МГТУ им. Баумана

Лабораторная работа №4

По курсу: "Анализ алгоритмов"

Параллельное умножение матриц

Работу выполнила: Овчинникова Анастасия, ИУ7-55Б

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

Москва, 2019

Оглавление

Введение

Целью данной работы является изучение алгоритмов умножения матриц. Задачи лабораторной работы:

1. реализовать алгоритм умножения матриц Винограда;
2. реализовать параллельный алгоритм умножения матриц Винограда;
3. сравнить время работы алгоритмов Винограда и параллельного алгоритма Винограда;
4. проследить зависимость времени работы параллельного алгоритма от числа параллельных потоков и размера матриц.

Аналитическая часть

Матрицей называют математический объект, эквивалентный двумерному массиву. Матрица представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы. Будем говорить исключительно о матрицах прямоугольной формы (в частных случаях - квадратной). Для матриц определена операция умножения. Матрица, получаемая в результате операции умножения, называется произведнием матриц.

Пусть даны две прямоугольные матрицы A и B размерности m на n и n на l соответсвенно:

$$\begin{bmatrix} a_{11} & ... & a_{1n} \\ ... & ... & ... \\ a_{m1} & ... & a_{mn} \end{bmatrix}$$

$$\begin{bmatrix} b_{11} & ... & b_{1l} \\ ... & ... & ... \\ b_{n1} & ... & b_{nl} \end{bmatrix}$$

Тогда матрица C размерностью m на l:

$$\begin{bmatrix} c_{11} & ... & c_{1l} \\ ... & ... & ... \\ c_{m1} & ... & c_{ml} \end{bmatrix}$$

в которой:
$c_{ij} = \sum_{r=1}^{n} a_{ir} \cdot b_{rj} (i = 0, 1, ..., m-1; j = 0, 1, ..., l-1)$
называется произведением матриц A и B.

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. Можно заметить также, что такое

умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Алгоритм Винограда считается более эффективным благодаря сокращению количества операций умножения.

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$. Их скалярное произведение равно:

$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4$

Это равенство можно переписать в виде:

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

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 столбца второй. Это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

Распараллеливание задачи

Параллельное программирование служит для создания программ, эффективно использующих ресурсы за счет одновременного исполнения кода на нескольких вычислительных узлах. Параллельное программирование является более сложным по сравнению с последовательным как в написании кода, так и в его отладке. Основная сложность при проектировании параллельных программ - обеспечить правильную последовательность взаимодействий между различными вычислительными процессами, а также координацию ресурсов, разделяемых между процессами.

Параллельный алгоритм Винограда

Как было показано в лабораторной работе №2, трудоемкость алгоритма Винограда имеет сложность O(lmn) для умножения размерности m на n и n на l. Чтобы ускорить работу алгоритма, необходимо распараллелить ту его часть, которая содержит три вложенных цикла.

Так как вычисление результата для каждой строки происходит независимо от вычисления результата для других строк, можно распараллелить часть кода, где происходят эти действия. Каждый поток будет выполнять вычисление определенных строк результирующей матрицы.

Таким образом, параллельно будут вычислятся та часть алгоритма Винограда, которая содержит три вложенных цикла и дополнительную проверку на четность размера матрицы.

Вывод

В данном разделе были рассмотрены идеи алгорима Винограда и параллельного алгоритма Винограда. Более подробно эти алгоритмы (в т.ч. их схемы) будут рассмотрены в следующих разделах.

Конструкторская часть

Требования к программе

Требования к вводу:
На вход программе подаются две матрицы и их размерности.

Требования к программе:
Корректное умножение двух матриц.

Схемы алгоритмов

На рисунке 1 представлена схема алгорима Винограда. На рисунке 2 представлена схема параллельного алгорима Винограда. На рисунке 3 изображена схема метода threadMultiply, передаваемого каждому потоку. В схемах используются обозначения, которые были введены при рассмотрении алгоритмов умножения матриц в предыдущем разделе.

