. Следовательно, разбиение строк на равные подмножества для каждого из потоков будет корректным. Таким образом, предлагается распараллелить участок кода, соответствующий участку B-C. (рисунок 1)

2.3. Вывод

В данном разделе были рассмотрены 3 схемы алгоритма Винограда: одна последовательная и две распараллеленные. Поскольку наибольшую сложность составляет заполнение результирующей матрицы с использованием тройного вложенного цикла, максимальная эффективность ожидается от второго способа распараллеливания.

3. Технологическая часть

В данном разделе рассмотрим язык программирования, среду разработки, требуемые инструменты для реализации. Также представим непосредственно реализацию.

3.1. Требования к программному обеспечению

I. Программа должна предусматривать ввод двух матриц указанной размерности. Заполнение матриц должно быть доступно как с клавиатуры, так и автоматически.
II. Выбор применяемого алгоритма осуществляется пользователем из списка алгоритмов, предложенных в меню.
III. На выходе программа выводит результирующую матрицу.
IV. Также необходимо предусмотреть выполнение замеров процессорного времени для каждой из реализаций.

3.2. Средства реализации

В данной работе используется язык программирования C++, из-за удобства представления матрицы в виде объекта класса (сокращает количество аргументов для передачи в функцию) и опыта написания на нем. Среда разработки – Qt.

Для замеров процессорного времени использовалась функция clock(). [4]

3.3. Реализация алгоритмов

Реализация последовательной версии алгоритма Винограда представлена в листинге 1.

Листинг 1. Алгоритм умножения матриц Винограда. Последовательная реализация.

```
#include "vinogradMul.h"

Matrix vinogradMul(Matrix matrA, Matrix matrB)
{
   int rowA = matrA.numRows;
   int sizeAB = matrA.numColumns;
   int colB = matrB.numColumns;

   int numPairs = sizeAB / 2;

   //Произведение элементов в каждой строке matrA
   int *rowFactor = createVector(rowA);
   for (int i = 0; i < rowA; i++) {
```

```
    for (int j = 0; j < numPairs; j++) {
        rowFactor[i] += matrA.ptr[i][2 * j] * matrA.ptr[i][2 * j + 1];
    }
  }

  //Произведение элементов в каждом столбце matrB
  int *colFactor = createVector(colB);
  for (int i = 0; i < colB; i++) {
    for (int j = 0; j < numPairs; j++) {
        colFactor[i] += matrB.ptr[2 * j][i] * matrB.ptr[2 * j + 1][i];
    }
  }

  //Вычисление произведения
  Matrix matrRes(rowA, colB);
  for (int i = 0; i < rowA; i++) {
    for (int j = 0; j < colB; j++) {
        matrRes.ptr[i][j] -= rowFactor[i] + colFactor[j];
        for (int k = 0; k < numPairs; k++) {
            matrRes.ptr[i][j] += (matrA.ptr[i][2 * k] + matrB.ptr[2 * k + 1][j]) *
                                 (matrA.ptr[i][2 * k + 1] + matrB.ptr[2 * k][j]);
        }
    }
  }

  //Случай нечетности
  if (sizeAB % 2 != 0) {
    for (int i = 0; i < rowA; i++) {
        for (int j = 0; j < colB; j++) {
            matrRes.ptr[i][j] += matrA.ptr[i][sizeAB - 1] * matrB.ptr[sizeAB - 1][j];
        }
    }
  }
  free(colFactor);
  free(rowFactor);
  return matrRes;
}
```

Реализация параллельных версий алгоритма Винограда представлена в листингах 2 – 3.

