МГТУ им. Баумана

Лабораторная работа №4

По курсу: "Анализ алгоритмов"

Матричное перемножение

Работу выполнил: Рязанов М. С., ИУ7-52Б

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

Москва, 2019

Оглавление

Введение **2**

1 Аналитическая часть **3**
1.1 Описание алгоритмов . . . 3
1.1.1 Классический алгоритм умножения матриц . . . 3
1.1.2 Умножение матриц по Винограду . . . 4
1.1.3 Выводы . . . 4

2 Конструкторская часть **6**
2.1 Разработка алгоритмов . . . 6

3 Технологическая часть **12**
3.1 Выбор ЯП . . . 12
3.2 Сведения о модулях программы . . . 12
3.3 Тестирование . . . 16
3.4 Интерфейс программы . . . 17
3.5 Сравнительный анализ реализаций . . . 17
3.5.1 Трудоемкость стандарного алгоритма . . . 18
3.5.2 Трудоемкость алгоритма Винограда . . . 18
3.5.3 Трудоемкость оптимизированного алгоритма Винограда . . . 19

4 Исследовательская часть **22**

Заключение **28**

Введение

Алгоритм умножения матриц применяется во многих серьезных вычислительных задачах, например, в таких областях как машинное обучение и компьютерная графика. Поиск способа оптимизации такой операции является важным вопросом.

Целью данной лабораторной работы является изучение алгоритмов, умножения матриц. Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать существующие методы решения задачи;
- разработать алгоритмы решения задачи;
- оптимизировать алгоритмы и расчитать их трудоемкость;
- выбрать технологии для последующей реализации и иследования алгоритмов;
- реализовать разработанные алгоритмы;
- произвести тестирование корректности работы реализаций;
- сравнить быстродействие реализаций;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В данном разделе анализируется классический подход к решению задачи об умножении матриц, а тажкже алгоритм Винограда.

1.1 Описание алгоритмов

1.1.1 Классический алгоритм умножения матриц

Матрица определяется как математический объект, записываемый в виде прямоуголольной таблицы чисел, которая представляется собой совокупность строк и столбцов, на пересечении которых находятся элементы. Количество строк и столбцов является размерностью матрицы.

Пусть даны две матрицы A размерностью $m \times q$ и B размерностью $q \times n$.

Тогда результатом умножения матрицы A на B является матрица C раз мерностью $m \times n$, в которой элемент c_{ij} вычисляется следующим образом[3]:

$$c_{ij} = \sum_{k=1}^{q} a_{ik} b_{kj}, \tag{1.1}$$

где
$i = \overline{1, m}$
$j = \overline{1, n}$

Заметим, что операция умножения двух матриц возможна только в том случае, если число столбцов в первом сомножителе равно числу строк во втором.

1.1.2 Умножение матриц по Винограду

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответсвующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.[4]

Рассмотрим два вектор $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно:

$$V \times W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w + 4. \tag{1.2}$$

Это равенство можно переписать в виде:

$$V \times W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4 \tag{1.3}$$

Кажется, что второе выражение задает больше работы, чем первое: вместо четырех умножений их щесть, а место трех сложений - десять. Менее очевидно, что выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого стоблца второй. На практике это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

В современных процессорах имеется несколько ядер, что позволяет одновременно выполнять вычисления. Для ускорения работы программы, можно ее независимые части выполнять параллельно. В случае перемножения матриц вычисление каждой строки происходит независимо от результата умножения других, поэтому данные задачи можно выполнять параллельно.

1.1.3 Выводы

В аналитическом разделе выполнено следующее:

- Расмотрены основные алгоритмы умножения матриц;
- Указаны идейные различия между ними;

- Обосновано сокращение операций в алгоритме Винограда;
- Для расмотренных алгоритмов получены расчетные соотонешния приведнные в формулах 1.1, 1.3.

