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

Дисциплина: Анализ алгоритмов

Лабораторная работа № 2

Умножение матриц

Гибадулин О.Н.
Студент группы ИУ7-52

2019 г.

Содержание

Введение

В связи с возрастающей потребностью решать задачи, связанные с обработкой матриц, такие как расчет новых координат тела в пространстве, растет необходимость в эффективных алгоритмах по работе с ними. Перемножение матриц - одна из стандартных и наиболее используемых операций над матрицами, поэтому существует несколько алгоритмов, позволяющих произвести подобные вычисления.

В данной работе требуется рассмотреть классический алгоритм и алгоритм Винограда для умножения матриц, а также провести их сравнительный анализ.

Цель работы: изучение алгоритма Винограда и классического алгоритма умножения двух матриц.

Задачи работы:

1. разработка и реализация алгоритмов;
2. исследование временных затрат алгоритмов;
3. описание и обоснование полученных результатов.

1 Аналитический раздел

В данном разделе будет описан алгоритм Винограда и стандартный алгоритм перемножения матриц.

1.1 Стандартный алгоритм

Рассмотрим стандартный алгоритм перемножения двух матриц. Пусть есть две матрицы A и B размера $a \cdot b$ и $c \cdot d$ соответственно. Тогда, результатом из умножения будет матрица C размером $a \cdot d$, имеющая вид(1):

$$\begin{bmatrix} c_{11} & c_{12} & \dots & c_{1d} \\ c_{21} & c_{22} & \dots & c_{2d} \\ \dots & \dots & \dots & \dots \\ c_{a1} & c_{a2} & \dots & c_{ad} \end{bmatrix} \tag{1}$$

Каждый элемент матрицы (1) представляет собой скалярное произведение соответствующих строки и столбца исходных матриц.

1.2 Алгоритм Винограда

Алгоритм Винограда это улучшенная версия стандартного алгоритм, где часть вычислений производится заранее. Рассмотрим два вектора:

$$V = (v_1, v_2, v_3, v_4) \tag{2}$$

и

$$W = (w_1, w_2, w_3, w_4) \tag{3}$$

Их скалярное произведение:

$$V * W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4. \tag{4}$$

Это равенство можно переписать в виде:

$$V*W = (v_1+w_2)\cdot(v_2+w_1)+(v_3+w_4)\cdot(v_4+w_3)-v_1\cdot v_2-v_3\cdot v_4-w_1\cdot w_2-w_3\cdot w_4 \tag{5}$$

Несмотря на то, что выражение (5) требует больше вычисления, чем (4), выражение в правой части последнего равенства (5) допускает предварительную обработку. Части этого выражения можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. В этом и заключается алгоритм Винограда.[1]

1.3 Вывод

В данном разделе были описан алгоритм Винограда и стандартный алгоритм перемножения матриц.

2 Конструкторский раздел

В данном разделе будет формализован и описан процесс вычисления произведения двух матриц с помощью диаграммы idef0, а также в соответствии с описанием алгоритмов, приведенными в аналитической части работы, будут рассмотрены схемы алгоритма Винограда и стандартного алгоритма перемножения матриц.

2.1 Формализация процесса

В данном пункте представлена idef0-диаграмма для описания функциональной модели процесса вычисления произведения двух матриц (рис. 2.1.1, рис. 2.1.2).