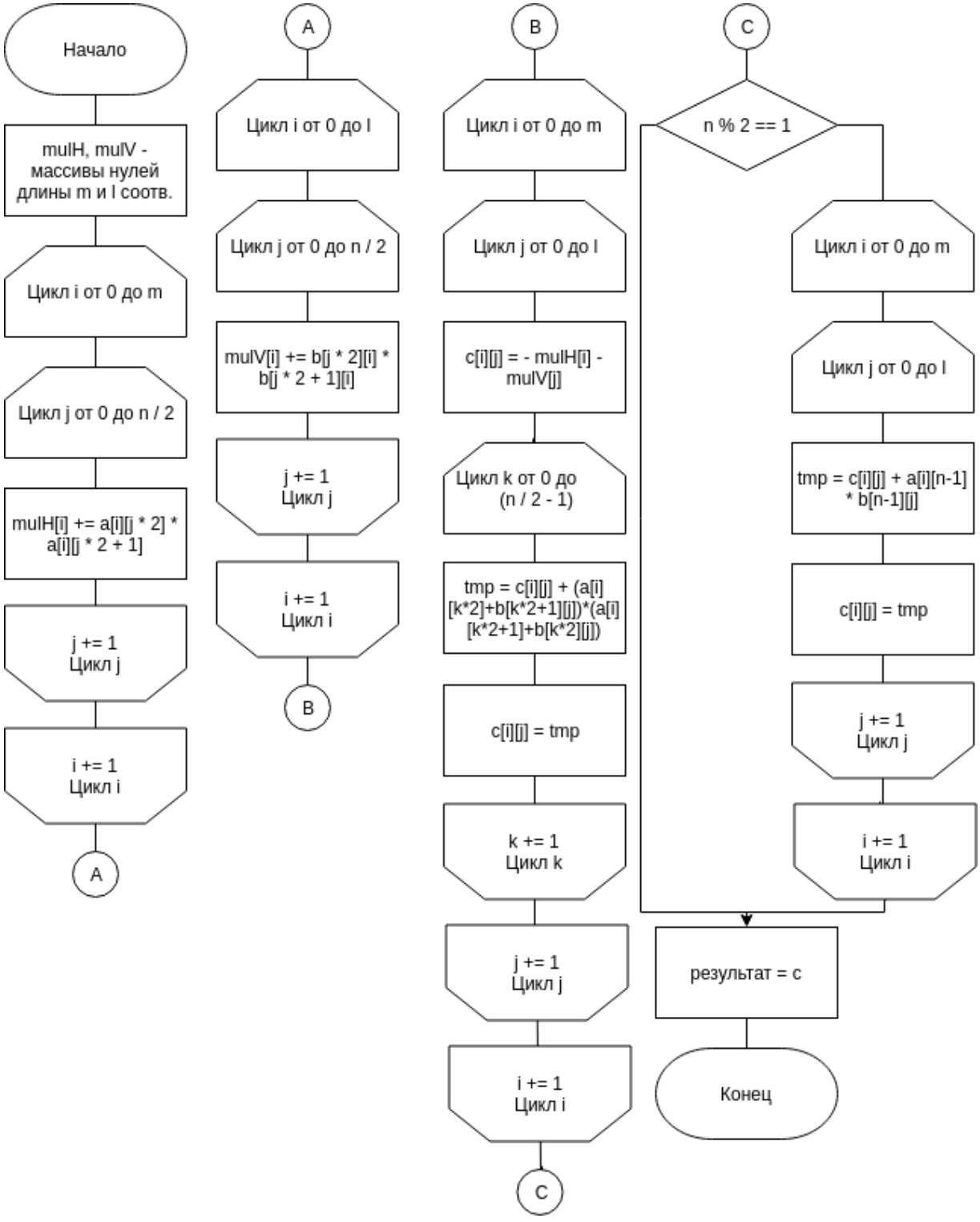

Рис. 1: Схема алгоритма Винограда

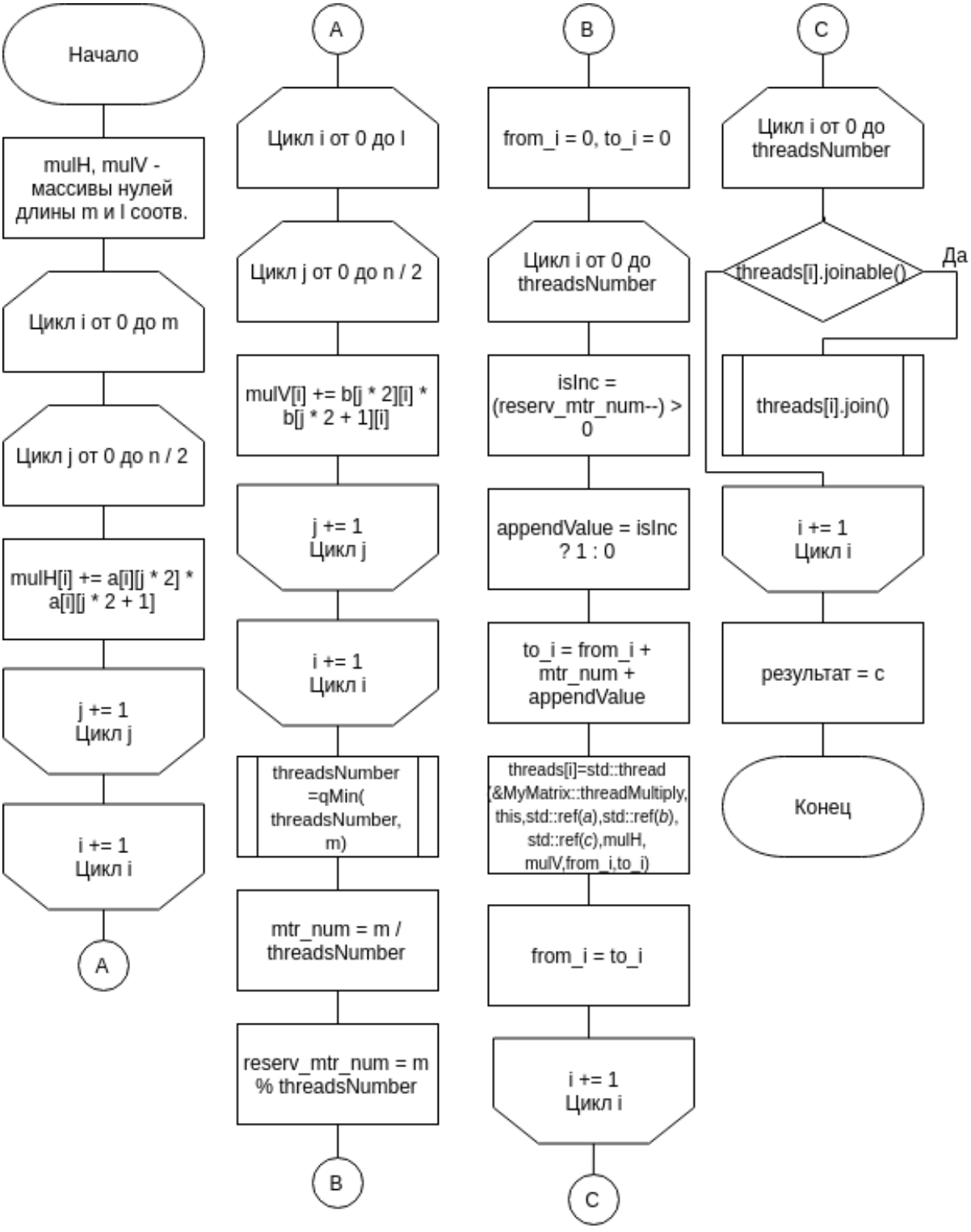

Рис. 2: Схема параллельного алгоритма Винограда

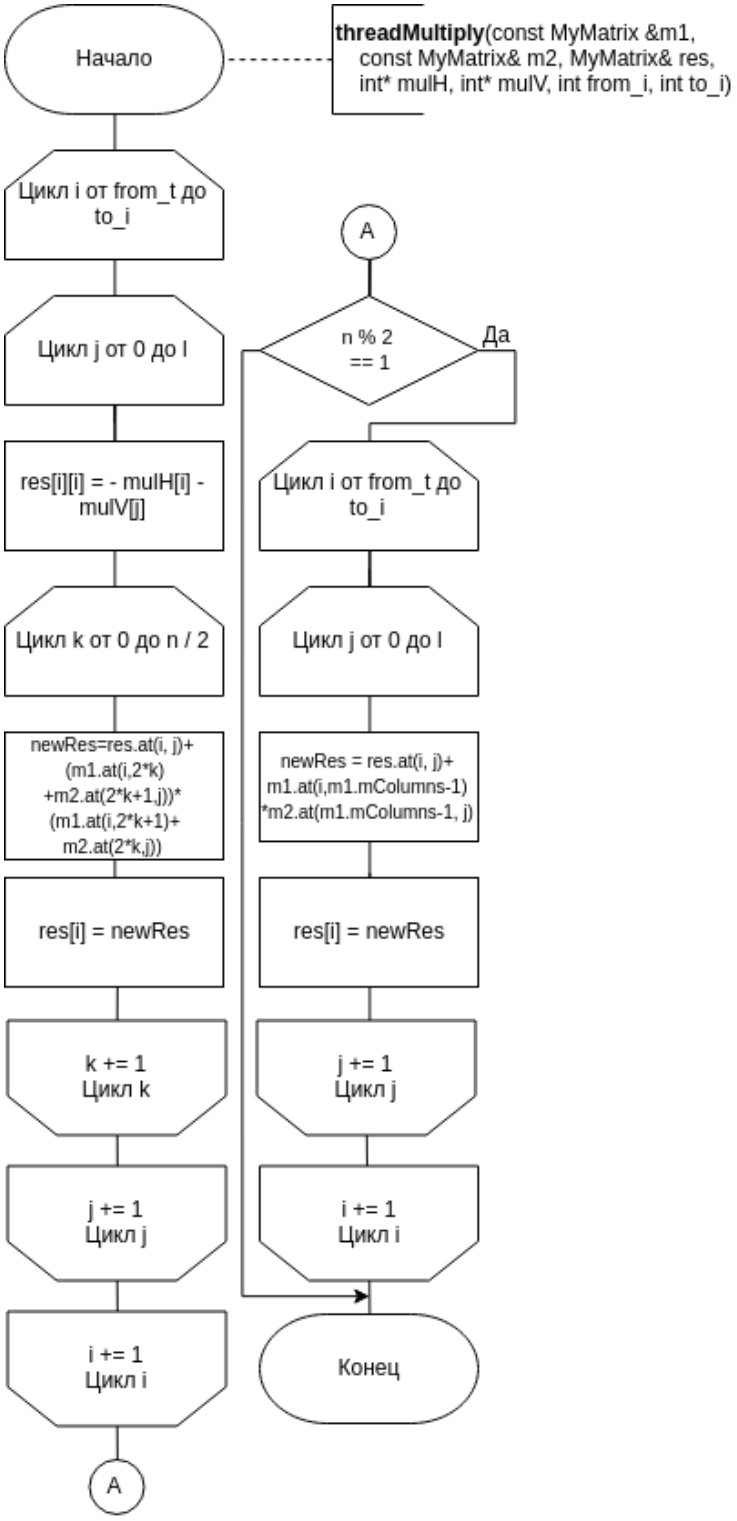

Рис. 3: Схема метода threadMultiply, передаваемого каждому потоку

Технологическая часть

Выбор языка программирования

В качестве языка программирования для реализации программы был выбран язык C++ и фреймворк Qt, потому что:

- язык C++ имеет высокую вычислительную производительность;
- язык C++ поддерживает различные стили программирования;
- в Qt существует удобный инструмент для тестирования - QtTest - который позволяет собирать тесты в группы, собирать результаты выполнения тестов, а также уменьшить дублирование кода при схожих объектах тестирования.

Для замеров времени использовался методы restart() и elapsed() класса QTime. Метод elapsed() возвращает количество миллисекунд, прошедших с момента последнего вызова start() или restart(). Для получения более достоверных результатов оптимизации компилятора были отключены (-O0).

Сведения о модулях программы

Программа состоит из следующих файлов:

