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

Дисциплина: Анализ алгоритмов

Лабораторная работа № 4

Параллельная реализация алгоритма Винограда

Студент:
Сиденко Анастасия Генадьевна
Группа: ИУ7-53Б

Преподаватели:
Строганов Юрий Владимирович
Волкова Лилия Леонидовна

2019 г.

Содержание

Введение

Матрицы упоминались ещё в древнем Китае, называясь тогда «волшебным квадратом». Основным применением матриц было решение линейных уравнений. Также волшебные квадраты были известны чуть позднее у арабских математиков, примерно тогда появился принцип сложения матриц. Сама теория матриц начала своё существование в середине XIX века. Термин «матрица» ввел Джеймс Сильвестр в 1850 г. Сегодня матрицы применяются уже не только при решении линейных уравнений. [1]

Умножение матриц — это один из базовых алгоритмов, который широко применяется в различных численных методах, и в частности в алгоритмах машинного обучения. Многие реализации прямого и обратного распространения сигнала в сверточных слоях нейронной сети базируются на этой операции. [2] В физике и других прикладных науках матрицы – являются средством записи данных и их преобразования.[4] В программировании – в написании программ, массивы. Широкое применение в компьютерной графике: любая картинка на экране – это двумерная матрица, элементами которой являются цвета точек, а также матрицы используются для преобразования фигур. [5]

В психологии понимание термина сходно с данным термином в математике, но взамен математических объектов подразумеваются некие "психологические объекты"– например, тесты. [6]

Кроме того, умножение матриц имеет широкое применение в экономике[7], биологии[8], химии[9].

Также существует абстрактная модель – теорию бракосочетаний в первобытном обществе, где с помощью матриц были показаны разрешенные варианты браков для представителей и даже потомков того или иного племени.[3]

Таким образом, умножение матриц – широко используемая операция, а значит существует необходимость сокращения вычислений данной операции.

В данной лабораторной работе ставятся следующие задачи.

1. Изучение алгоритма Винограда с оптимизациями и его распараллеливание.
2. Оценка трудоемкости алгоритма умножения матриц.
3. Получение практических навыков параллельной реализации данного алгоритма на одном из языков программирования (с нативными потоками).
4. Сравнительный анализ алгоритма по затрачиваемым ресурсам (зависимость времени от длины строки) для разного количества потоков.
5. Экспериментальное подтверждение различий в трудоемкости алгоритма.

1 Аналитическая часть

Умножение матриц – это одна из основных вычислительных операций. Вычислительная сложность стандартного алгоритма умножения матриц порядка N составляет $O(N^3)$. Но существуют более сложные алгоритмы, которые дают лучший результат, например алгоритм Винограда.

1.1 Описание задачи

Пусть даны две прямоугольные матрицы $A[M \times N]$ и $B[N \times Q]$:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1q} \\ b_{21} & b_{22} & \cdots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nq} \end{bmatrix}$$

Тогда матрица $C[M \times Q]$ – произведение матриц:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1q} \\ c_{21} & c_{22} & \cdots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mq} \end{bmatrix},$$

в которой каждый элемент вычисляется по формуле 1:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \ \ (i = 1, 2, \ldots l; j = 1, 2, \ldots n) \quad (1)$$

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором; в этом случае говорят, что матрицы "согласованы".

В данной лабораторной работе стоит задача распараллеливания алгоритма Винограда. Так как каждый элемент матрицы С вычисляется независимо от других и матрицы А и В не изменяются, то для того, чтобы вычислить произведение параллельно, достаточно просто указать какие элементы С какому потоку вычислять.

Теоретически, для нахождения значения каждой из ячеек можно создать свой поток. Но ОС не позволит создать такое число потоков, ресурсов на их создание не хватит и создание потока также занимает определенный промежуток времени.

Оптимальное число потоков равно $4 *$ количество логических ядер.

В данной работе для реализации многопоточности будут использованы нативные потоки. В программировании нативные потоки – это потоки выполнения, управляющиеся операционной системой в пространстве ядра.[10]

Помимо нативных, существуют зелёные потоки – это потоки выполнения, управление которыми вместо операционной системы выполняет виртуальная машина. Управление ими происходит в пользовательском пространстве, что позволяет им работать в условиях отсутствия поддержки встроенных потоков.[10]

1.2 Пути решения

Сложность вычисления произведения матриц порядка N по определению составляет составляет $O(N^3)$, однако существуют более эффективные алгоритмы, применяющиеся для перемножения матриц.