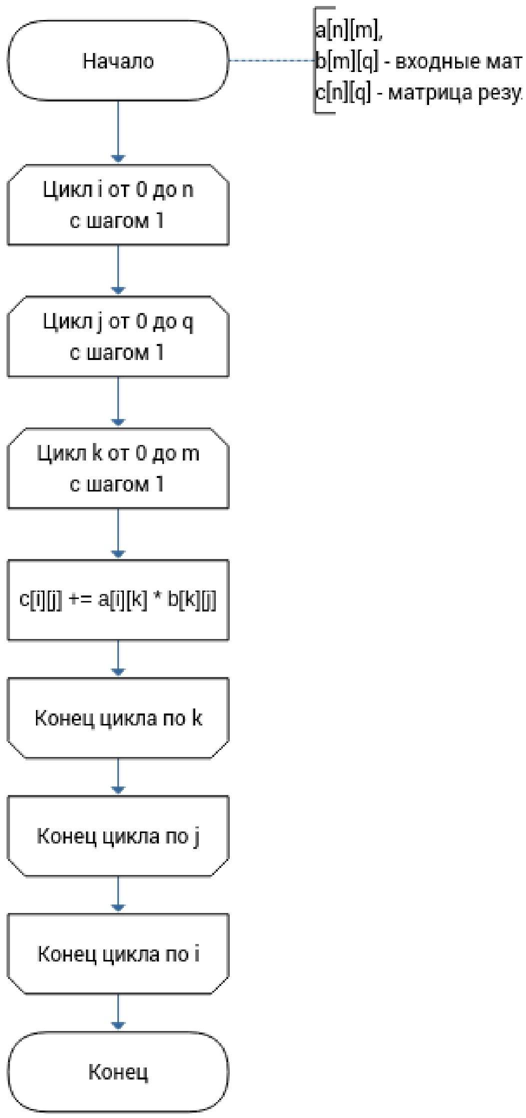
2 Конструкторская часть

2.1 Разработка алгоритмов

Общая идея обоих алгоритмов сводится к вычислению рассчетных соотношений 1.1, 1.3. За тем исключением, что для алгоритма Винограда заранее вычислить значения для каждой строки и столбца.

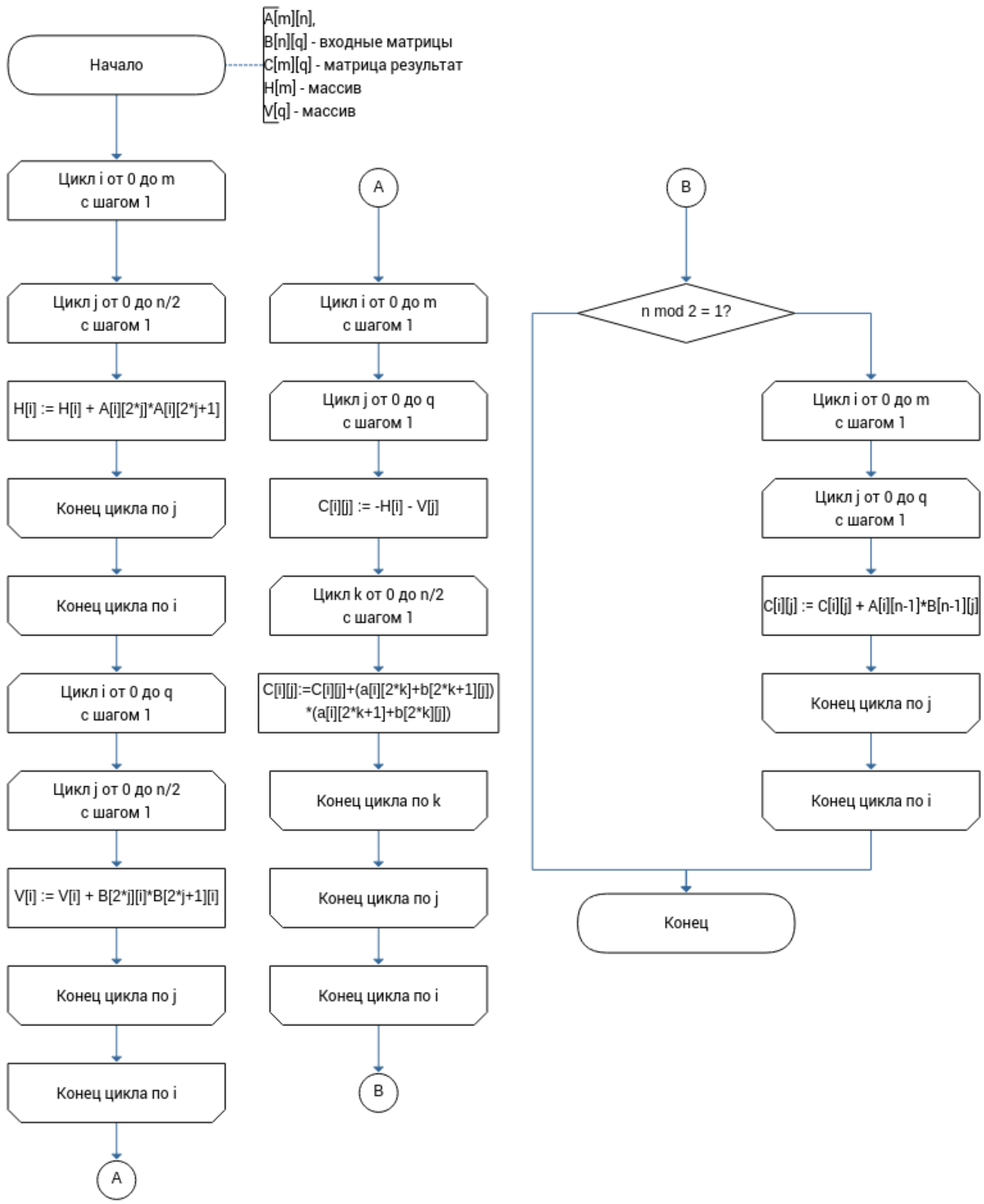
На рисинуке 2.1 приведна схема классического алгоритма умножения матриц

Рис. 2.1: Схема обычного алгоритма умножения матриц



На рисунке 2.2 приведена схема алгоритма Винограда

Рис. 2.2: Схема алгоритма Винограда



К алгоритму Винограда также можно применить следующие оптимизации:

- цикл с шагом 2 вместо условия завершения n / 2;
- использование буфера для подсчета элемента в результирующей матрице;
- корректировка значений для матриц нечетных размеров внутри общего цикла вычисления результата.