- mymatrix.h, mymatrix.cpp - заголовочный файл и файл, в котором расположена реализация алгоритмов умножения матриц;
- main.cpp - главный файл программы, в котором расположена реализация меню;
- testmymatrix.h, testmymatrix.cpp - файл и заголовочный файл, в котором расположена реализация тестов.

Листинги кода алгоритмов

Код алгоритма Винограда представлен в листинге 1. Код параллельного алгоритма Винограда представлен в листинге 2. Код метода, который передается каждому рабочему потоку, представлен в листинге 3.

Листинг 1: Алгоритм Винограда

```
MyMatrix MyMatrix::multiplyVinograd(const MyMatrix &m)
{
    if (mColumns != m.mRows)
        throw std::logic_error("Attempt to multiply matrices of
           different dimensions.");

    MyMatrix result(mRows, m.mColumns, 0);

    if (result.mColumns == 0 && result.mRows == 0)
        return result;

    int* mulH = new int[result.mRows]{0};
    int* mulV = new int[result.mColumns]{0};

    for (int i = 0; i < result.mRows; ++i)
        for (int j = 0; j < m.mRows / 2; ++j)
            mulH[i] += this->at(i, j * 2) * this->at(i, j * 2 + 1);

    for (int i = 0; i < result.mColumns; ++i)
        for (int j = 0; j < m.mRows / 2; ++j)
            mulV[i] += m.at(2 * j, i) * m.at(j * 2 + 1, i);

    for (int i = 0; i < result.mRows; ++i)
        for (int j = 0; j < result.mColumns; ++j)
        {
            result.set(i, j, - mulH[i] - mulV[j]);
            for (int k = 0; k < m.mRows / 2; ++k)
            {
                int newRes = result.at(i, j) +
                             (this->at(i, 2 * k) + m.at(2 * k + 1, j))
                                 *
                             (this->at(i, 2 * k + 1) + m.at(2 * k, j))
                                ;
                result.set(i, j, newRes);
            }
        }

    if (mColumns % 2)
    {
        for (int i = 0; i < mRows; ++i)
            for (int j = 0; j < m.mColumns; ++j)
            {
                int newRes = result.at(i, j) +
                             this->at(i, mColumns - 1) *
                             m.at(mColumns - 1, j);
                result.set(i, j, newRes);
            }
    }
    delete [] mulH;
    delete [] mulV;
    return result;
}
```