Также можно распараллелить данные алгоритмы, скорость работы, при разумном выборе количеств потоков, возрастет.

1.3 Выводы

В данной работе стоит задача реализации алгоритма умножения матриц. Необходимо сравнить обычную реализацию алгоритма с многопоточной.

2 Конструкторская часть

В данной работе для нахождения произведения матриц используется алгоритм Винограда и его многопоточная реализация.

Распределять нагрузку будем простым способом: каждый из потоков обрабатывает одинаковое число рядов и один поток ещё обрабатывает остаток.

2.1 Функциональная модель

На рисунке 1 представлена функциональная модель нашей задачи.

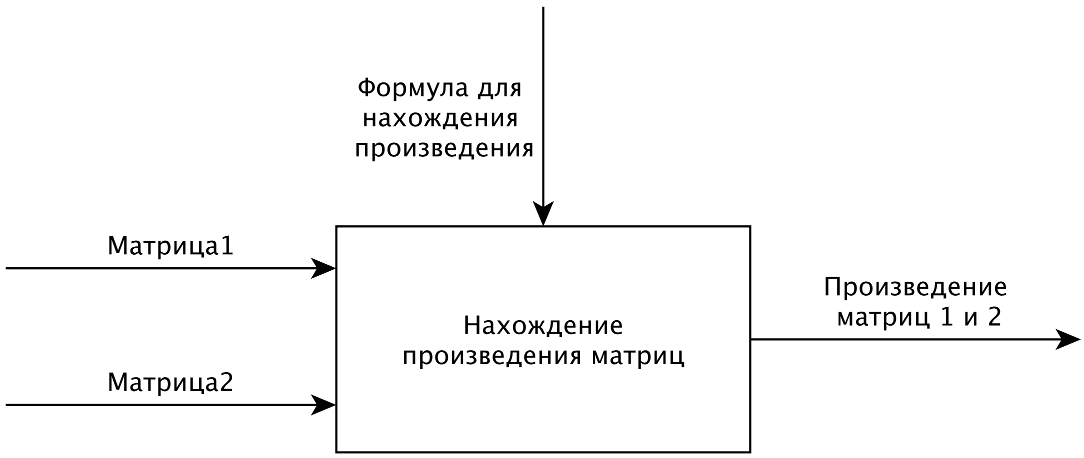

Рис. 1 - Функциональная модель алгоритма нахождения произведения матриц.

2.2 Схемы алгоритмов

Приведем схему алгоритма (см. рисунки 2-3).