На рисунке 2.3 приведена схема оптимизированного алгоритма Винограда

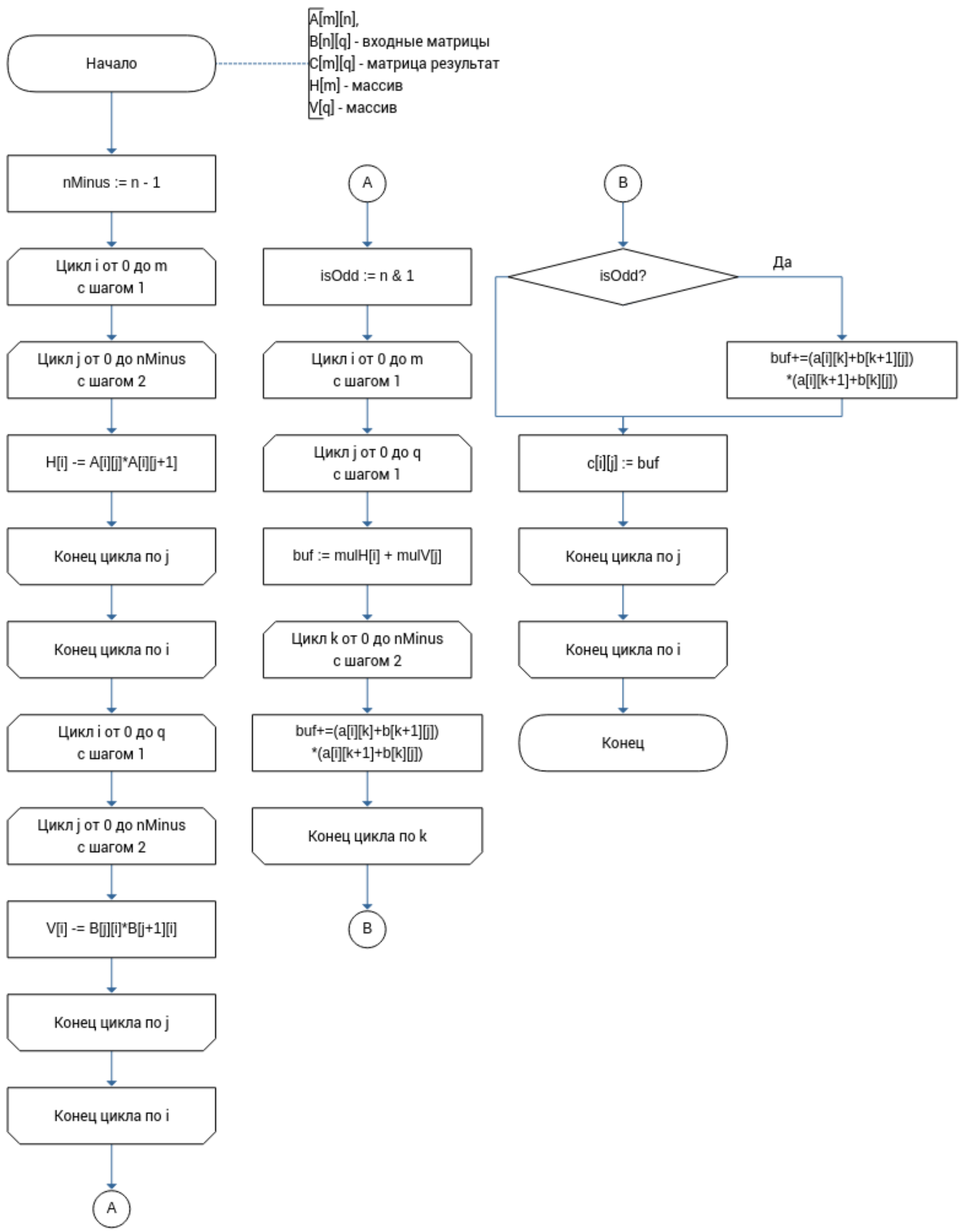


Рис. 2.3: Схема оптимизированного алгоритма Винограда

Выводы: В данном разделе были разработаны следующие алгоритмы умножения матриц

- Классический алгоритм (рисунок 2.1);
- Алгоритм Винограда (рисунок 2.2);
- Оптимизированный алгоритм Винограда(рисунок 2.3).

3 Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран C++[1], так как его стандартная библиотека предоставляет для удобной работы с потоками, что позволит быстрее реализовать многопоточную версию алгоритма.

Время работы алгоритмов было замерено с помощью функции highresolutionclock::now()‘ из библиотеки chrono[2].