Листинг 2. Алгоритм Винограда распараллеливания заполнения векторов.

```
#include <thread>
#include "parallelVectors.h"

Matrix parallelVectors(Matrix matrA, Matrix matrB, int numThreads)
{
  int rowA = matrA.numRows;
  int sizeAB = matrA.numColumns;
  int colB = matrB.numColumns;

  int numPairs = sizeAB / 2;

  //Создание потоков
  thread* threads = new thread[numThreads];

  //Произведение элементов в каждой строке matrA
  int *rowFactor = createVector(rowA);
  int rowsPerThred = rowA / numThreads;
```

```
int beginRow = 0;
for (int i = 0; i < numThreads; i++)
{
   int endRow = beginRow + rowsPerThred;
   if (i == numThreads - 1)
   {
      endRow = rowA;
   }

   threads[i] = thread(threadVectorRow, matrA, rowFactor, beginRow, endRow);
   beginRow = endRow;
}

for (int i = 0; i < numThreads; i++)
{
   threads[i].join();
}

//Произведение элементов в каждом столбце matrB
int *colFactor = createVector(colB);
int colsPerThred = colB / numThreads;
int beginCol = 0;
for (int i = 0; i < numThreads; i++)
{
   int endCol = beginCol + colsPerThred;
   if (i == numThreads - 1)
   {
      endCol = colB;
   }

   threads[i] = thread(threadVectorCol, matrB, colFactor, beginCol, endCol);
   beginCol = endCol;
}

for (int i = 0; i < numThreads; i++)
{
   threads[i].join();
}

//Вычисление произведения
Matrix matrRes(rowA, colB);
for (int i = 0; i < rowA; i++)
{
   for (int j = 0; j < colB; j++)
   {
      matrRes.ptr[i][j] = - rowFactor[i] - colFactor[j];
      for (int k = 0; k < numPairs; k++)
      {
         matrRes.ptr[i][j] += (matrA.ptr[i][2 * k] + matrB.ptr[2 * k + 1][j]) *
                              (matrA.ptr[i][2 * k + 1] + matrB.ptr[2 * k][j]);
      }
   }
}


//Случай нечетности
  if (sizeAB % 2 != 0)
  {
     for (int i = 0; i < rowA; i++)
     {
        for (int j = 0; j < colB; j++)
        {
           matrRes.ptr[i][j] += matrA.ptr[i][sizeAB - 1] * matrB.ptr[sizeAB - 1][j];
```

```
        }
      }
    }
    free(colFactor);
    free(rowFactor);
    return matrRes;
}

void threadVectorRow(Matrix matrA, int* rowFactor, int beginRow, int endRow)
{
    for (int i = beginRow; i < endRow; i++)
    {
        for (int j = 0; j < matrA.numColumns / 2; j++)
        {
            rowFactor[i] += matrA.ptr[i][2 * j] * matrA.ptr[i][2 * j + 1];
        }
    }
}

void threadVectorCol(Matrix matrB, int* colFactor, int beginCol, int endCol)
{
    for (int i = beginCol; i < endCol; i++)
    {
        for (int j = 0; j < matrB.numRows / 2; j++)
        {
            colFactor[i] += matrB.ptr[2 * j][i] * matrB.ptr[2 * j + 1][i];
        }
    }
}
```

Листинг 3. Алгоритм Винограда распараллеливания заполнения результирующей матрицы.

```
#include "parallelCycle.h"

Matrix parallelCycle(Matrix matrA, Matrix matrB, int numThreads)
{
    int rowA = matrA.numRows;
    int sizeAB = matrA.numColumns;
    int colB = matrB.numColumns;

    int numPairs = sizeAB / 2;

    //Произведение элементов в каждой строке matrA
    int *rowFactor = createVector(rowA);
    for (int i = 0; i < rowA; i++)
    {
        for (int j = 0; j < numPairs; j++)
        {
            rowFactor[i] += matrA.ptr[i][2 * j] * matrA.ptr[i][2 * j + 1];
        }
    }

    //Произведение элементов в каждом столбце matrB
    int *colFactor = createVector(colB);
    for (int i = 0; i < colB; i++)
    {
```

```
        for (int j = 0; j < numPairs; j++)
        {
            colFactor[i] += matrB.ptr[2 * j][i] * matrB.ptr[2 * j + 1][i];
        }
    }

    Matrix matrRes(rowA, colB);

    //Создание потоков
    thread* threads = new thread[numThreads];
    int rowsPerThred = rowA / numThreads;
    int beginRow = 0;

    for (int i = 0; i < numThreads; i++)
    {
        int endRow = beginRow + rowsPerThred;
        if (i == numThreads - 1)
        {
            endRow = rowA;
        }

        threads[i] = thread(threadCycle, matrA, matrB, matrRes,
                            rowFactor, colFactor, beginRow, endRow);
        beginRow = endRow;
    }

    for (int i = 0; i < numThreads; i++)
    {
        threads[i].join();
    }

    //Случай нечетности
    if (sizeAB % 2 != 0)
    {
        for (int i = 0; i < rowA; i++)
        {
            for (int j = 0; j < colB; j++)
            {
                matrRes.ptr[i][j] += matrA.ptr[i][sizeAB - 1] * matrB.ptr[sizeAB - 1][j];
            }
        }
    }

    free(colFactor);
    free(rowFactor);

    return matrRes;
}

void threadCycle(Matrix matrA, Matrix matrB, Matrix matrRes,
                 int* rowFactor, int* colFactor, int beginRow, int endRow)
{
    //Вычисление произведения
    for (int i = beginRow; i < endRow; i++)
    {
        for (int j = 0; j < matrB.numColumns; j++)
        {
```

```
            matrRes.ptr[i][j] -= rowFactor[i] + colFactor[j];

            for (int k = 0; k < matrB.numRows / 2; k++)
            {
                matrRes.ptr[i][j] += (matrA.ptr[i][2 * k + 1] + matrB.ptr[2 * k][j]) *
                                     (matrA.ptr[i][2 * k] + matrB.ptr[2 * k + 1][j]);
            }
        }
    }
}
```