Алгоритм Винограда с оптимизациями

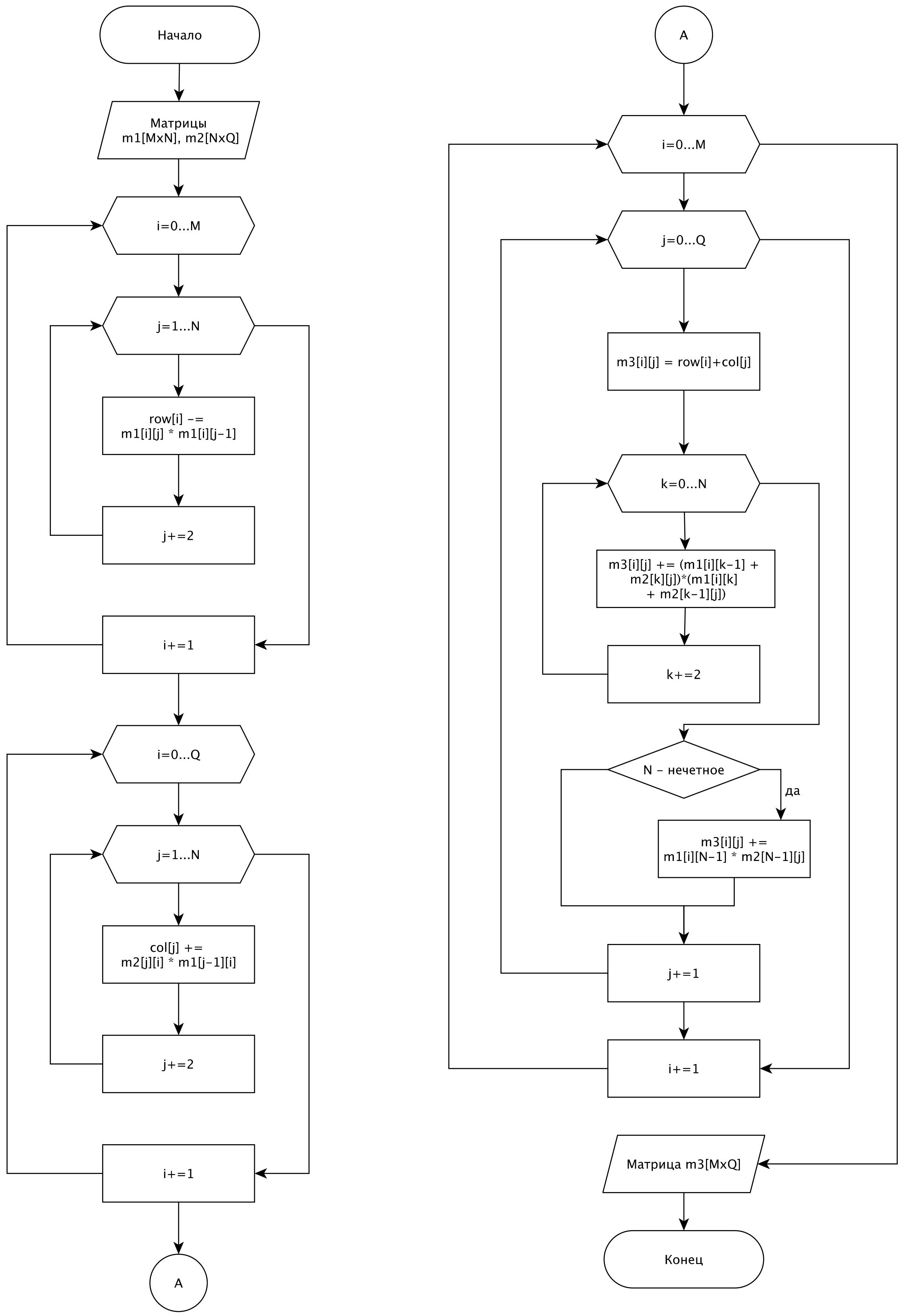

Рис. 2 - Алгоритм нахождения произведения матриц методом Винограда с оптимизациями