3.2 Сведения о модулях программы

Программа состоит из:

- main.cpp - главный файл программы - точка входа программы, обработка входных данных;
- matrix.hpp - файл с реализацией алгоритмов;
- profile.cpp - файл с тестами производительности;
- testmatrix.cpp - файл с тестами.

В листинге 3.1 приведена реализация последовательного алгоритма Винограда.

Листинг 3.1: Оптимизированная функция умножения матриц по Винограду

```
template <class T>
Matrix<T> Matrix<T>::mul(const Matrix &left, const Matrix &
    right) {
```

```
    int m = left.rows();
    int n = right.rows();
    int q = right.cols();
    int nMinus = n - 1;
    Matrix<T> res(m, q);


    std::vector<T> mulH(m);
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < nMinus; j+=2){
            mulH[i] -= left[i][j] * left[i][j+1];
        }
    }

    std::vector<T> mulV(q);
    for (int i = 0; i < q; ++i) {
        for (int j = 0; j < nMinus; j += 2) {
            mulV[i] -= right[j][i] * right[j+1][i];
        }
    }

    bool isOdd = n % 2 == 1;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < q; ++j) {
            T buf = mulH[i] + mulV[j];
            for (int k = 0; k < nMinus; k += 2) {
                buf += (left[i][k] + right[k+1][j]) * (left[i][k+1] + right[k][j]);
            }
            if (isOdd) {
                buf += left[i][nMinus] * right[nMinus][j];
            }
            res[i][j] = buf;
        }
    }

    return res;
}
```

В листингах 3.2, 3.3, 3.4, 3.5, 3.6 приведена реализация многопоточного алгоритма Винограда.

Листинг 3.2: Многопоточная реализация алгоритма Винограда

```
template <class T>
Matrix<T> Matrix<T>::mul(const Matrix &left, const Matrix &right, size_t threads) {
    int m = left.rows();
    int n = right.rows();
    int q = right.cols();
    Matrix<T> res(m, q);

    std::vector<std::thread> trd(threads);

    std::vector<T> mulH(m);
    for (size_t i = 0; i < threads; ++i) {
        trd[i] = std::thread(rowsMul, std::ref(mulH), std::ref(left), i, threads);
    }

    joinThreads(trd);

    std::vector<T> mulV(q);
    for (size_t i = 0; i < threads; ++i) {
        trd[i] = std::thread(colsMul, std::ref(mulV), std::ref(right), i, threads);
    }

    joinThreads(trd);

    for (int i = 0; i < threads; ++i) {
        trd[i] = std::thread(mainMull, std::ref(left), std::ref(right), std::ref(res), std::ref(mulH), std::ref(mulV), i, threads);
    }

    joinThreads(trd);

    return res;
}
```

Листинг 3.3: Функция предподсчета произведений по строкам

```
template <class T>
void Matrix<T>::rowsMul(std::vector<T> &mulH, const Matrix
    &left, size_t i, size_t threads) {
     int nMinus = left.cols() - 1;
     int m = left.rows();

     for (; i < m; i+=threads) {
          for (int j = 0; j < nMinus; j+=2){
               mulH[i] -= left[i][j] * left[i][j+1];
          }
     }
}
```

Листинг 3.4: Функция предподсчета произведений по столбцам

```
template <class T>
void Matrix<T>::colsMul(std::vector<T> &mulV, const Matrix
    &right, size_t i, size_t threads) {
     int nMinus = right.rows() -1;
     int q = right.cols();

     for (; i < q; i+=threads) {
          for (int j = 0; j < nMinus; j += 2) {
               mulV[i] -= right[j][i] * right[j+1][i];
          }
     }
}
```

Листинг 3.5: Функция умножения матриц на основе предпосчитанных данных

```
template <class T>
void Matrix<T>::mainMull(const Matrix<T> &left, const
    Matrix<T> &right, Matrix<T>  &res,
     const std::vector<T> &mulH, const std::vector<T> &mulV,
     size_t i, size_t threads) {

     int m = left.rows();
     int n = right.rows();
     int q = right.cols();
```

```
    int nMinus = n - 1;
    bool isOdd = n & 1;

    for (; i < m; i += threads) {
        for (int j = 0; j < q; ++j) {
            T buf = mulH[i] + mulV[j];
            for (int k = 0; k < nMinus; k += 2) {
                buf += (left[i][k] + right[k+1][j]) * (left[i][k+1] + right[k][j]);
            }
            if (isOdd) {
                buf += left[i][nMinus] * right[nMinus][j];
            }
            res[i][j] = buf;
        }
    }
}
```

Листинг 3.6: Функция присоединения потоков

```
template <class T>
void Matrix<T>::joinThreads(std::vector<std::thread> &vec) {
    for (auto &t : vec) {
        t.join();
    }
}
```

3.3 Тестирование

Для каждой из функции были составленые следующие наборы тестов:

- пустые матрицы;
- матрицы не подходящих размеров;
- квадратные матрицы;
- матрицы разных размеров;

- матрицы четных и нечетных размеров;
- нулевые матрицы;
- единичные матрицы;
- матрицы с отрицательными элементами.