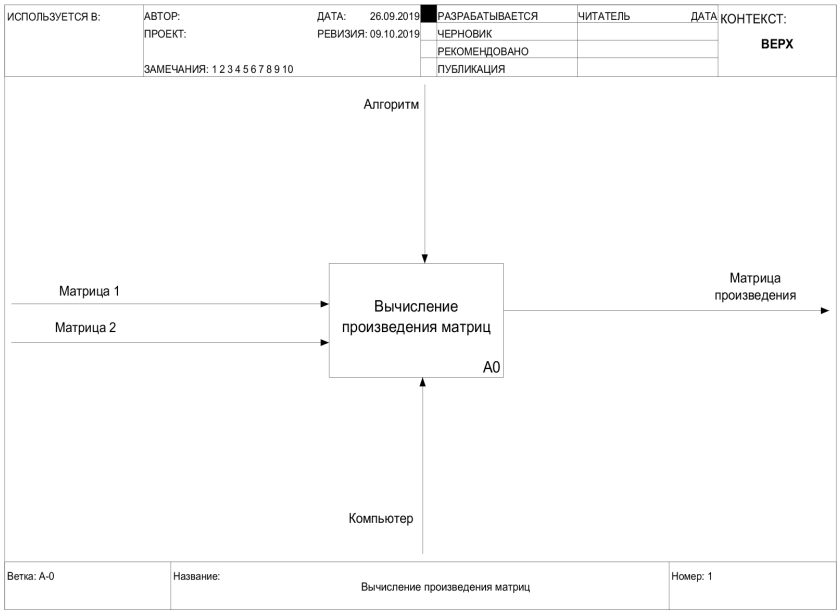

Рисунок 2.1.1. – функциональная схема верхнего уровня процесса умножения матриц

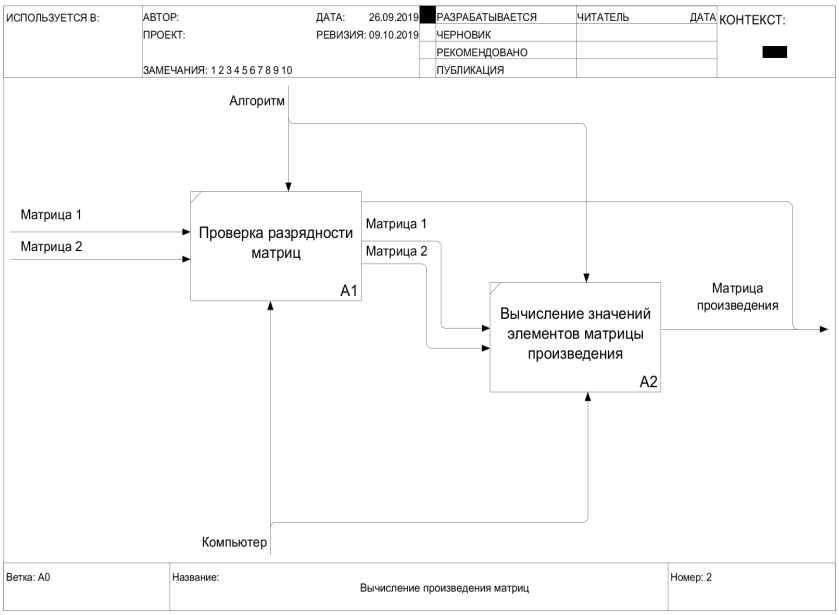

Рисунок 2.1.2. – функциональная схема второго уровня процесса умножения матриц

2.2 Разработка Алгоритмов

В данном пункте представлены схемы алгоритмов перемножения двух матриц. Оба алгоритмы начинаются с проверки их разрядности. Если количество строк или столбцов хотя бы одной из матриц равно 0 или количество столбцов первой не равно количеству строк во второй, то выходим из алгоритма, возвращая нулевую матрицу как матрицу произведения.

В стандартном алгоритме (рис. 2.2.1) для каждой ячейки конечной матрицы (C) вычисляется сумма произведений каждого элемента столбца первой матрицы (A) на соответсвующий элемент строки второй матрицы (B).