Многопоточный алгоритм Винограда с оптимизациями

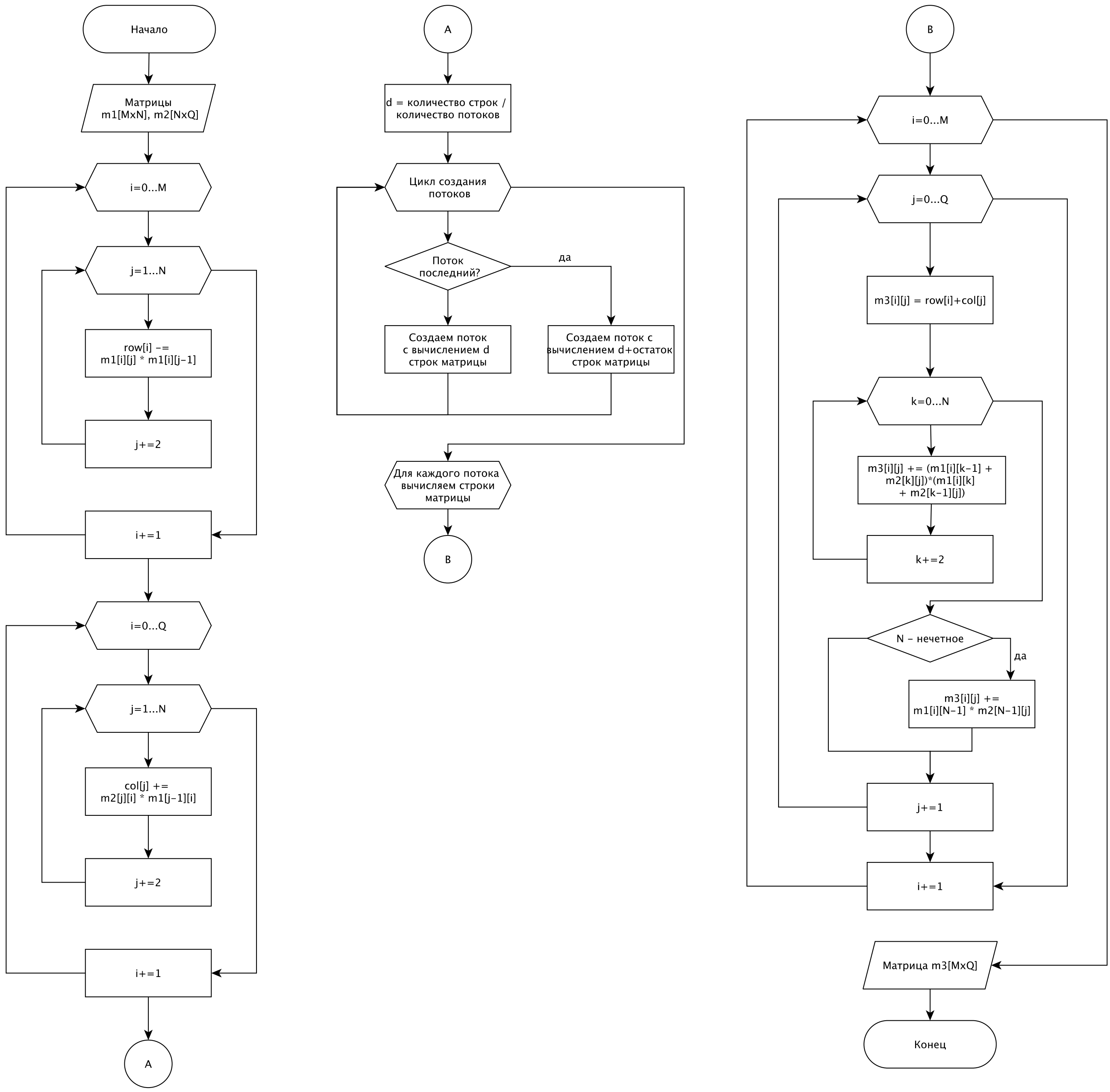

Рис. 3 - Алгоритм нахождения произведения матриц методом Винограда с оптимизациями и параллельными вычислениями

2.3 Выводы

Несмотря на сложность увеличение сложности реализации алгоритма Винограда с параллельными потоками, скорость вычисления произведения должна значительно сократиться.

Необходимо разработать данные алгоритмы и убедиться в корректности наших предположений.

3 Технологическая часть

Стоит задача разработки и сравнительного анализа алгоритмов, вычисляющих произведения матриц в однопоточной и многопоточной реализациях.

В реализациях в целях увеличения точности подсчета времени вывод матрицы был вынесен за пределы функций-алгоритмов. В целях наглядности были опущены части программ, не относящиеся к работе алгоритмов.

3.1 Требования к программному обеспечению

ПО должно предоставлять возможность замеров процессорного времени выполнения реализации каждого алгоритма. Требуется провести замеры для варьирующихся размеров матриц: от 100 до 1000 и от 101 до 1001. Один эксперимент ставится не менее 5 раз, результат одного эксперимента рассчитывается как среднее значение результатов проведенных испытаний с одинаковыми входными данными.

3.2 Средства реализации

В качестве языка программирования был выбран C++, так как я знакома с этих языком программирования и он удовлетворяет требованиям об необходимости использовании нативных потоков. [11]

Для замеров времени была выбран метод $high\_resolution\_clock :: now()$, возвращает текущее время процессора как число тиков, выраженное в микросекундах в Unix.

Для генерации случайных матриц, заданного размера использовался метод $rand()$.

Для работы с потоками использована библиотека $< thread >$.

