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

Дисциплина: Анализ алгоритмов

Лабораторная работа 4

Умножение матриц с помощью алгоритма Винограда с параллельными вычислениями.

Студент группы ИУ7-55,
Шестовских Николай Александрович
Преподаватели: Волкова Л.Л., Строганов Ю.В.

2019 г.

Содержание

Введение

Целью данной работы является изучение и реализация параллельного алгоритма Винограда для умножения матриц. Необходимо сравнить зависимость времени работы алгоритма от числа параллельных потоков исполнения и размера матриц, провести сравнение стандартного и параллельного алгоритма.

1 Аналитическая часть

В данной части будут рассмотрены теоретические основы алгоритмов.

1.1 Теоретические сведения об умножении матриц

Матрица – это прямоугольная таблица каких-либо элементов. Здесь и далее мы будем рассматривать только матрицы, элементами которых являются числа. Упорядоченная пара чисел (n, m), где n - количество строк в матрице, m - количество столбцов, называется размерностью матрицы, обозначается обычно m x n[1].

Пусть имеются две матрицы: A и B размерами n x l и l x m соответственно.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & ... & a_{1,l} \\ a_{2,1} & a_{2,2} & ... & a_{2,l} \\ ... & ... & ... & ... \\ a_{n,1} & a_{n,2} & ... & a_{n,l} \end{bmatrix}$$

$$\begin{bmatrix} b_{1,1} & b_{1,2} & ... & b_{1,m} \\ b_{2,1} & b_{2,2} & ... & b_{2,m} \\ ... & ... & ... & ... \\ b_{l,1} & b_{l,2} & ... & b_{l,m} \end{bmatrix}$$

Произведением матриц A и B размерами n x l и l x m соответственно называется матрица C размерами n x m, каждый элемент которой вычисляется по формуле (1):

$$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j} \tag{1}$$

$$\begin{bmatrix} c_{1,1} & b_{1,2} & ... & c_{1,m} \\ c_{2,1} & b_{2,2} & ... & c_{2,m} \\ ... & ... & ... & ... \\ c_{n,1} & c_{n,2} & ... & c_{n,m} \end{bmatrix}$$

1.2 Алгоритм Винограда

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Также некоторые вычисления можно произвести заранее, что ускорит выполнение алгоритма. Рассмотрим два вектора V = (v_1, v_2, v_3, v_4) и W = (w_1, w_2, w_3, w_4)
Их скалярное произведение находится по формуле (2)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{2}$$

Равенство (2) можно переписать в виде (3)

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{3}$$

В Алгоритме Винограда используется скалярное произведение из формулы 2, в отличие от стандартного алгоритма. Алгоритм Винограда позволяет выполнить предварительную обработку матрицы и запомнить значения для каждой строки/столбца матриц. Над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения[2].

1.3 Параллельные вычисления

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

При использовании многопроцессорных вычислительных систем с общей памятью обычно предполагается, что имеющиеся в составе системы процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти, и время доступа к памяти является одинаковым (при одновременном доступе нескольких процессоров к одному и тому же элементу памяти очередность и синхронизация доступа обеспечивается на аппаратном уровне). Многопроцессорные системы подобного типа обычно именуются симметричными мультипроцессорами (symmetric multiprocessors, SMP).
Перечисленному выше набору предположений удовлетворяют также активно развиваемые в последнее время многоядерные процессоры, в которых каждое ядро представляет практически независимо функциони рующее вычислительное устройство.

Обычный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью – создание новых параллельных методов на основе обычных последовательных программ, в которых или автоматически компилятором, или непосредственно программистом выделяются участки независимых друг от друга вычислений. Возможности автоматического анализа программ для порождения параллельных вычислений достаточно ограничены, и второй подход является преобладающим. При этом для разработки параллельных программ могут применяться как новые алгоритмические языки, ориентированные на параллельное программирование, так и уже имеющиеся языки, расширенные некоторым набором операторов для параллельных вычислений.

Широко используемый подход состоит и в применении тех или иных библиотек, обеспечивающих определенный программный интерфейс (application programming interface, API) для разработки параллельных программ. В рамках такого подхода наиболее известны Windows Thread API. Однако первый способ применим только для ОС семейства Microsoft Windows, а второй вариант API является достаточно трудоемким для использования и имеет низкоуровневый характер [3].

1.4 Параллельный алгоритм Винограда

Трудоемкость алгоритма Винограда имеет сложность $O(nmk)$ для умножения матриц $n1 \times m1$ на $n2 \times m2$. Чтобы улучшить алгоритм, следует распараллелить ту часть алгоритма, которая содержит 3 вложенных цикла. Помимо этого, можно объединить всю остальную часть алгоритма и ее тоже распараллелить, тем самым разбив алгоритм на два последовательно выполняющихся участка.

В результирующей матрице каждая ячейка вычисляется независимо от других, поэтому, вычисляя отдельные строки разными потоками, получится не сталкиваться с проблемой "разделяемых данных"между потоками, так как каждый поток будет отвечать за свой участок итоговой матрицы.

1.5 Вывод

В данном разделе были рассмотрены общие сведения об умножении матриц, алгоритм Винограда, способ его распараллеливания а также теоретические сведения о параллельных вычислениях.