3.4 Интерфейс программы

Для программы был разработан консольный интерфейс позволяющий вводить матрицы различных размеров, а также выбирать функцию выполняющую умножение(однопоточная и многопоточная с выбором числа потоков) . На рисунке 3.1 приведен интерфейс программы

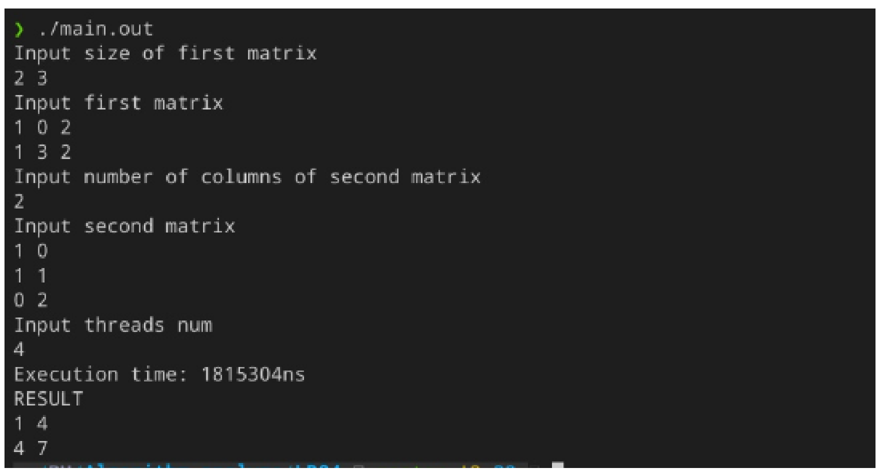


Рис. 3.1: Интерфейс программы

3.5 Сравнительный анализ реализаций

Примем следующую теоретическую модель вычислений для оценки трудоемкости алгоритмов:

- операции единичной стоимости(арифмитические, сравнения, обращение по адресу)

- Циклы определяются формулой:

$$f_{cycle} = f_{init} + f_{cmp} + n \times (f_{inner} + f_{inc} + f_{cmp}), \qquad (3.1)$$

  где
  n - количество итераций цикла,
  f_{cycle} - трудоемкость цикла,
  f_{init} - трудоемкость инициализации,
  f_{cmp} - трудоемкость сравнения,
  f_{inner} - трудоемкость тела цикла,
  f_{inc} - трудоемкость инкремента;

- переход по условию в условном операторе равен нулю, но при этом в расчет входит трудоемкость вычисления условия.

Пусть даны две матрицы A размерностью $m \times n$ и B размерностью $q \times n$, результат их умножения - матрица C размерностью $m \times n$.

3.5.1 Трудоемкость стандартного алгоритма

Опираясь на указанные выше правила, подсчитаем трудоемкость стандартного алгоритма умножения матриц:

$$f_{std} = 2+m\cdot(2+2+n\cdot(2+2+q\cdot(2+\underbrace{6}_{[]}+\underbrace{1}_{\times}+\underbrace{1}_{+}))) = 10nmq+4mn+4m+2 \qquad (3.2)$$

3.5.2 Трудоемкость алгоритма Винограда

Заполнение массива произведений строчных элементов:

$$f_{row} = 2+m\cdot(2+2+1+\frac{q}{2}\cdot(3+\underbrace{6}_{[]}+\underbrace{1}_{=}+\underbrace{2}_{+}+\underbrace{3}_{\times})) = \frac{15}{2}mq+5m+2 \qquad (3.3)$$

Заполнение массива произведений столбцовых элементов производится аналогично, поэтому можно перейти сразу к формуле:

$$f_{col} = \frac{15}{2}qn + 5n + 2. \qquad (3.4)$$

Основной цикл вычисления значений элементов результирующей матрицы:

$$f_{inner} = 2+m\cdot(2+2+n\cdot(2+7+3+\frac{q}{2}]\cdot(3+\underbrace{12}_{[]}+\underbrace{1}_{=}+\underbrace{+}_{5}+\underbrace{5}_{\times} = 13mnq+12qn+4m+2. \quad (3.5)$$

Учет условия перехода при значении q:

$$f_{cond} = 2+\left[\begin{array}{c} 0, \text{если } q \text{ четное} \\ 2+m\cdot(2+2+n\cdot(2\underbrace{8}_{[]}\underbrace{1}_{=}+\underbrace{1}_{+}+\underbrace{1}_{\times}+\underbrace{2}_{-}), \text{иначе} \end{array}\right. = \left[\begin{array}{c} 0, \text{ если } q - \text{четное} \\ 15mn+4m+2 \end{array}\right.$$

Таким образом, получаем формулу вычисления произведения матриц алгоритмом Винограда:

$$f_V = 13mnq+\frac{15}{2}mq+\frac{39}{2}qn+9m+5n+8+\left[\begin{array}{cc} 0 & , \text{если } q - \text{четное} \\ 15mn+4m+2, & \text{иначе} \end{array}\right. \quad (3.7)$$