3.3 Листинг кода

Многопоточный алгоритм Винограда:

```
void cycle(vector< vector<int> > m1, vector< vector<int> > m2,
                 vector< vector<int> > &m3, vector<int> row, vector<int> col,
                 int m_begin, int m_end) {
int n = m1[0].size();
int q = m2[0].size();
for (int i = m_begin; i < m_end; i++) {
   for (int j = 0; j < q; j++){
      m3[i][j] = row[i] + col[j];
      for (int k = 1; k < n; k += 2) {
         m3[i][j] += (m1[i][k - 1] + m2[k][j])*(m1[i][k] + m2[k - 1][j]);
      }
      if (1 == n % 2)
         m3[i][j] += m1[i][n - 1] * m2[n - 1][j];
      }
}
}

int vinograd_optimizate_multiplication_matrix(vector< vector<int> > m1,
                         vector< vector<int> > m2, int count) {
   if (m2.size() != m1[0].size()) {
      return -1;
   } else {
      int m = m1.size();
      int n = m1[0].size();
      int q = m2[0].size();
      vector< vector<int> > m3(m, vector<int> (q, 0));

      vector<int> row(m, 0);
      for (int i = 0; i < m; i++) {
```

```
            for (int j = 1; j < n; j += 2){
                row[i] -= m1[i][j] * m1[i][j - 1];
            }
        }

        vector<int> col(q, 0);
        for (int j = 0; j < q; j++) {
            for (int i = 1; i < n; i += 2){
                col[j] -= m2[i][j] * m2[i - 1][j];
            }
        }

        int d = m / count;
        vector<thread> func_thread;
        for (int i = 0; i < count; i++) {
            if (i == count - 1) {
                func_thread.push_back(thread(cycle, m1, m2, ref(m3), row, col,
                                            i * d, (1 + i) * d + m % count));
            } else {
                func_thread.push_back(thread(cycle, m1, m2, ref(m3), row, col,
                                            i * d, (1 + i) * d));
            }
        }
         for (int i = 0; i < count; i++) {
             func_thread[i].join();
         }
    }
}
```

Алгоритм Винограда:

```

int vinograd_optimizate_multiplication_matrix(vector< vector<int> > m1,
                              vector< vector<int> > m2, int count) {
    if (m2.size() != m1[0].size()) {
        return -1;
    } else {
        int m = m1.size();
        int n = m1[0].size();
        int q = m2[0].size();
        vector< vector<int> > m3(m, vector<int> (q, 0));

        vector<int> row(m, 0);
        for (int i = 0; i < m; i++) {
            for (int j = 1; j < n; j += 2){
                row[i] -= m1[i][j] * m1[i][j - 1];
            }
        }

        vector<int> col(q, 0);
        for (int j = 0; j < q; j++) {
            for (int i = 1; i < n; i += 2){
                col[j] -= m2[i][j] * m2[i - 1][j];
            }
        }

        for (int i = m_begin; i < m_end; i++) {
            for (int j = 0; j < q; j++){
                m3[i][j] = row[i] + col[j];
                for (int k = 1; k < n; k += 2) {
                    m3[i][j] += (m1[i][k - 1] + m2[k][j])*(m1[i][k] + m2[k - 1][j]);
                }
```

```
                if (1 == n % 2)
                    m3[i][j] += m1[i][n - 1] * m2[n - 1][j];
            }
        }
    }
}
```

3.4 Тестирование

В таблице 1 представлена заготовка данных для тестирования наших алгоритмов.

Матрица1	**Матрица2**	**Ожидаемый результат**
$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$	$\begin{bmatrix}30 & 36 & 42\\66 & 81 & 96\\102 & 126 & 150\end{bmatrix}$
$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$	Матрицы не могут быть перемножены
$\begin{bmatrix}1 & 2\\4 & 5\\7 & 8\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$	$\begin{bmatrix}9 & 12 & 15\\24 & 33 & 42\\39 & 54 & 69\end{bmatrix}$

Таблица 1. Подготовленные тестовые данные.

3.5 Выводы

Реализованы алгоритмы, подготовлены тесты для оценки качества их работы.

Получены практические навыки реализации алгоритмов матричного умножения методом Винограда и его распараллеливание.

4 Экспериментальная часть

4.1 Примеры работы

На рисунке 4 представлены примеры работы программы на разных входных данных. В параметрах командной строки можно указать желаемое число используемых потоков, по умолчанию 1.