2 Конструкторская часть

В данной части будут рассмотрены схема алгоритма винограда и модели его распараллеливания.

2.1 Схема алгоритма Винограда

На рисунке 1 приведена схема алгоритма Винограда.

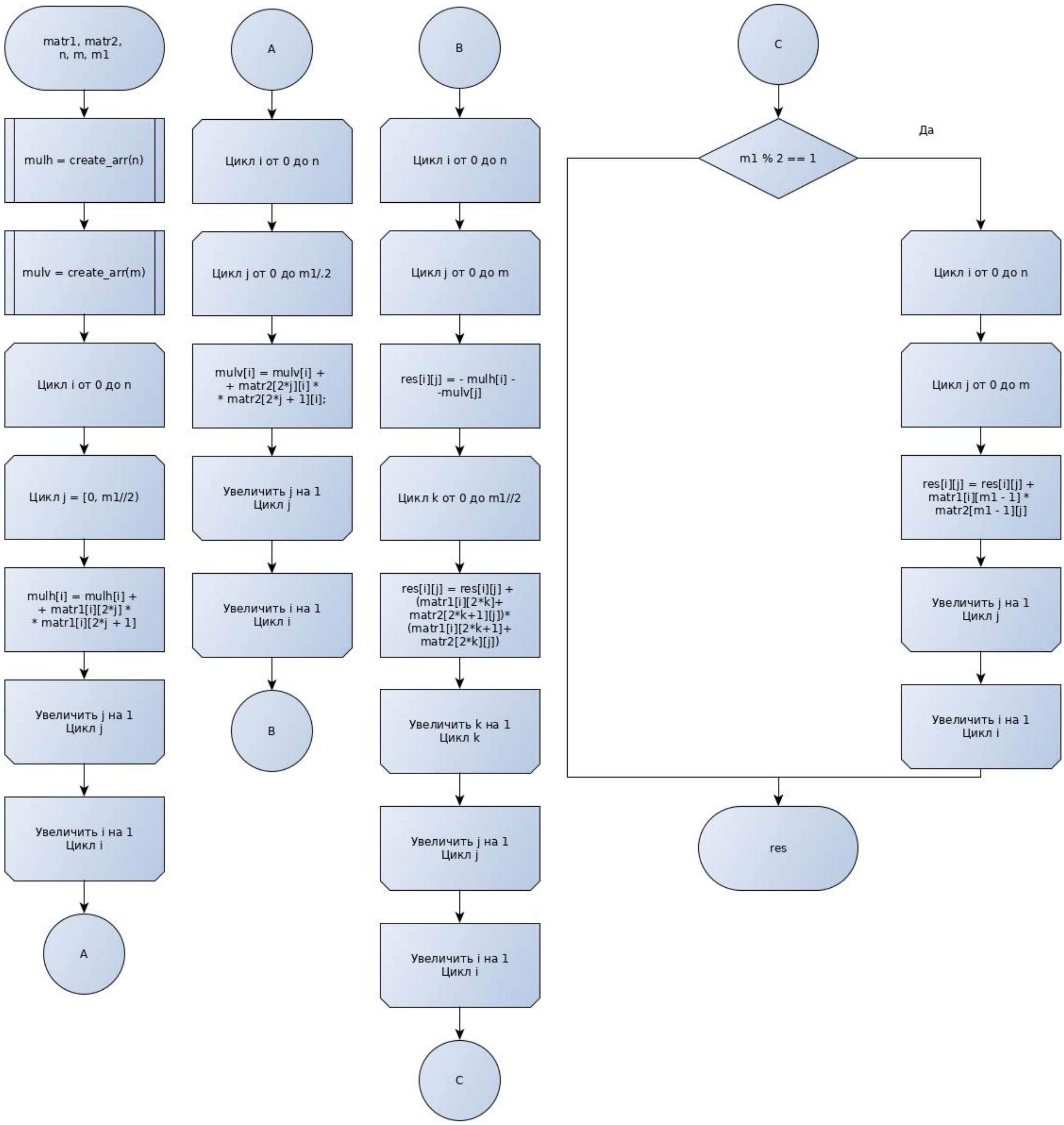

Рис. 1: Алгоритм Винограда

2.2 Модель организации параллельных вычислений

Алгоритм Винограда можно разбить условно на 4 части:

1. вычисление вектора mulh(на Рис. 1 эта часть находится в промежутке от начала алгоритма до соединителя A);
2. вычисление вектора mulv (на Рис. 1 от A до B);
3. основная часть(на Рис. 1 от A до C);
4. дополнительные вычисления в случае нечетного количества столбцов в первой матрице(на Рис. 1 от C до конца алгоритма).