3.5.3 Трудоемкость оптимизированного алгоритма Винограда

Заполнение массива произведений строчных элементов:

$$f_{row} = 2+m\cdot(2+2+\frac{q}{2}\cdot(2+\underbrace{5}_{[]}+\underbrace{1}_{\times}+\underbrace{1}_{+}+\underbrace{1}_{-=})) = 5mq+4m+2. \quad (3.8)$$

Заполнение массива произведений столбцовых элементов производится аналогично, поэтому можно перейти сразу к формуле:

$$f_{col} = 5nq+4n+2. \quad (3.9)$$

Основной цикл вычисления значений элементов результирующей матрицы для четных q:

$$f_{inner} = \underbrace{3}_{isOdd}+2+m\cdot(2+2+n\cdot(2+2+\underbrace{1}_{if}+\underbrace{4}_{[]}+\underbrace{2}_{=}+\underbrace{1}_{+}+$$
$$+\frac{q}{2}\cdot(2+\underbrace{8}_{[]}+\underbrace{1}_{+=}+\underbrace{4}_{+}+\underbrace{1}_{\times}))) = 8mnq+11mn+4m+5 \quad (3.10)$$

Для нечетных q:

$$f_{inner} = \underbrace{3}_{isOdd} + 2 + m \cdot (2 + 2 + n \cdot (2 + 2 + \underbrace{1}_{if} + \underbrace{8}_{[]} + \underbrace{2}_{=} + \underbrace{1}_{+} + \underbrace{1}_{+=} + \underbrace{1}_{\times} + \\ + \frac{q}{2} \cdot (2 + \underbrace{8}_{[]} + \underbrace{1}_{+=} + \underbrace{4}_{+} + \underbrace{1}_{\times})) = 8mnq + 18mn + 4m + 5 \quad (3.11)$$

Общая трудоемкость:

$$f_V = 8mnq + 5mq + 5nq + 8m + 9 + \left[\begin{array}{c} 11mn, \text{ если } q - \text{четное} \\ 18mn, \text{ иначе} \end{array} \right. \quad (3.12)$$

Выводы:

- В качестве языка программирования выбран C++, так имеет обширную стандартную библиотеку работы с потоками.
- Алгоритмы реализованы в виде подпрограмм, код которых приведен в листингах : 3.1, 3.2
- Разработаны тестовые случаи для проверки корректности работы функций
- Был выбран консольный интерфейс программы, так как прост в реализации и использовании при небольшой функциональности программмы.
- Были расчитаны трудоемкости реализованных алгоритмов

4 Исследовательская часть

Для оценки временной эффективности реализованных функций были произведены замеры времени вычислений на квадратных матрицах разного замера, с разными степенями оптимизации при компиляции. Замеры производились на компьютере с процессором 4 x 2.5MHz.

На рисунках 4.1 4.2 приведены графики зависимости времени вычислений от размера матрицы для матриц четных и нечетных размеров при ключе оптимизации -O0.