3.4. Описание тестирования

Тестирование осуществляется по принципу «черного ящика».

Для проверки корректности программы необходимо предусмотреть наборы различных тестов, включающих в себя случаи матриц размерности 1, единичных матриц, эквивалентных матриц.

3.5. Вывод

В данном разделе были рассмотрены инструменты, необходимые для реализаций версий алгоритма Винограда, а также были представлены непосредственно реализации.

4. Экспериментальная часть

В данном разделе рассмотрим примеры работы программы, произведем тестирование, выполним эксперименты по замеру времени, а также выполним сравнительный анализ полученных данных.

4.1. Примеры работы программы

Взаимодействие с программой (меню, выбор способа создания матрицы, выбор алгоритма, вывод результата) представлено на рисунке 4.

```
Multiply matrix.......1
Measure time..........2

1

Matrix creation
First matrix
        Number of rows:3
        Number of columns:5
Second matrix
        Number of rows:5
        Number of columns:6

First matrix
Keyboard input.......1
Generate.............2
2

1 7 0 7 5
7 1 3 6 1
5 4 5 7 5

Second matrix
Keyboard input.......1
Generate.............2
2

4 6 0 7 1 8
8 6 6 8 8 8
4 1 1 5 0 0
3 5 3 1 7 4
7 6 0 0 2 5

Choose the option:
        Vinograd multiplication.............1
        Vinograd parallel vectors...........2
        Vinograd parallel cycle.............3
        Exit................................4
1

Result matrix
116 113 63 70 116 117
73 87 27 78 59 93
128 124 50 99 96 125
Choose the option:
        Vinograd multiplication.............1
        Vinograd parallel vectors...........2
        Vinograd parallel cycle.............3
        Exit................................4
2

Result matrix
116 113 63 70 116 117
73 87 27 78 59 93
128 124 50 99 96 125
```

Рисунок 4. Пример взаимодействия с программой.

4.2. Технические характеристики устройства.

Операционная система - LinuxMint 64-bit;

Память - 8 ГБ;

Процессор - Intel(R) Core(TM) i3-7100U CPU @ 2.40 ГГц;

Логических процессов - 4.

4.3. Результаты тестирования

Результаты тестирования приведены в таблице 4.

Таблица 4. Результаты тестирования.

Входные параметры		Ожидаемый результат	Полученный результат
Матрица A	Матрица B		
4	8	32	32
1 7 0 7 5 7	1 3 6 1 5 4 5 7 5 4 6 0 7 1 8 8 6 6	36 52 41 29 47 4 81 63 123 83 107 70	36 52 41 29 47 4 81 63 123 83 107 70
1 0 0 0 1 0 0 0 1	1 7 0 7 5 7 1 3 6 1 5 4	1 7 0 7 5 7 1 3 6 1 5 4	1 7 0 7 5 7 1 3 6 1 5 4
1 2 3 4 8 2 8 4 6	1 2 3 4 8 2 8 4 6	33 30 25 52 80 40 72 72 68	33 30 25 52 80 40 72 72 68

Поскольку результаты работы реализаций алгоритма совпадают, в столбце «Полученный результат» приведено значение, одинаковое для всех реализаций.

4.4. Сравнение реализаций по времени работы

Выполняются две серии экспериментов по замеру времени на квадратных матрицах.