В таком случае можно распараллелить часть 3, модель представлена на рисунке 2, квадраты на ней - этапы алгоритма, перегородки - ожидание всех потоков:

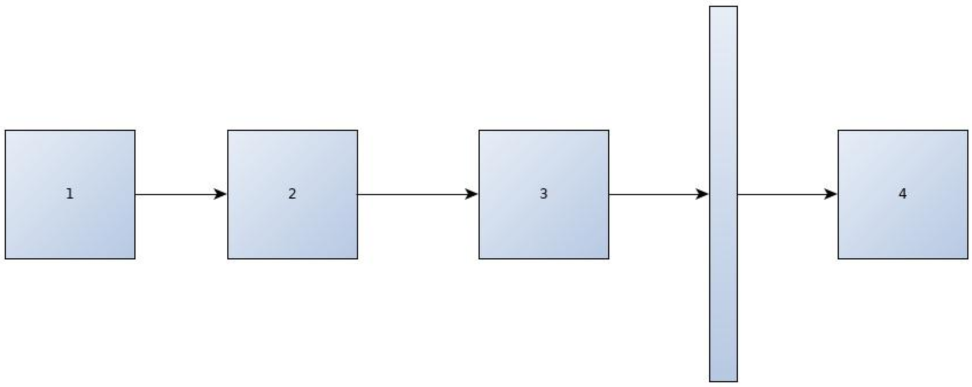

Рис. 2: Модель распараллеливания 1 для алгоритма Винограда

Также можно заметить, что этап 4 является независимым от этапов 1 и 2 с точки зрения разделяемой памяти, друг с другом они также независимы. В связи с этим можно использовать модель распараллеливания, представленную на рисунке 3:

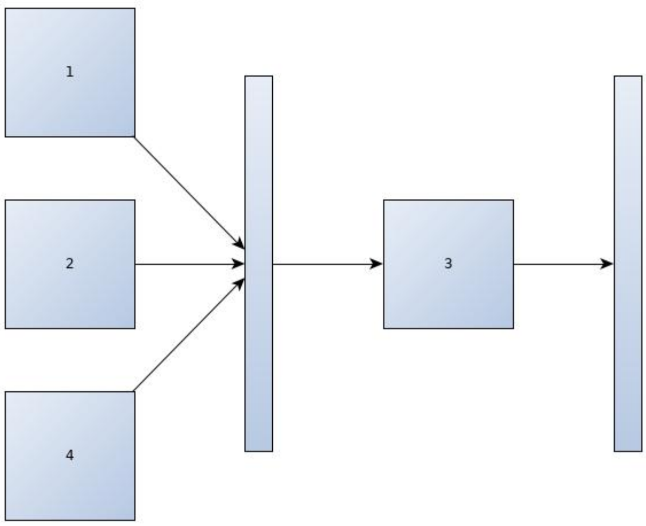

Рис. 3: Модель распараллеливания 2 для алгоритма Винограда

2.3 Вывод

В данном разделе была рассмотрена схема алгоритма Винограда и модели для его распараллеливания - модель, распараллеливающая только тройной цикл и модель, распараллеливающая тройной цикл и все кроме него.

3 Технологическая часть

В данном разделе будут приведены листинги алгоритма Винограда и его вариантов с параллельными вычислениями на языке c++

3.1 Требования к программному обеспечению

Входные данные - матрица1, матрица2, их размеры. Выходные данные - произведение матриц.

Рис. 4: IDEF0-диаграмма, описывающая алгоритм умножения матриц

3.2 Средства реализации

Программа была написана на языке C++[7], так как этот язык хорошо сбалансирован с точки зрения быстродействия и предоставляемого функционала, в качестве среды был использован QTCreator[8], так как он бесплатный и достаточно удобный для настройки сборки и программирования в объектно-ориентированном стиле. Для параллельных вычислений использовалась библиотека thread[10], для замеров времени - библиотека chrono[9].

Наблюдатель joinable проверяет потенциальную возможность работы потока в параллельном контексте. Операция join ожидает завершения потока[5].

3.3 Листинг кода

В листингах 1-7 приведены все рассматриваемые в рамках данной лабораторной работы алгоритмы, написанные на языке C++.

Листинг 1: Алгоритм Винограда

```
1  Matrix Vinograd(Matrix& matr1, Matrix& matr2)
2  {
3    int n = matr1.Matrix::GetN();
4    int m = matr2.Matrix::GetM();
5    int m1 = matr1.Matrix::GetM();
6    double *mulh = new double[n];
7    double *mulv = new double[m];
8
9    Matrix res = Matrix(n, m);
10
11   for (int i = 0; i < n; i++)
12     for (int j = 0; j < m1/2; j++)
13       mulh[i] = mulh[i] + matr1[i][2*j] * matr1[i][2*j + 1];
14
15   for (int i = 0; i < m; i++)
16     for (int j = 0; j < m1/2; j++)
17       mulv[i] = mulv[i] + matr2[2*j][i] * matr2[2*j + 1][i];
18
19   for (int i = 0; i < n; i++)
20   {
21     for (int j = 0; j < m; j++)
22     {
23       res[i][j] = - mulh[i] - mulv[j];
24       for (int k = 0; k < m1/2; k++)
25         res[i][j] = res[i][j] + (matr1[i][2*k]+matr2[2*k+1][j])*(matr1[i][2*k+1]+
             matr2[2*k][j]);
26     }
27   }
28
29   if (m1 % 2)
30   {
31     for(int i = 0; i < n; i++)
32       for(int j = 0; j < m; j++)
33         res[i][j] = res[i][j] + matr1[i][m1 - 1] * matr2[m1 - 1][j];
34   }
35   delete[] mulh;
36   delete[] mulv;
37   return res;
38 }
```

На листинге 1 мы видим, что первой части алгоритма соответствует блок кода на 11-13 строках, второй части - блок кода на 15-17 строках, третьей - блок кода на 19-27 строках, четвертой - на 29-34 строках.