Листинг 2: Параллельный алгорим Винограда

```
MyMatrix MyMatrix::multiplyVinograd_multithread(const MyMatrix &m, int threadsNumber)
{
    if (threadsNumber < 0)
        throw std::logic_error("Threads number is under zero.");

    if (mColumns != m.mRows)
        throw std::logic_error("Attempt to multiply matrices of different dimensions.");

    MyMatrix result(mRows, m.mColumns, 0);

    if (result.mColumns == 0 && result.mRows == 0)
        return result;

    int* mulH = new int[result.mRows]{0};
    int* mulV = new int[result.mColumns]{0};

    for (int i = 0; i < result.mRows; ++i)
        for (int j = 0; j < m.mRows / 2; ++j)
            mulH[i] += this->at(i, j * 2) * this->at(i, j * 2 + 1);

    for (int i = 0; i < result.mColumns; ++i)
        for (int j = 0; j < m.mRows / 2; ++j)
            mulV[i] += m.at(2 * j, i) * m.at(j * 2 + 1, i);

    std::vector<std::thread> threads;
    threadsNumber = qMin(threadsNumber, this->mRows);
    int mtr_num = this->mRows / threadsNumber;
    int reserv_mtr_num = this->mRows % threadsNumber;
    int from_i = 0, to_i;
    for (int i = 0; i < threadsNumber; ++i)
    {
        bool isInc = (reserv_mtr_num--) > 0;
        int appendValue = isInc ? 1 : 0;
        to_i = from_i + mtr_num + appendValue;
        threads.push_back(std::thread(&MyMatrix::threadMultiply, this,
            std::ref(*this), std::ref(m), std::ref(result), mulH, mulV
            , from_i, to_i));
        from_i = to_i;
    }

    for (int i = 0; i <  threadsNumber; ++i)
    {
        if (threads.at(i).joinable())
            threads.at(i).join();
    }

    delete [] mulH;
    delete [] mulV;

    return result;
}
```

Листинг 3: Метод threadMultiply, передаваемый каждому потоку

```
void MyMatrix::threadMultiply(const MyMatrix &m1, const MyMatrix& m2,
    MyMatrix& res, int* mulH, int* mulV, int from_i, int to_i)
{
    for (int i = from_i; i < to_i; ++i)
        for (int j = 0; j < m2.mColumns; ++j)
        {
            res.set(i, j, - mulH[i] - mulV[j]);
            for (int k = 0; k < m2.mRows / 2; ++k)
            {
                int newRes = res.at(i, j) +
                             (m1.at(i, 2 * k) + m2.at(2 * k + 1, j)) *
                             (m1.at(i, 2 * k + 1) + m2.at(2 * k, j));
                res.set(i, j, newRes);
            }
        }

    if (m1.mColumns % 2 == 1)
    {
        for (int i = from_i; i < to_i; ++i)
            for (int j = 0; j < m2.mColumns; ++j)
            {
                int newRes = res.at(i, j) +
                             m1.at(i, m1.mColumns - 1) *
                             m2.at(m1.mColumns - 1, j);
                res.set(i, j, newRes);
            }
    }
}
```

Тесты

Тестирование проводилось с помощью модуля QtTest. Для этого был написан класс TestMyMatrix. Сначала каждый алгоритм умножения матриц тестировался по отдельности на заранее заготовленном наборе тестовых данных. После этого генерировались пары случайных квадратных матриц целых чисел в диапазоне от -50 до 50 размером от 0 до 200 с шагом 5 (для каждой размерности матрицы генерировалось 10 случайных матриц). Проводилоь умножение этих пар матриц с помощью алгорима Винограда и параллельного алгорима Винограда и сравнивались результаты работы этих двух алгоритмов. Затем генерировались пары случайных прямоугольных матриц целых чисел в диапазоне от -50 до 50 размером от 0 до 200 с шагом 5 (для каждой размерности матрицы генерировалось 10 случайных матриц). Проводилась умножение этих пар матриц с алгорима Винограда и параллельного алгорима Винограда и сравнивались результаты работы этих двух алгоритмов.

Все написанные тесты были пройдены.