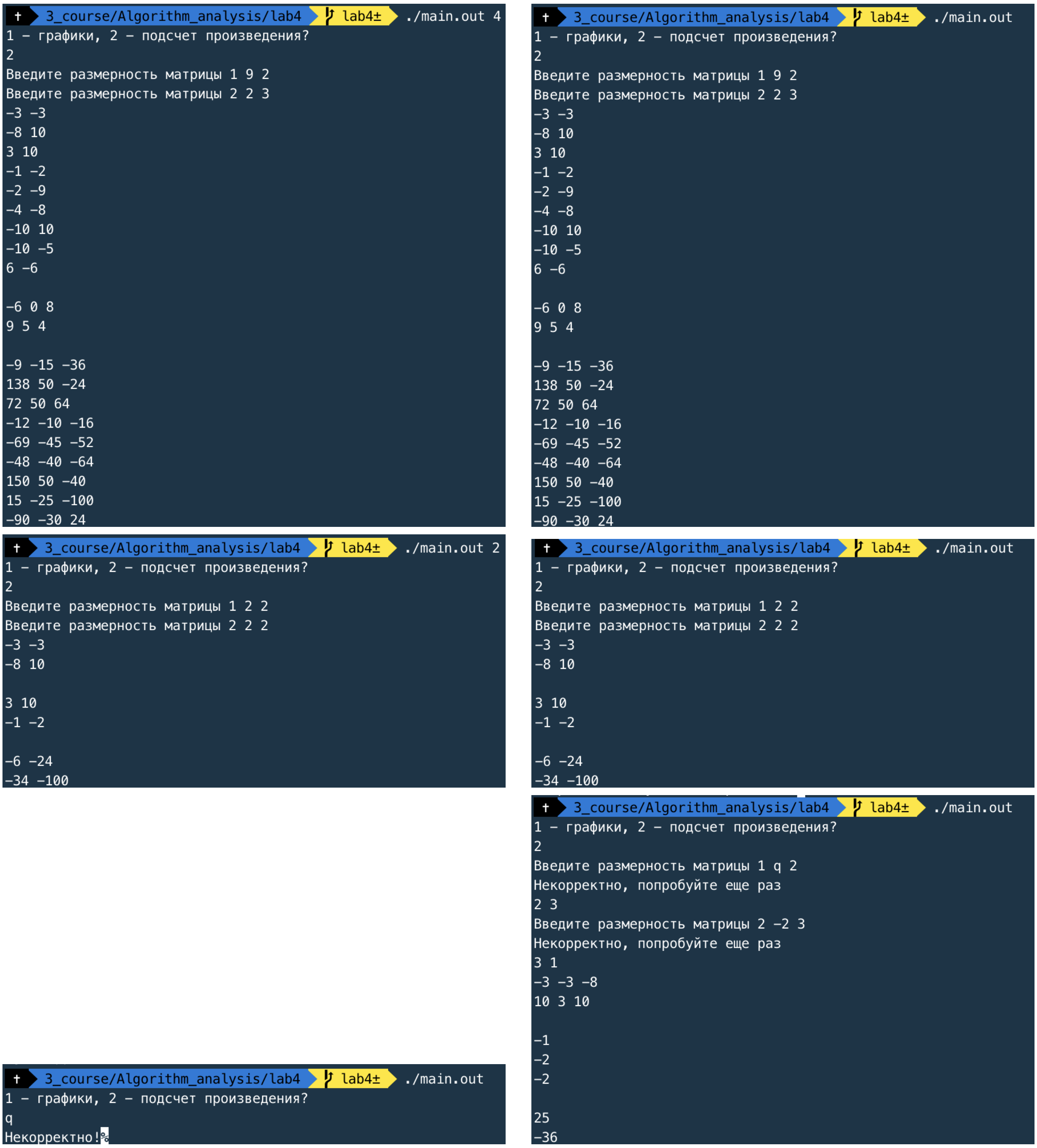

Рис. 4 - Примеры работы

4.2 Результаты тестирования

Проверяем нашу программу на тестах из таблицы 1. Полученные результаты представлены в таблице 2.

Матрица1	Матрица2	Виноград
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	Матрицы не могут быть перемножены
$\begin{bmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 9 & 12 & 15 \\ 24 & 33 & 42 \\ 39 & 54 & 69 \end{bmatrix}$

Таблица 2. Тестирование программы.

Тесты пройдены

4.3 Замеры времени

На графиках 5-6 представлено сравнение алгоритмов умножения матриц.