Листинг 2: Этап 3(тройной цикл) в алгоритме Винограда

```
void func_thread(Matrix &matr1, Matrix &matr2, Matrix &res, double* mulh, double*
    mulv, int num, int count)
{
  for(int i = num; i < matr1.Matrix::GetN(); i += count)
  {
    for(int j = 0; j < matr2.Matrix::GetM(); j++)
    {
      res[i][j] = res[i][j] - mulh[i] - mulv[j];
      for(int k = 0; k < matr1.Matrix::GetM() / 2; k++)
      {
        res[i][j] = res[i][j] + (matr1[i][2*k] + matr2[2*k+1][j]) * (matr1[i][2*k +
            1] + matr2[2*k][j]);
      }
    }
  }
}
```

На листинге 3 представлен алгоритм Винограда по модели 2, рассмотренной в разделе 2.2 .

Листинг 3: Алгоритм Винограда по модели 1

```
Matrix Vinograd_Parallell1(Matrix& matr1, Matrix& matr2, int count)
{
  int n = matr1.Matrix::GetN();
  int m = matr2.Matrix::GetM();
  int m1 = matr1.Matrix::GetM();
  double *mulh = new double[n];
  double *mulv = new double[m];

  Matrix res = Matrix(n, m);

  for (int i = 0; i < n; i++)
    for (int j = 0; j < m1/2; j++)
      mulh[i] = mulh[i] + matr1[i][2*j] * matr1[i][2*j + 1];

  for (int i = 0; i < m; i++)
    for (int j = 0; j < m1/2; j++)
      mulv[i] = mulv[i] + matr2[2*j][i] * matr2[2*j + 1][i];

  std::thread threads[count];

  for(int i = 0; i < count; i++)
    threads[i] = std::thread(func_thread, std::ref(matr1), std::ref(matr1), std::ref(
        res), mulh, mulv, i, count);

  for(int i = 0; i < count; i++)
    if(threads[i].joinable())
      threads[i].join();

  if (m1 % 2)
    for(int i = 0; i < n; i++)
      for(int j = 0; j < m; j++)
        res[i][j] = res[i][j] + matr1[i][m1 - 1] * matr2[m1 - 1][j];
  delete[] mulh;
  delete[] mulv;
  return res;
}
```

Листинг 4: Этап 1(вычисление mulh) в алгоритме Винограда

```
void Get_Mulh(Matrix& matr1, double *mulh, int i, int count)
{
  int n = matr1.Matrix::GetN();
  int m1 = matr1.Matrix::GetM();
  for (int i = 0; i < n; i += count)
    for (int j = 0; j < m1/2; j++)
      mulh[i] = mulh[i] + matr1[i][2*j] * matr1[i][2*j + 1];
}
```

Листинг 5: Этап 2(вычисление mulv) в алгоритме Винограда

```
void Get_Mulv(Matrix& matr1, Matrix& matr2, double *mulv, int i, int count)
{
  int m = matr2.Matrix::GetM();
  int m1 = matr1.Matrix::GetM();

  for (; i < m; i += count)
    for (int j = 0; j < m1/2; j+)
      mulv[i] = mulv[i] + matr2[2*j][i] * matr2[2*j + 1][i];
}
```

Листинг 6: Этап 4(дополнительные вычисления) в алгоритме Винограда

```
void func_thread2(Matrix& matr1, Matrix& matr2, Matrix& res, int i, int count)
{
  int n = matr1.Matrix::GetN();
  int m = matr2.Matrix::GetM();
  int m1 = matr1.Matrix::GetM();

  for(; i < n; i += count)
    for(int j = 0; j < m; j++)
      res[i][j] = res[i][j] + matr1[i][m1 - 1] * matr2[m1 - 1][j];
}
```

На листинге 7 представлен алгоритм Винограда по модели 2, рассмотренной в разделе 2.2 .

Листинг 7: Алгоритм Винограда по модели 2

```
Matrix Vinograd_Parallell2(Matrix& matr1, Matrix& matr2, int count)
{
  int n = matr1.Matrix::GetN();
  int m = matr2.Matrix::GetM();
  int m1 = matr1.Matrix::GetM();
  double * mulh = new double[n];
  double * mulv = new double[m];
  for (int i = 0; i < n; i++)
    mulh[i] = 0;
  for (int i = 0; i < m; i++)
    mulv[i] = 0;

  Matrix res = Matrix(n, m);

  if (m1 % 2)
  {

    int count1 = (count + (count % 4)? (4 - count % 4) : 0)/4;
    std::thread t1[count1];
    std::thread t2[count1];
    std::thread t3[count1*2];
    for(int i = 0; i < count1; i++)
    {
      t1[i] = std::thread(Get_Mulh, std::ref(matr1), mulh, i, count1);
      t2[i] = std::thread(Get_Mulv, std::ref(matr1), std::ref(matr2), mulv, i, count1
          );
      if(t1[i].joinable())
        t1[i].join();
      if(t2[i].joinable())
        t2[i].join();
    }
    for(int i = 0; i < count1 * 2; i++)
    {
      t3[i] = std::thread(func_thread2, std::ref(matr1), std::ref(matr2), std::ref(
          res), i, count1*2);
      if(t3[i].joinable())
        t3[i].join();
    }
  }
  else
  {
    int count1 = (count + ((count % 2)? 1 : 0))/2;
    std::thread t1[count1];
    std::thread t2[count1];

    for(int i = 0; i < count1; i++)
    {
      t1[i] = std::thread(Get_Mulh, std::ref(matr1), mulh, i, count1);
      t2[i] = std::thread(Get_Mulv, std::ref(matr1), std::ref(matr2), mulv, i, count1
          );
      if(t1[i].joinable())
        t1[i].join();
      if(t2[i].joinable())
        t2[i].join();
```

```
        }
      }
      std::thread threads[count];

      for(int i = 0; i < count; i++)
        threads[i] = std::thread(func_thread, std::ref(matr1), std::ref(matr1), std::ref(
            res), &mulh[0], &mulv[0], i, count);

      for(int i = 0; i < count; i++)
        if(threads[i].joinable())
          threads[i].join();
      delete[] mulh;
      delete[] mulv;
        return res;
    }
```