Таблица 1: Набор тестовых данных

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$
$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}1\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$
$\begin{bmatrix}1\end{bmatrix}$	$\begin{bmatrix}1\end{bmatrix}$	$\begin{bmatrix}1\end{bmatrix}$
$\begin{bmatrix}2\end{bmatrix}$	$\begin{bmatrix}2\end{bmatrix}$	$\begin{bmatrix}4\end{bmatrix}$
$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$	$\begin{bmatrix}2 & 2\\2 & 2\end{bmatrix}$
$\begin{bmatrix}2 & 2\\2 & 2\end{bmatrix}$	$\begin{bmatrix}2 & 2\\2 & 2\end{bmatrix}$	$\begin{bmatrix}8 & 8\\8 & 8\end{bmatrix}$
$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$
$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$	$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$
$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$	$\begin{bmatrix}3 & 3\\7 & 7\end{bmatrix}$
$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}4 & 6\\4 & 6\end{bmatrix}$
$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}7 & 10\\15 & 22\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$
$\begin{bmatrix}1 & 1\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$
$\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}$	$\begin{bmatrix}6 & 5 & 4\\3 & 2 & 1\end{bmatrix}$	$\begin{bmatrix}12 & 9 & 6\\30 & 23 & 16\\48 & 37 & 26\end{bmatrix}$
$\begin{bmatrix}1 & 4 & 8\\16 & 32 & 64\end{bmatrix}$	$\begin{bmatrix}1 & 3\\9 & 27\\81 & 243\end{bmatrix}$	$\begin{bmatrix}685 & 2055\\5488 & 16464\end{bmatrix}$
$\begin{bmatrix}1 & 4 & 1\\5 & 9 & 2\\6 & 5 & 3\\5 & 8 & 9\end{bmatrix}$	$\begin{bmatrix}7 & 9 & 3 & 2\\3 & 8 & 4 & 6\\2 & 6 & 4 & 3\end{bmatrix}$	$\begin{bmatrix}21 & 47 & 23 & 29\\66 & 129 & 59 & 70\\63 & 112 & 50 & 51\\77 & 163 & 83 & 85\end{bmatrix}$

Использованный набор тестовых данных приведен в таблице 1.

Исследовательская часть

Постановка эксперимента

В рамках данной работы были проведены следующие эксперименты.

1. Замеры времени работы алгорима Винограда для квадратных матриц размерностей от 100 x 100 до 1000 x 1000 с шагом 100.
2. Замеры времени работы параллельного алгорима Винограда для квадратных матриц размерностей от 100 x 100 до 1000 x 1000 с шагом 100.
3. Замеры времени работы алгорима Винограда для квадратных матриц размерностей от 101 x 101 до 1001 x 1001 с шагом 100.
4. Замеры времени работы параллельного алгорима Винограда для квадратных матриц размерностей от 101 x 101 до 1001 x 1001 с шагом 100.

В рамках каждого эксперимента для матриц каждой размерности проводилось по 10 замеров, и вычислялось среднее время работы конкретного алгоритма на матрицах данной размерности.

Измерения проводились на компьютере HP Pavilion Notebook на базе Intel Core i5-7200U, 2.50 Гц с 6 Гб оперативной памяти и с 4 логическими ядрами под управлением операционной системы Linux Mint.

Сравнительный анализ на материале экспериментальных данных

В таблице 1 представлены результаты замеров времени работы алгоритма Винограда (0, мс) и параллельного алгорима Винограда на одном потоке (1, мс), на двух потоках (2, мс) и т.д. на квадратных матрицах четного размера (размерность матрицы указана в столбце "Р-ть") в миллисекундах. В таблице 2 представлены результаты замеров времени работы алгоритма Винограда (0, мс) и параллельного алгорима Винограда на одном потоке (1, мс), на двух потоках (2, мс) и т.д. на квадратных матрицах нечетного размера (размерность матрицы указана в столбце "Р-ть") в миллисекундах.

Таблица 2: Результаты замеров времени для матриц четного размера

Р-ть	0, мс	1, мс	2, мс	4, мс	8, мс	16, мс
100	16	20	9	8	7	8
200	133	133	66	62	63	61
300	452	457	228	210	216	212
400	1077	1080	541	501	510	506
500	2147	2155	1073	998	995	986
600	3728	3861	2019	1710	1716	1714
700	6063	6115	3042	2855	2871	2745
800	9288	9408	4625	4191	4210	4207
900	13724	13870	7087	6330	6320	6341
1000	19275	19374	10407	9089	9181	9238