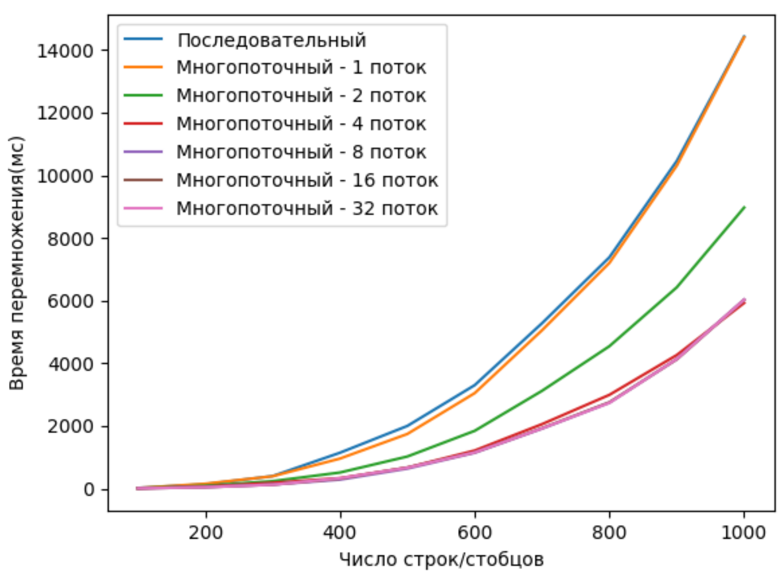


Рис. 4.1: Время выполнения на матрицах четного размера

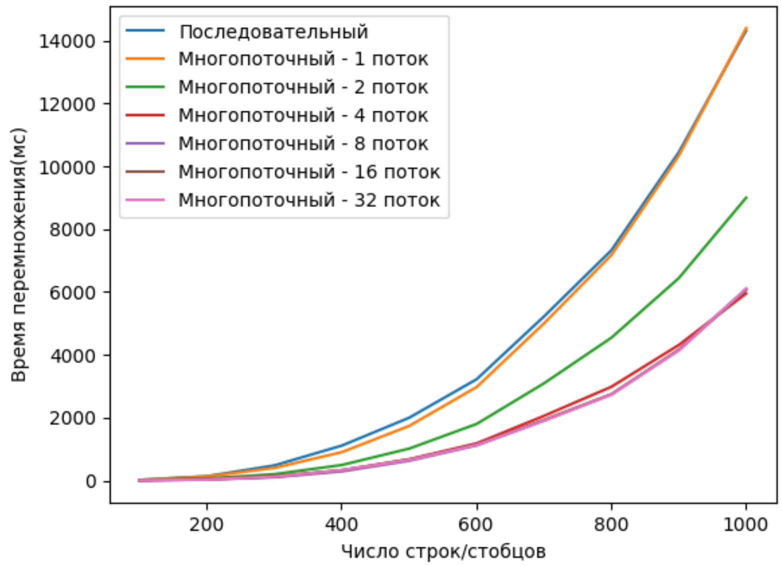


Рис. 4.2: Время выполнения на матрицах нечетного размера

Из приведенных графиков видно, что кривая времени выполения каждого из алгоритма имеет порядок n^3. В тоже время выполнения алгоритмов начинает заметно различаться при размере матриц больше 200. Наименьшее время выполнения имеет функция перемножения с 4 потоками, дальнейшее увеличение числа потоков, не дает выигрыша в скорости.

На рисунках 4.3, 4.4 приведены графики зависимости времени вычислений от размера матриц для матриц четных и нечетных размеров при ключе оптимизации -O2.

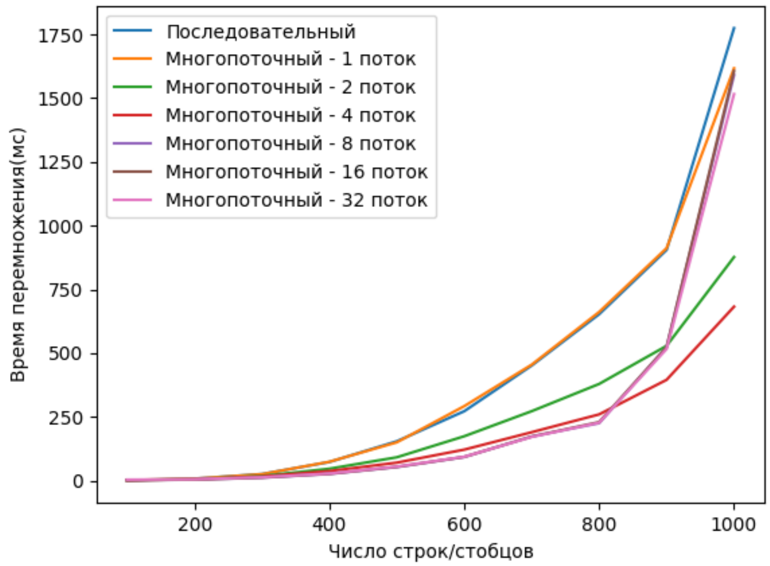


Рис. 4.3: Время выполнения на матрицах четного размера -O2

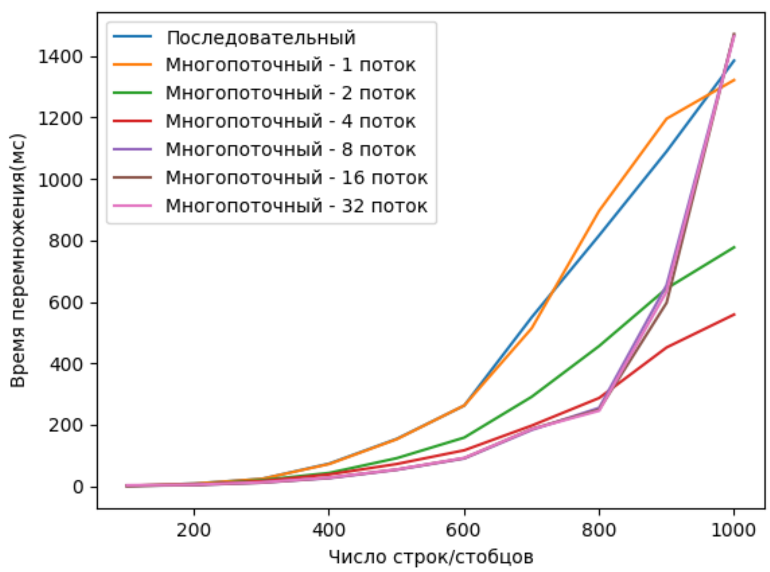


Рис. 4.4: Время выполнения на матрицах нечетного размера -O2

Из приведенных графиков видно, что кривая времении выполнения каждого из алгоритмов также имеет порядок n^3. Но время выполнения в 10 раз меньше чем для реализаций собранных без оптимизаций. Также как и в случае без оптимизаций лучшее время показывает версия с 4 потоками. Но при матрицах размерами больше 800 наблюдается деградация времени выполения версий с числом потоков больше 4, а на размерах матриц больше 900 время их время работы достигает временени работы последовательной реализации.