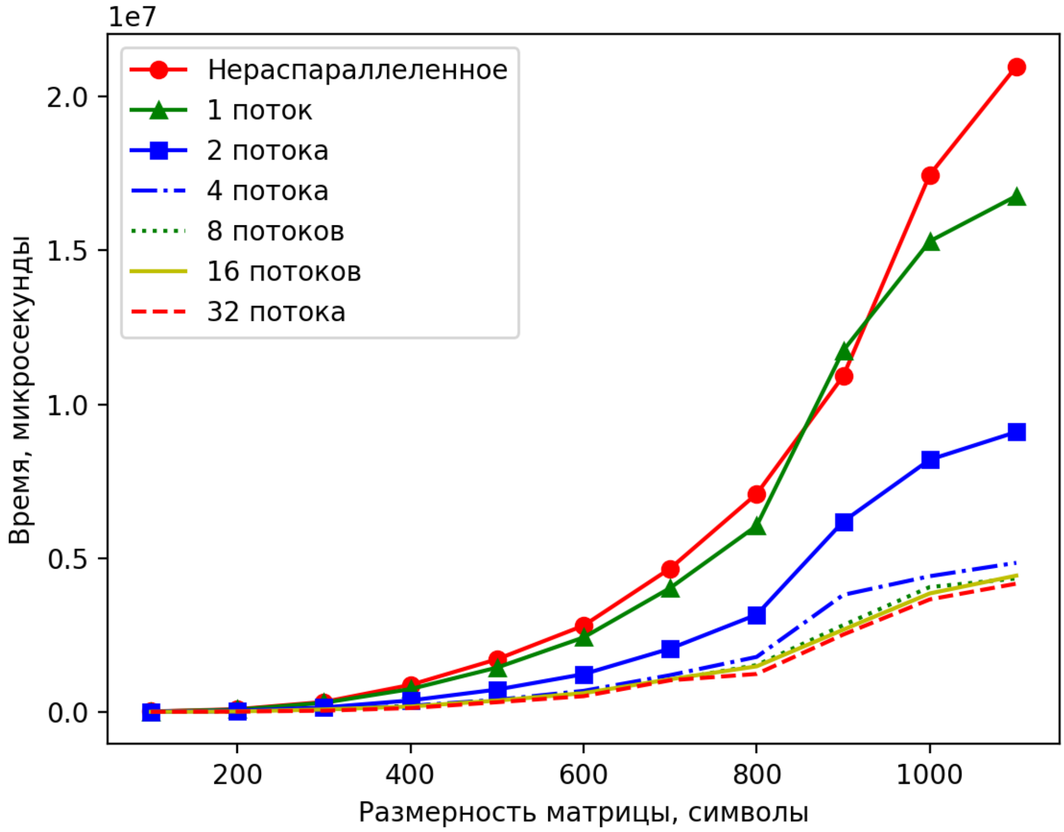

Рис. 6 - Сравнение реализации алгоритмов нахождения произведения матриц при четных размерностях