3.4 Вывод

В данном разделе были рассмотрены листинг алгоритма Винограда, а также листинги моделей распараллеливания этого алгоритма, рассмотренных в разделе 2.2.

4 Экспериментальная часть

В данном разделе будет экспериментально найдено быстродействие алгоритма Винограда в сравнении с его распараллеленными по модели 1 и модели 2 версиями.

4.1 Постановка эксперимента

Требуется провести замеры времени для работы алгоритмов при размерах матрицы от 100 до 1000 с шагом 100 и при размерах матрицы от 101 до 1001 с шагом 100, так как в Алгоритме Винограда при нечетных размерах матрицы добавляются дополнительные вычисления, это должно сыграть роль в модели распараллеливания 2, где эти вычисления происходят одновременно с нахождением векторов mulh и mulv.

Тестовый компьютер:

- ЦПУ -INTEL® PENTIUM® N5000 (4 физических ядра, 4 логических ядра, базовая тактовая частота - 1.1 ГГц, максимальная - 2.7 ГГц, в большинстве случаев тактовая частота ≈ 2.3 ГГц)[6]
- ОЗУ - 8 ГБ 2400 МГц
- ОС - Ubuntu Mate 18.04

4.2 Результаты эксперимента

На рисунках 5 и 6 представлены замеры времени для обычного алгоритма Винограда и распараллеленного на 2, 4, 8, 16 потоков по модели 1. На рисунках 7 и 8 представлены замеры времени для алгоритма Винограда, распараллеленного на 2, 4, 8, 16 потоков алгоритма винограда по моделям 1 и 2.