Таблица 3: Результаты замеров времени для матриц нечетного размера

Р-ть	0, мс	1, мс	2, мс	4, мс	8, мс	16, мс
101	17	27	9	9	8	8
201	135	136	69	62	64	62
301	458	459	230	215	215	214
401	1096	1116	561	509	519	510
501	2147	2167	1078	994	1006	996
601	3775	3786	1903	1804	1746	1742
701	6181	6224	3074	2788	2828	2882
801	9458	9520	4870	4231	4350	4213
901	14023	14089	7164	6304	6335	6330
1001	19596	19680	10036	9192	9389	9386

На рисунке 4 представлены результаты замеров времени работы алгоритма Винограда (noThreadEven) и параллельного алгорима Винограда на одном потоке (thread1Even), на двух потоках (thread2Even), на четырех потоках (thread4Even), на восьми потоках (thread8Even) и на шестнадцати потоках (thread16Even) на квадратных матрицах четного размера.

На рисунке 5 представлены результаты замеров времени работы алгоритма Винограда (noThreadEven) и параллельного алгорима Винограда на одном потоке (thread1Even), на двух потоках (thread2Even), на четырех потоках (thread4Even), на восьми потоках (thread8Even) и на шестнадцати потоках (thread16Even) на квадратных матрицах нечетного размера.