1. Эксперименты проводятся на матрицах размерами 10, 11, 20, 21, 30, 31, 40, 41, 50, 51, 100, 101, 200, 201. Число потоков постоянно и равно 4. Поскольку работа алгоритма Винограда зависит от четности размеров, разобьем тестирование на четный и нечетный случаи. Результаты замеров приведены на рисунке 4.1.

2. Эксперименты проводятся на матрицах постоянных размеров: 50 элементов. Число потоков переменно: 1, 2, 4, 8, 16, 32. Результаты замеров приведены на рисунке 4.2.

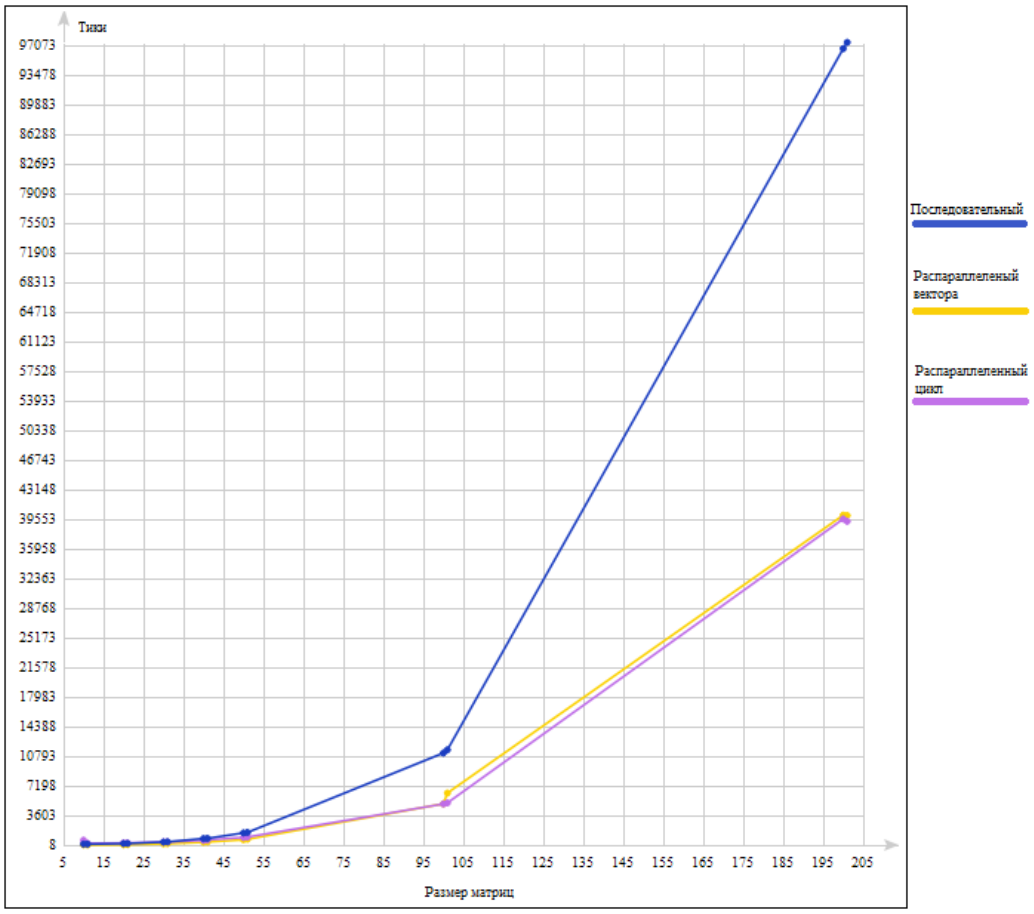

Рисунок 4.1. График зависимости времени работы алгоритмов от размеров матрицы.