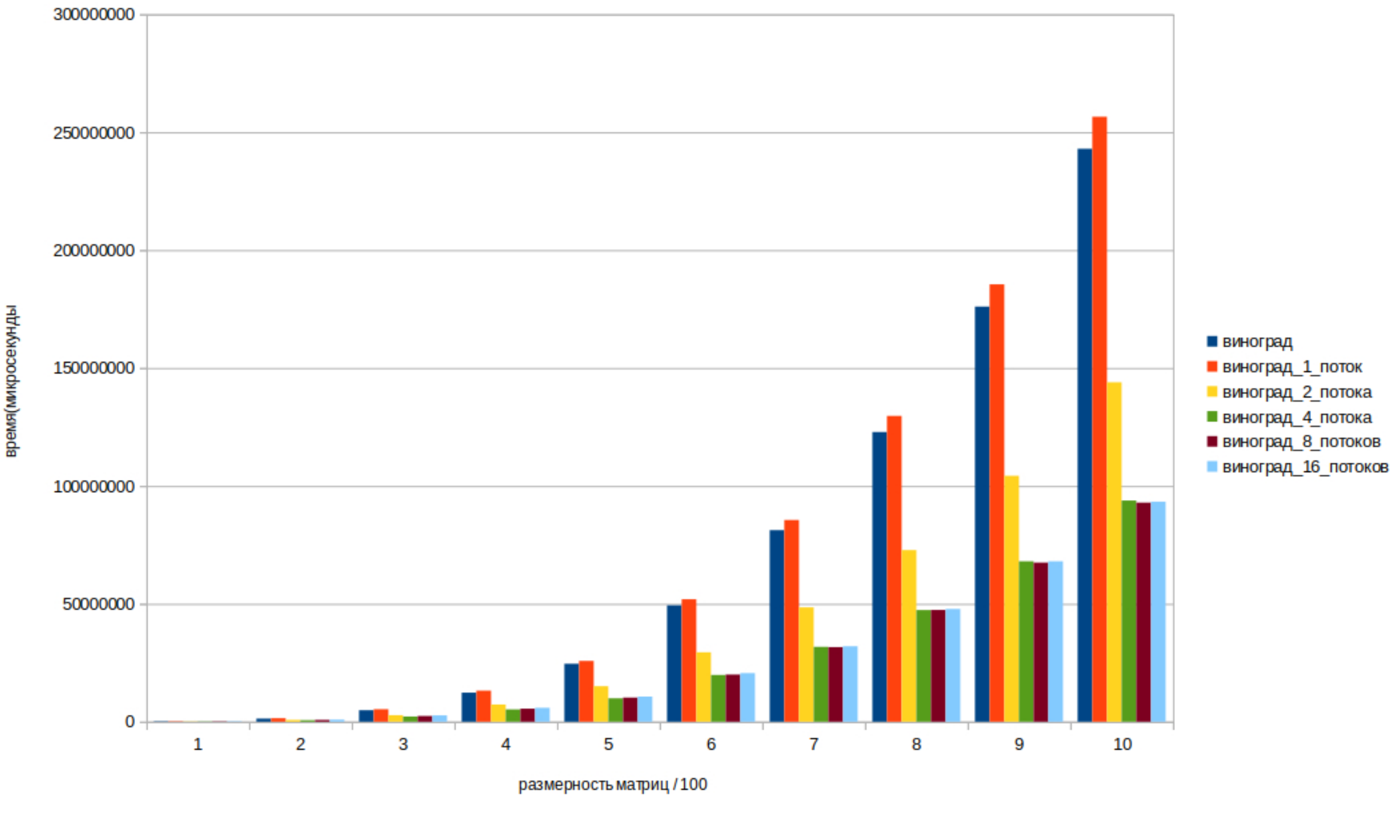

Рис. 5: Сравнение быстродействия алгоритма Винограда и алгоритма Винограда, распараллеленного по модели 1 на четных размерах матрицы

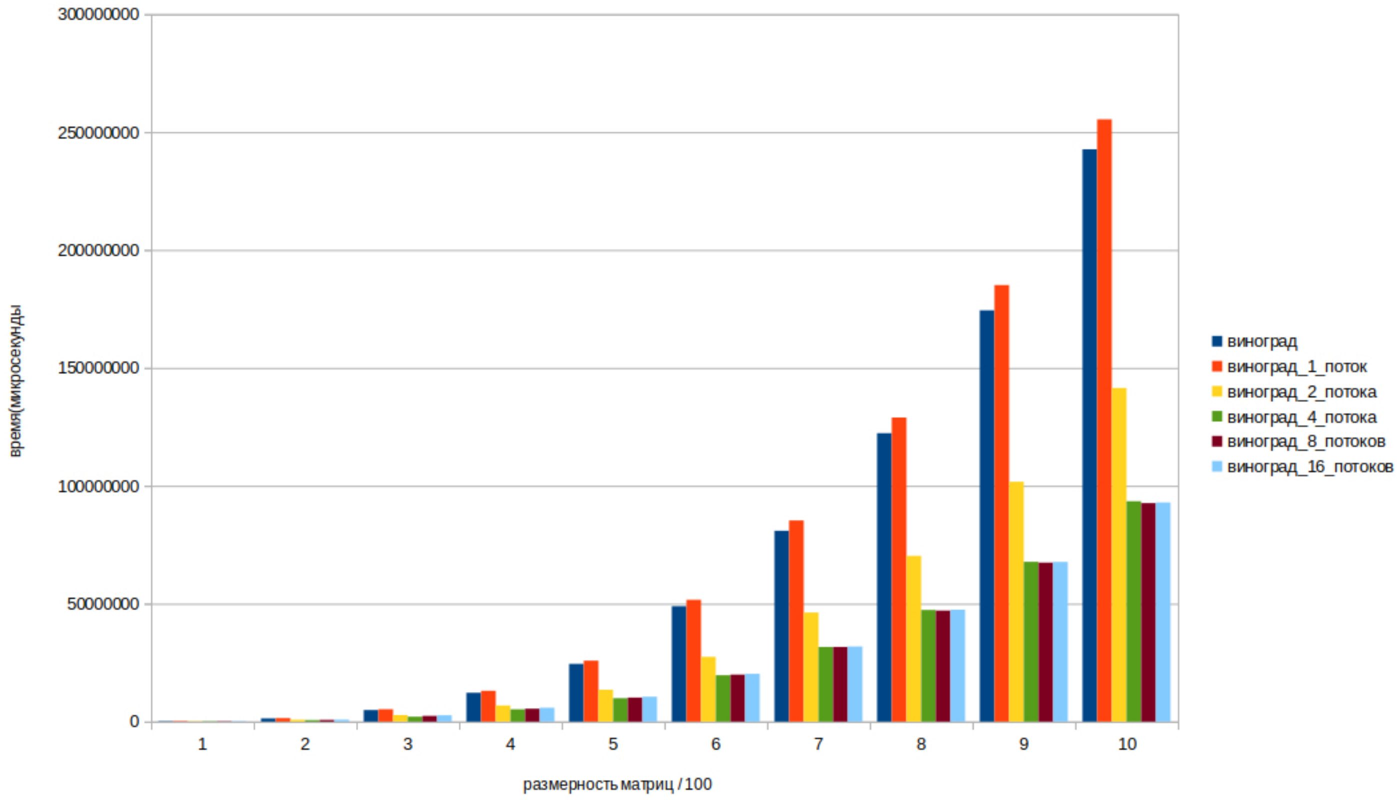

Рис. 6: Сравнение быстродействия алгоритма Винограда и алгоритма Винограда, распараллеленного по модели 1 на нечетных размерах матрицы