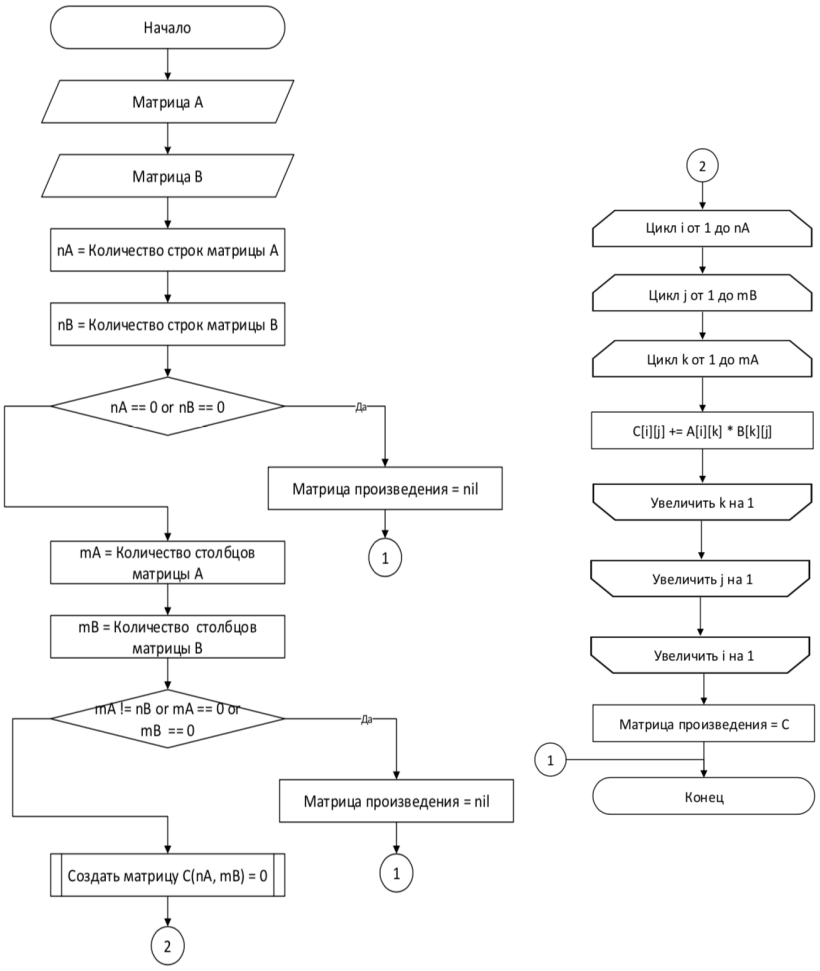

Рисунок 2.2.1. – схема стандартного алгоритма перемножения матриц

В алгоритме Винограда (рис. 2.2.2, рис. 2.2.3) для каждой строки первой матрицы (A) вычисляются и заносятся в массив RF произведения элементов, стоящих на чётных и нечётных позициях. Аналогично вычисляются и заносятся в массив CF произведения значений, находящихся в каждой колонке второй матрицы (B).

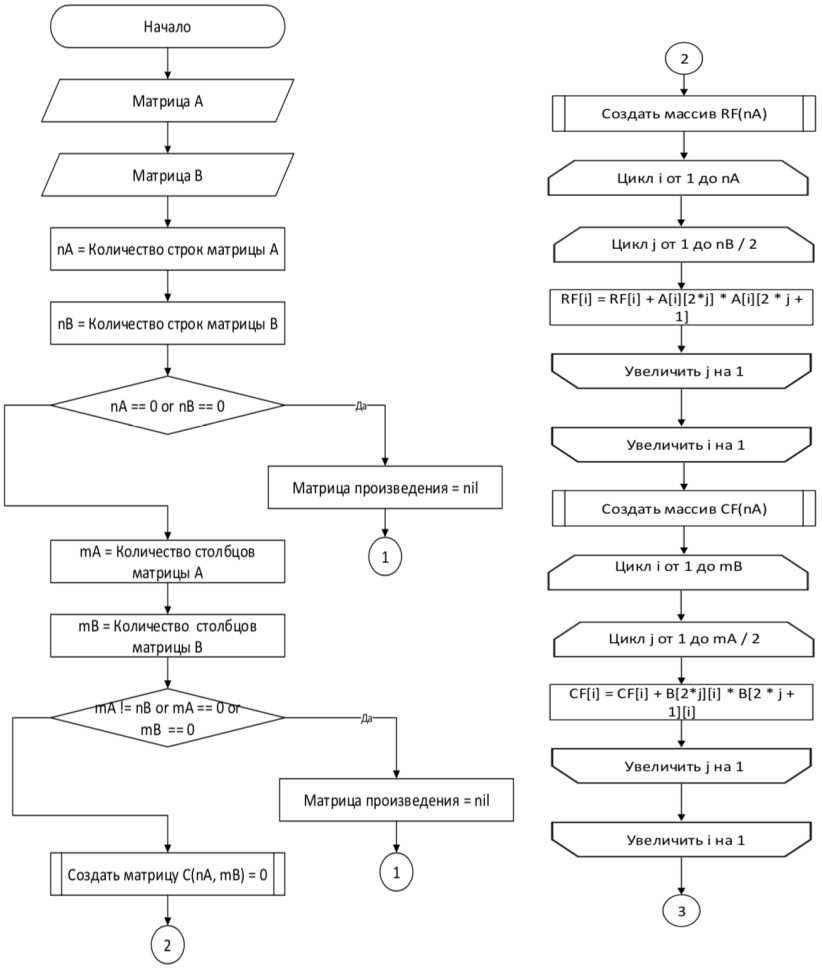

Рисунок 2.2.2. – схема алгоритма Винограда

Далее в алгоритме Винограда для каждой ячейки конечной матрицы (C) вычисляется сумма произведений сумм чётных элементов с нечётными из столбцов первой матрицы (A) и строк второй матрицы (B). Также для матриц с нечётным значением размерности ко всем элементам конечной матрицы прибавляется произведение значений элементов из последнего столбца первой матрицы (A) и значений элементов последней строки второй матрицы (B).