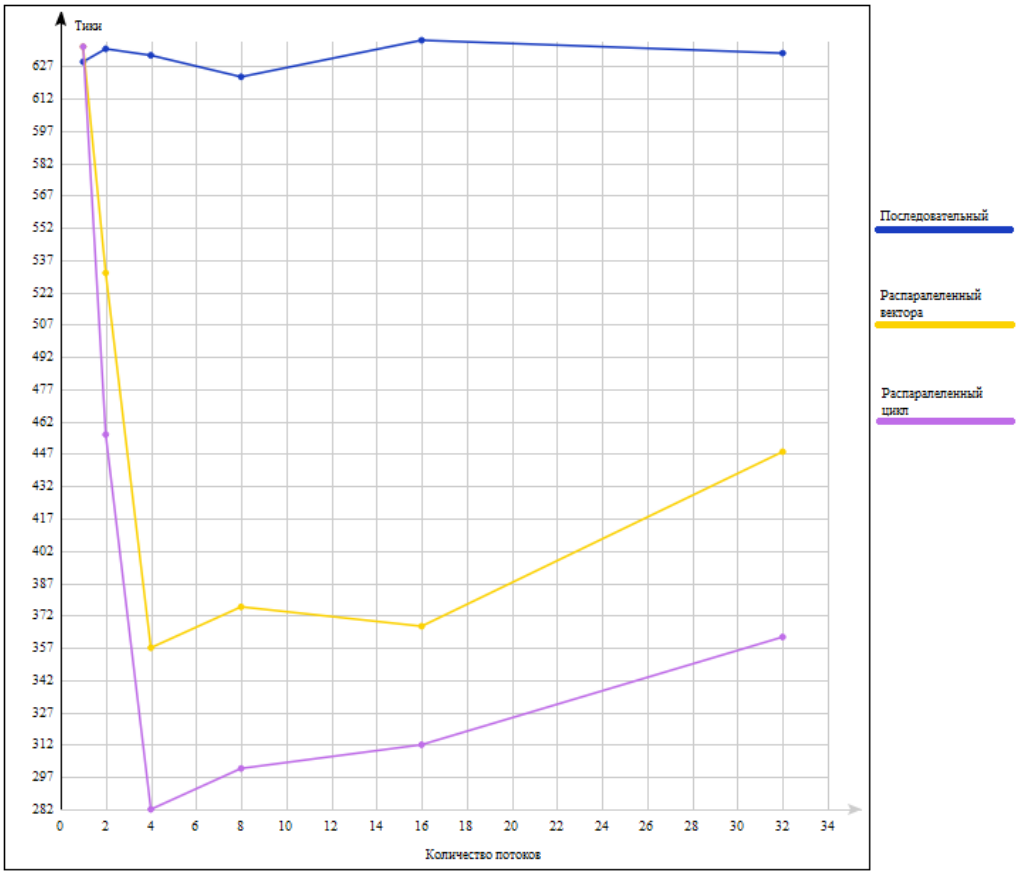

Рисунок 4.2. График зависимости времени работы алгоритмов от количества потоков.

Следует отметить, что самой долгой реализацией алгоритма для матриц больших размеров является последовательная версия. Самой быстрой – распараллеливание тройного вложенного цикла заполнения результирующей матрицы. Что касается матриц небольших размеров (до 50), то линейная реализация работает быстрее, так как накладные расходы на параллельные реализации (дополнительный цикл, операции; создание потоков) превышают выгоду по времени.

Заключение

В ходе лабораторной работы мы разработали и реализовали три версии алгоритма Винограда умножения матриц, а также провели анализ скорости работы для каждой версии.

В результате анализа было установлено, что на матрицах больших размеров (более 50) самой долгой является последовательная реализация. Так, максимальная разница по времени составляет 1.7 и 2 раза с распараллеленной версией с векторами и циклом соответственно. Поскольку эксперименты выполнялись на устройстве с 4 логическими процессами, то наибольшая скорость распараллеленных версий будет достигнута при создании 4 потоков.

На матрицах небольших размеров (до 50) последовательная версия работает быстрее, поскольку дополнительные накладные расходы при распараллеливании превышают выгоду по времени.

Что касается двух распараллеленных версий, распараллеливание цикла работает быстрее распараллеливания заполнения векторов в 1.2 раза. Это связано с тем, что заполнение результирующей матрицы имеет наибольшую трудоемкость, и распределение тройного вложенного цикла на потоки дает больший выигрыш по времени, чем распределение двойного вложенного цикла.

Список литературы

1. Линейная алгебра.-URL: http://www.academiaxxi.ru/WWW_Books/HM/La/toc.htm (дата обращения: 05.09.2020). Текст: электронный.
2. S. A. Winograd. A new algorithm for inner product // IEEE Trans. Comp. — 1968. — C-18.
3. Афанасьев К. Е. Многопроцессорные вычислительные системы и параллельное программирование. – Кемерово: Кузбассвузиздат, 2003.
4. Техническая документация.-URL: https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/clock?view=vs-2019 (дата обращения: 20.09.2020). Текст: электронный.