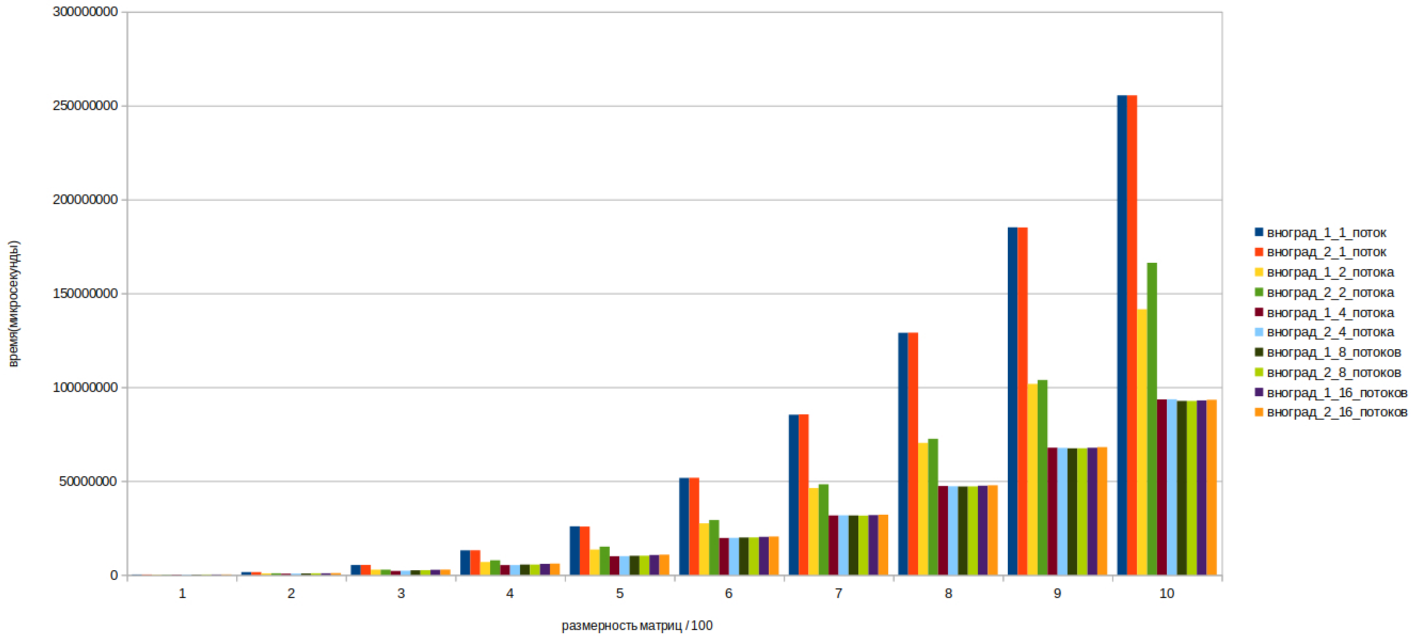

Рис. 7: Сравнение быстродействия алгоритма Винограда, распараллеленного по моделям 1 и 2 на четных размерах матрицы

4.3 Анализ полученых результатов эксперимента

На рисунках 5 и 6 мы видим, что наибольший прирост производительности происходит при переходе от обычного алгоритма Винограда к двухпоточному(разница растет от 52% до 72% при возрастающей матрице), меньше, но все же значительный прирост алгоритм получает при переходе от 2 к 4 потокам(от 23% до 51%). При дальнейшем увеличении числа потоков программы при матрицах размером 400x400 производительность растет не сильно(в районе 0.5%), в то время как на более маленьких матрицах наоборот падает(на 8% при переходе от 4 к 8 потокам и от 8 к 16). Следует заметить, что компьютер, на котором производилось тестирование, имеет 4 логических ядра, то есть он физически не может обрабатывать количество потоков большее, чем 4, что и вызвало столь малый прирост при переходе от 4 к 8 и 16 потокам. Что касается стандартного алгоритма Винограда и однопоточного, видно, что однопоточная реализация всегда отстает(хотя разница уменьшается от 14% при матрицах 100x100 до 5% при матрицах 1000x1000),это обусловлено работой библиотеки pthreads для обеспечения потоков на уровне пользователя. По грисункам 5 и 6 мы делаем вывод, что нет смысла параллелить алгоритм Винограда на число потоков большее, чем логических ядер процессора

По рисункам 7 и 8 можно заметить, что при равных количествах алгоритм Винограда, распараллеленный по моделям 1 и 2 практически не отличается, при размерностях матрийц (в районе от 0.1% до 3% в пользу то одной модели, то другой), из чего мы делаем вывод, что в алгоритме достаточно распараллелить только самую трудоемкую часть, чтобы получить достаточный прирост производительности.