На рисунках 4.5, 4.6 приведены графики зависимости времени вычислений от размера матриц для матриц четных и нечетных размеров при ключе оптимизации -O3.

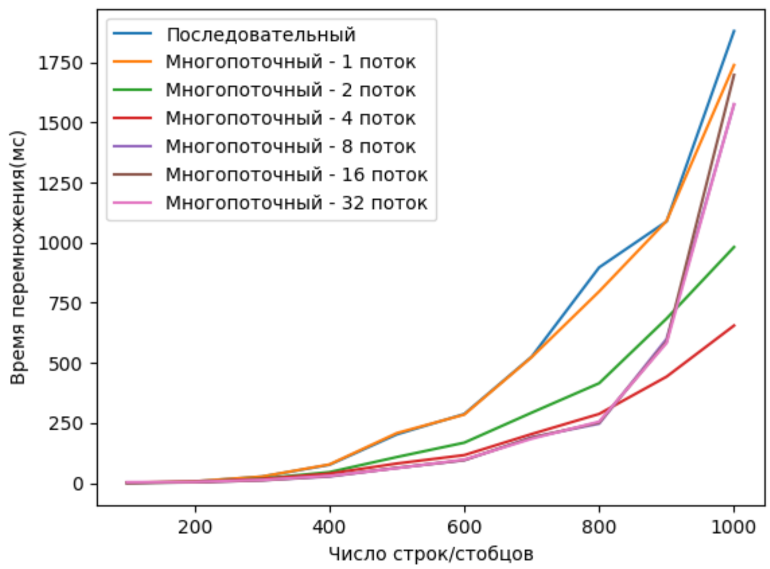


Рис. 4.5: Время выполнения на матрицах четного размера -O3

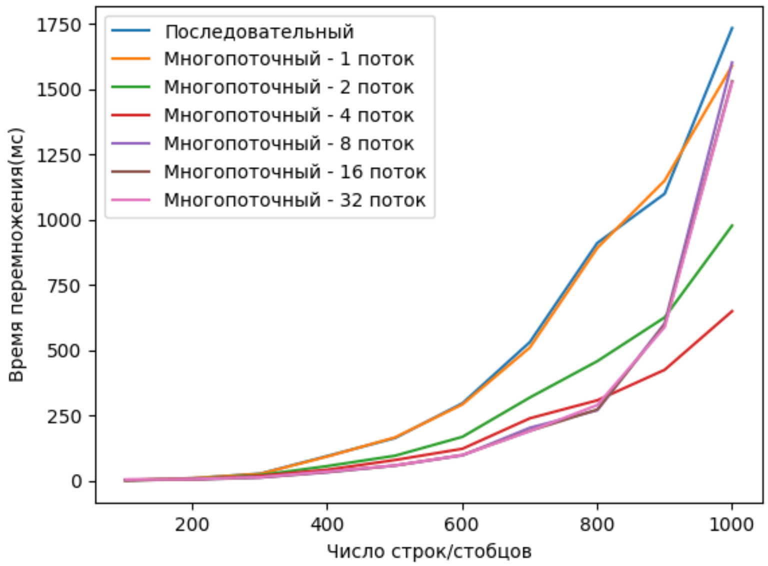


Рис. 4.6: Время выполнения на матрицах четного размера -O3

Из этих графиков видно, что оптимизации -O3 дает небольшой прирост скорости относительно -O2, наиболее быстрой является версия с 4 потоками, а версии с большим числом потоков работают значительно медленнее на матрицах размером больше 800.

Выводы:

- Все алгоритмы имеют временную ассимптотику $O(n^3)$
- Наибольшая производительность достигается при числе потоков, равном числу процессоров, а для оптимизаций -O2, -O3 наблюдается деградации времени перемножения при числе потоков больше 4 и размерах матриц больше 800.
- Время пермножения для самой быстрой из рассмотренных реализаций(-O3 - 4 потока) в среднем в 30 раз меньше времени перемножения для последовательной версии с отключенной оптимизацией.

Заключение

Был изучены и реализованы алгоритмы умножения матриц: классический и алгоритм Винограда. Была произведена оптимизация алгоритма Винограда. Для реализованных алгоритмов были расчитаны трудоемкости.

Экспериментально было подтверждено различие во временной эффективности разных алгоритмов. Обоснован выбор числа потоков для наибольшего быстродействия программы.

В результате исследований пришел к выводу, что оправдано использование многопоточного алгоритма с числом потоков не превышающих число ядер процессора. И что особенно это критично при использовании ключей оптимизации -O2 и -O3

Список литературы

[1] *C++ reference*. URL: `http://www.cplusplus.com/reference/`. (accessed: 02.02.2020).

[2] *Chrono*. URL: `http://www.cplusplus.com/reference/chrono/`. (accessed: 04.02.2020).

[3] Г. Корн. *Алгебра матриц и матричное исчисление*. М: Наука, 1978.

[4] Дж. Макконнелл. *Анализ алгоритмов. Активный обучающий подход*. Информатика и вычислительная биология. М.:Техносфера, 2009.