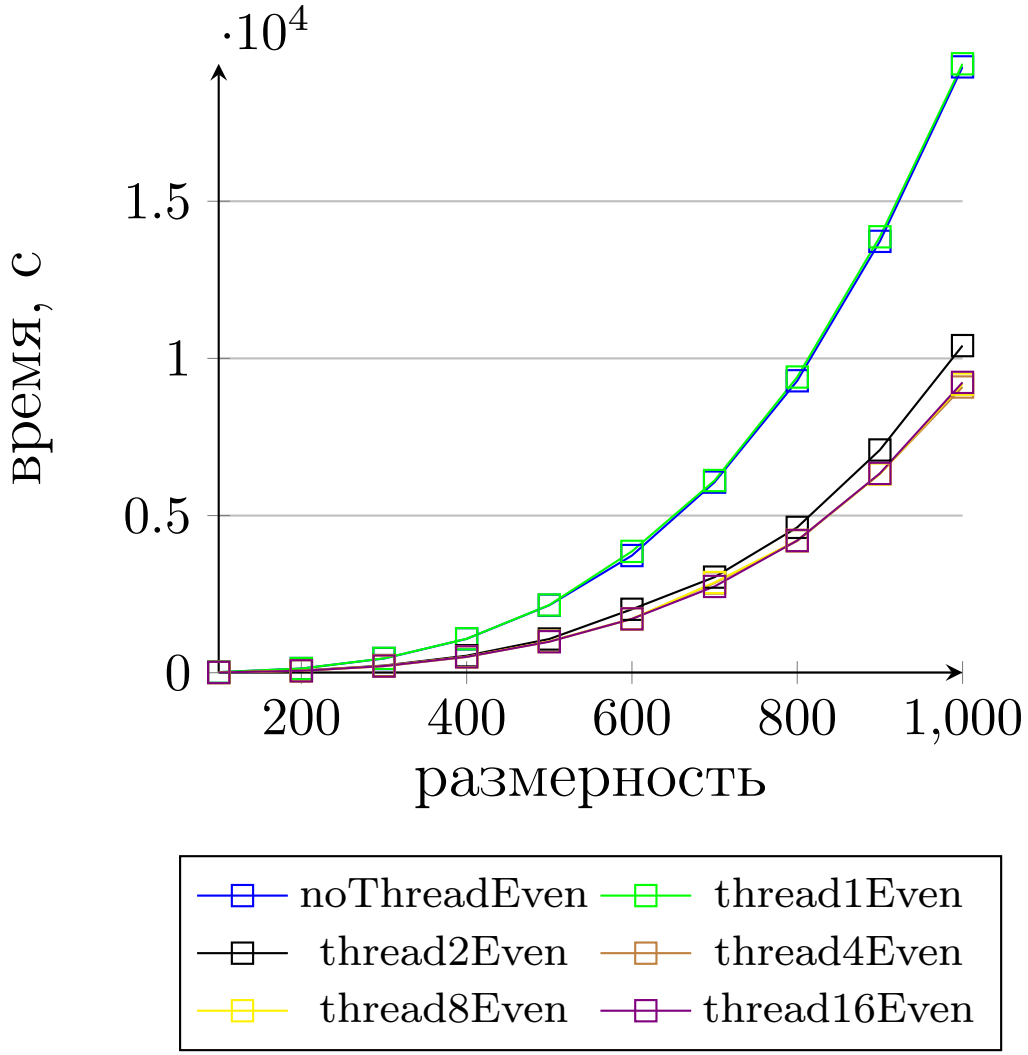

Рис. 4: Результаты замеров времени для матриц четного размера

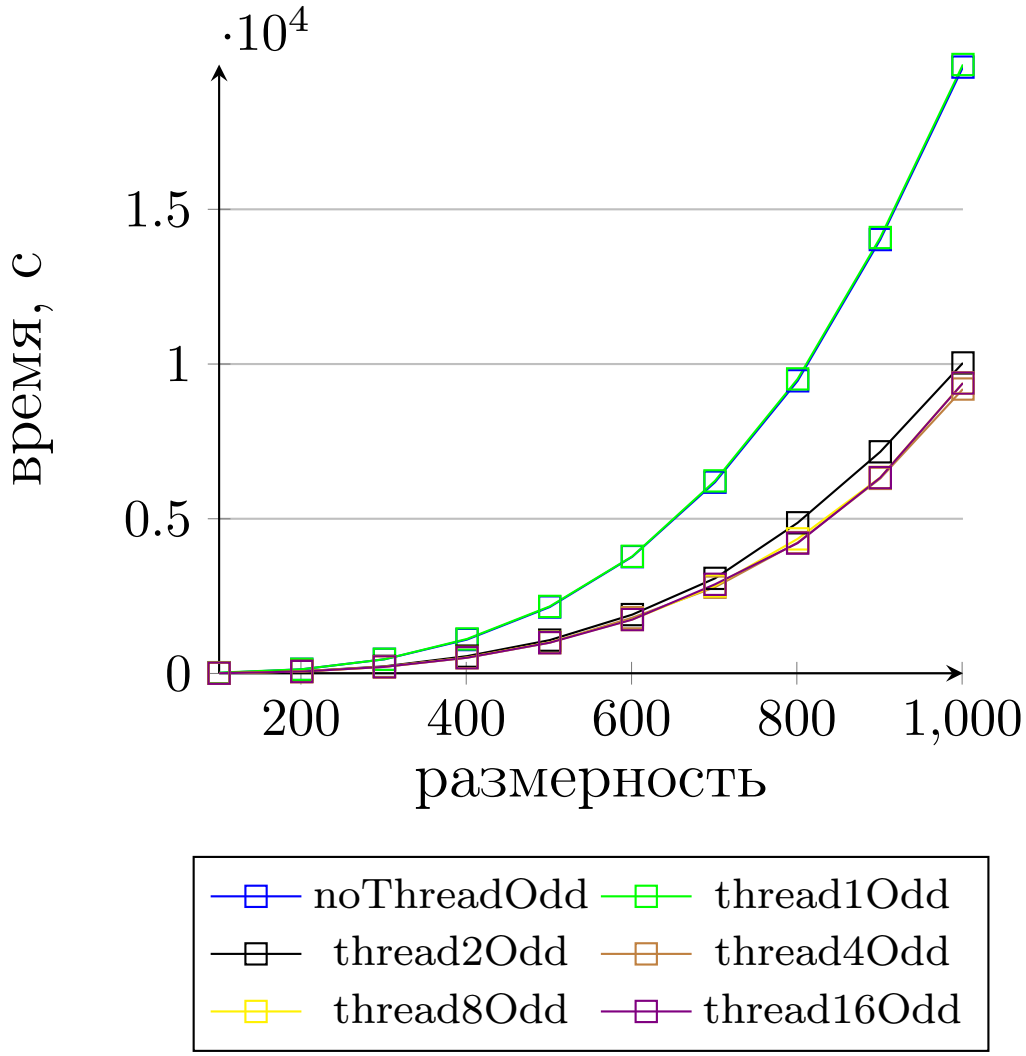

Рис. 5: Результаты замеров времени для матриц нечетного размера

Выводы

В результате проведенных экспериментов было установлено, что:

1. параллельная реализация алгоритма Винограда с одним рабочим потоком незначительно уступает по времени последовательной реализации алгорима Винограда (из-за небольших затрат времени на создание рабочего потока);

2. максимальная производительность достигается при 4-х рабочих потоках, что соответствует количеству логических ядер компьютера, на котором проводились эксперименты;

3. при количестве потоков больше 4-х наблюдается незначительное падение производительности из-за затрат времени на создание рабочих потоков;

4. параллельный алгоритм при количестве рабочих потоков от 2-х до 16-ти работает быстрее, чем последовательный алгоритм.

Заключение

В ходе данной лабораторной работы были реализованы последовательный и параллельный алгоритмы Винограда.

Было выполнено сравнение зависимости скорости работы данных алгоритмов от размерности матрицы и количества рабочих потоков.