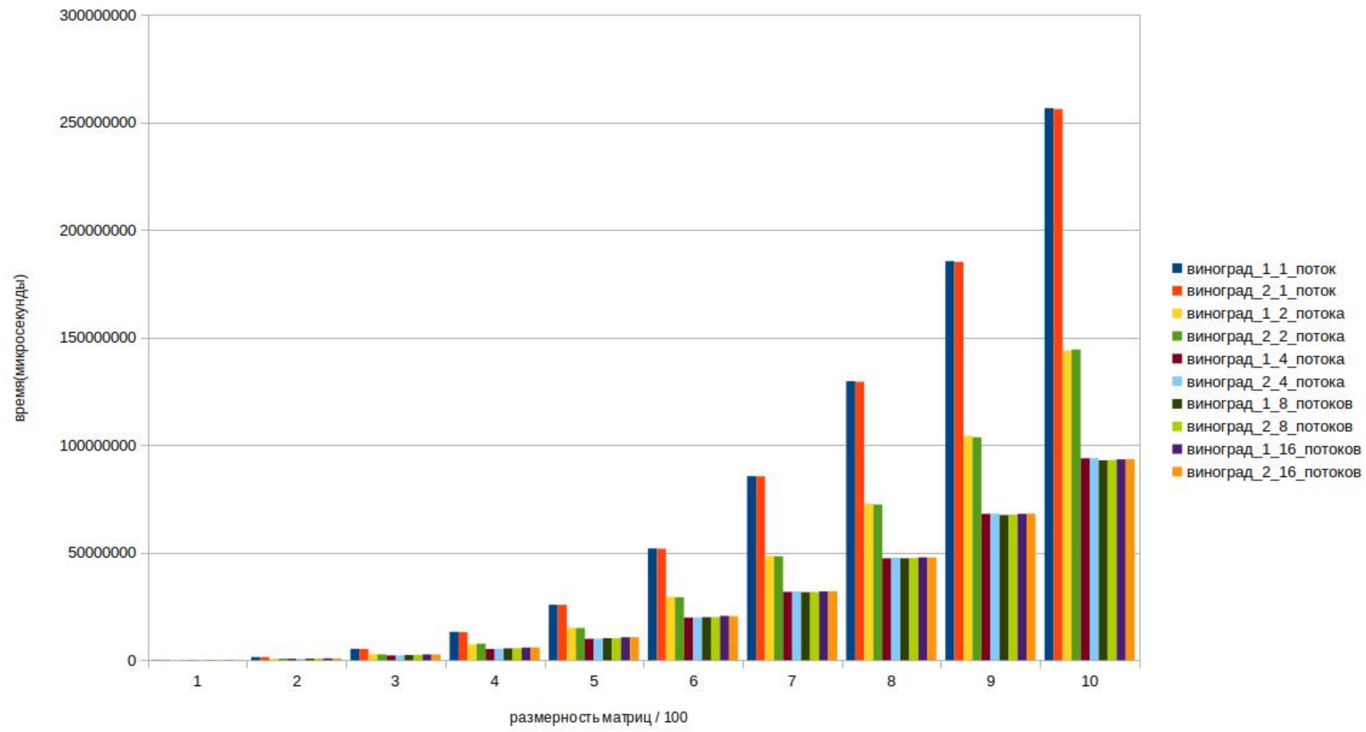

Рис. 8: Сравнение быстродействия алгоритма Винограда, распараллеленного по моделям 1 и 2 на нечетных размерах матрицы

4.4 Вывод

В данном разделе было рассмотрено сравнение быстродействия алгоритма Винограда и его распараллеленных версий по модели 1 и 2. По результатам тестирования оказалось, что наибольший прирост наблюдается при переходе от обычного алгоритма Винограда к четырехпоточной версии модели, в которой распараллелен только тройной цикл.

Заключение

В ходе лабораторной работы были исследованы методы распараллеливания алгоритма Винограда и их эффективность на практике а так же получены навыки в области параллельных вычислений. По результатам тестирования эффективности двух моделей распараллеливания алгоритма Винограда были сделаны следующие выводы.

- Нет смысла параллелить алгоритмы на количество потоков больше, чем число логических ядер процессора, однако, если программное обеспечение разрабатывается для широкого круга компьютеров, не стоит забывать, что существуют процессоры с 16 и больше потоками, на которых чем сильнее распараллелена программа, тем лучше.

- В алгоритме Винограда достаточно распараллелить только самую трудоемкую часть алгоритма, параллеливание остальных участков не дает прироста больше 3%, а иногда и вовсе вредит.

Список литературы

[1] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16

[2] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.-М.:Техносфера, 2009.

[3] Константин Баркалов, Владимир Воеводин, Виктор Гергель. Intel Parallel Programming [Электронный ресурс], - режим доступа https://www.intuit.ru/studies/courses/4447/983/lecture/14925

[4] Pthreads: Потоки в русле POSIX. [Электронный ресурс], - режим доступа https://habr.com/ru/post/326138/

[5] Multithreading in C++ [Электронный ресурс], - режим доступа https://www.geeksforgeeks.org/multithreading-in-cpp/

[6] Спецификации процессора INTEL PENTIUM 5000 [Электронный ресурс] - режим доступа https://www.intel.ru/content/www/ru/ru/products/processors/pentium/n5000.html

[7] Документация по языку C++ [Электронный ресурс] - режим доступа https://devdocs.io/cpp/

[8] Документация по среде QTCreator [Электронный ресурс] - режим доступа http://doc.crossplatform.ru/qtcreator/1.2.1/

[9] Документация по библиотеке chrono [Электронный ресурс] - режим доступа https://docs.microsoft.com/ru-ru/cpp/standard-library/chrono?view=vs-2019

[10] Документация по библиотеке chrono [Электронный ресурс] - режим доступа https://docs.microsoft.com/ru-ru/cpp/standard-library/chrono?view=vs-2019