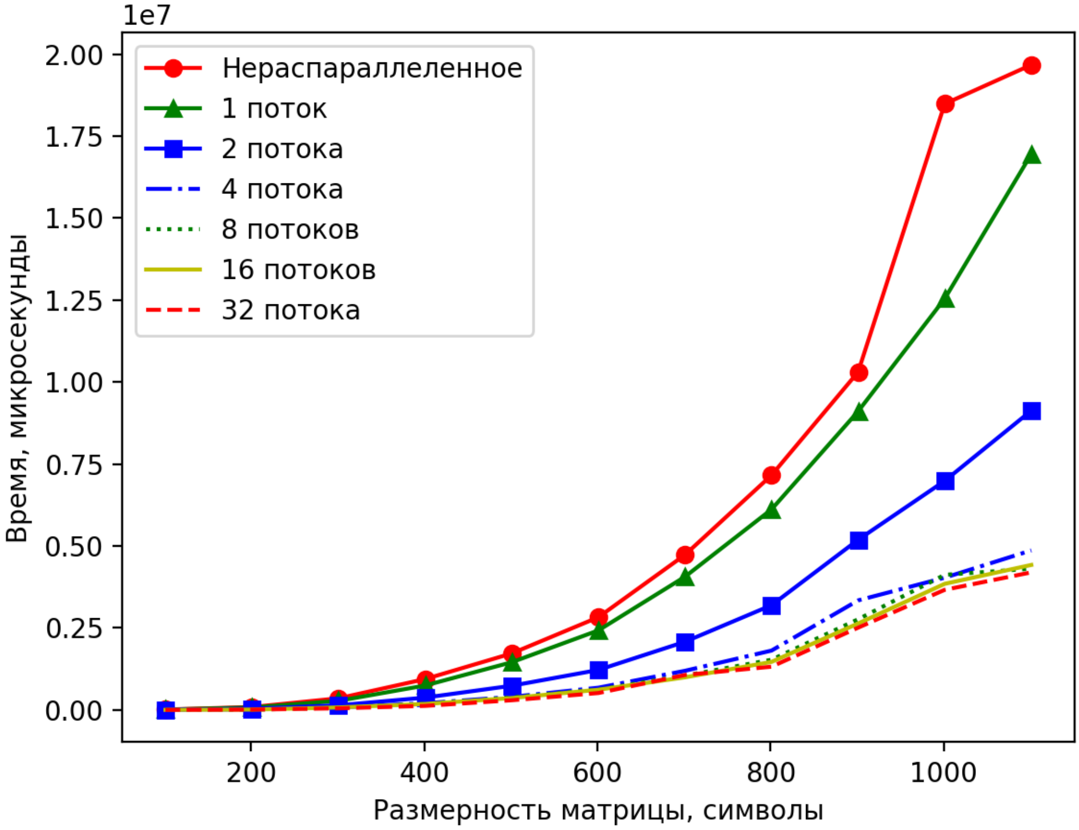

Рис. 7 - Сравнение реализации алгоритмов нахождения произведения матриц при нечетных размерностях

4.4 Выводы

Как мы видим из графиков предположения о более быстрой работе при использовании многопоточности подтвердились. Однако использование более 4 потоков не совсем оправдано.

Так, на графиках видим, что при использовании потоков с 4 до 32, скорость работы увеличивается незначительно, однако замеры времени не включают в себя создание потоков, соответственно, чем больше используется потоков, тем больше скорость работы.

В результате, оптимальным является использование 4 потоков. Скорость работы относительно многопоточного с 1 потоком, сокращается в 4 раза, а по сравнению с однопоточным в 4.5 раза.

Заключение

В данной лабораторной работе было реализованы и проанализированы алгоритмы нахождения произведения матриц методом Винограда в многопоточном и однопоточном режимах.

Матрицы и матричное умножение активно применяется:

1. в сверточных слоях нейронных сетей для реализации прямого и обратного распространения сигнала
2. в физике и математике для записи данных и их преобразования
3. в компьютерной графике, для отображения изображений и их преобразований
4. в психологии, для анализа результатов тестов
5. в экономике, биологии, химии и других науках.

При сравнении данных алгоритмов пришли к следующим выводам:

1. Что использование многопоточности значительно сокращает скорость работы программы.
2. Однако не стоит создавать огромное количество потоков, ведь так вы проиграете по времени и по памяти на создании потоков. Оптималбное число потоков - 4.

В данной лабораторной работе выполнены следующие задачи.

1. Изучены алгоритмы умножения матриц: Винограда и его параллельная реализация.
2. Получены практические навыки реализации данных алгоритмов на одном из языков программирования.
3. Проведен сравнительный анализ алгоритмов по затрачиваемым ресурсам (зависимость времени от длины строки).
4. Экспериментально подтверждено различие в трудоемкости алгоритмов.

Список литературы

[1] [Электронный ресурс]. - Режим доступа: http://poivs.tsput.ru/ru/Math/Algebra/LinearAlgebra/Matrices (дата обращения: 22.10.2019)

[2] [Электронный ресурс]. - Режим доступа: https://habr.com/ru/post/359272/(дата обращения: 22.10.2019)

[3] [Электронный ресурс]. - Режим доступа: https://urok.1sept.ru/статьи/637896/(дата обращения: 22.10.2019)

[4] [Электронный ресурс]. - Режим доступа: http://we.easyelectronics.ru/Theory/cifrovye-rekursivnye-filtry-chast-1.html(дата обращения: 22.10.2019)

[5] [Электронный ресурс]. - Режим доступа: http://window.edu.ru/resource/898/72898/files/stup559.pdf(дата обращения: 22.10.2019)

[6] [Электронный ресурс]. - Режим доступа: http://vekkv.ru/holotropnoe-dyhanie-transpersonalnaya-psihologiya/1859/(дата обращения: 22.10.2019)

[7] [Электронный ресурс]. - Режим доступа: https://www.eduherald.ru/ru/article/view?id=14118(дата обращения: 22.10.2019)

[8] [Электронный ресурс]. - Режим доступа: https://cyberleninka.ru/article/n/matrichnyy-printsip-v-biologii-progiloe-nastoyaschee-buduschee(дата обращения: 22.10.2019)

[9] [Электронный ресурс]. - Режим доступа: http://www.chemicals-el.ru/chemicals-2400-1.html(дата обращения: 22.10.2019)

[10] [Электронный ресурс] - Режим доступа: https://habr.com/ru/post/39543/ (дата обращения: 22.10.2019)

[11] [Электронный ресурс] - Режим доступа: https://xakep.ru/2013/10/25/polythread-soft/ (дата обращения: 22.10.2019)