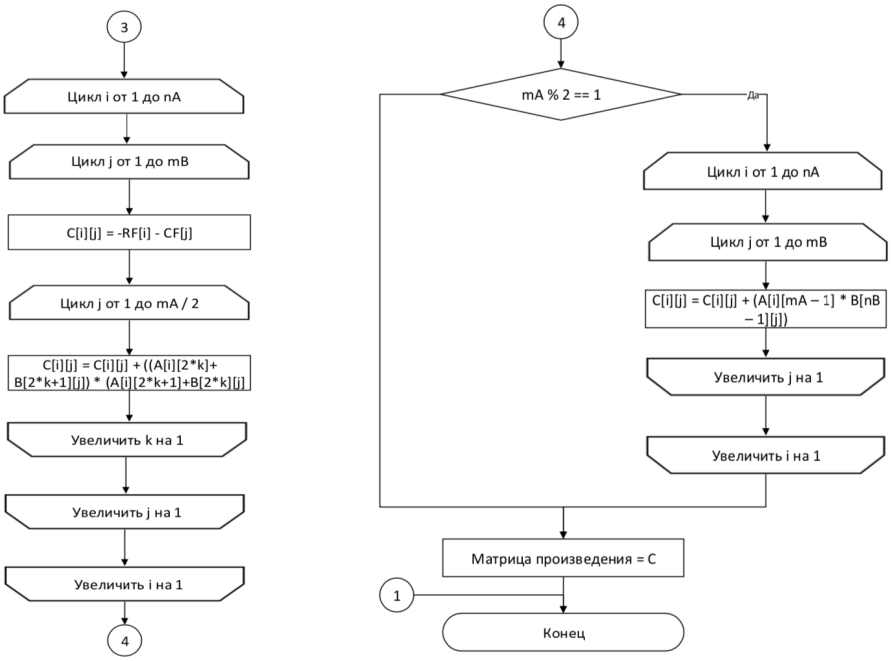

Рисунок 2.2.3. – схема алгоритма Винограда

2.3 Расчет сложности алгоритмов

Модель вычислений

Для корректного расчёта сложности алгоритмов следует описать модель вычислений. Любые арифметические вычисления имеют стоимость 1. Проверка условия имет стоимость 1, переход в блок иначе не имеет стоимости. В циклах происходит начальная инициализация и проверка, стоимость которых составляет 1. В каждой итерации цикла происходит проверка условия и увеличение счётчика, каждый из которых имеет стоимость 1, соответственно, каждая итерация цикла имеет добавочную стоимость 2.

При подсчёте используется следующие наименования:

1. n = nA - количество строк первой матрицы;
2. m = mB - количество столбцов второй матрицы;

3. k = mA - количество столбцов второй матрицы;
4. t = nB / 2 = mA / 2 - количество столбцов/строк первой/второй матрицы.

Классический алгоритм:

- Вычисление матрицы - 2 + n(2 + 2 + m(2 + 2 + k(2 + 9))) = 11nmk + 4nm + 4n + 2
- Худший случай - 11nmk + 4nm + 4n + 2 $\sim$ O(n^3)
- Лучший случай - 11nmk + 4nm + 4n + 2 $\sim$ O(n^3)

Алгоритм Винограда:

- Вычисление RowFactor: 2 + n(2 + 2 + t(3 + 12)) = 15nt + 4n + 2
- Вычисление ColumnFactor: 2 + m(2 + 2 + t(3 + 12)) = 15mt + 4m + 2
- Вычисление матрицы - 2 + n(2 + 2 + m(2 + 2 + t(3 + 23))) = 13nmk + 4nm + 4n + 2
- Вычисления для матрицы с нечётной размерностью 2 + 2 + n(2 + 2 + m(2 + 13)) = 15nm + 4n + 4
- Худший случай - 13mnk + 4nm + 15nt + 8n + 4 $\sim$ On^3)
- Лучший случай - 13mnk + 19nm + 15nt + 12n + 8 $\sim$ O(n^3)

Улучшенный алгоритм Винограда:

- Вычисление RowFactor: 2 + n(2 + 2 + t(2 + 11)) = 13nt + 4n + 2
- Вычисление ColumnFactor: 2 + m(2 + 2 + t(2 + 11)) = 13mt + 4m + 2
- Вычисление матрицы с чётной размерностью - 2 + 2 + n(2 + 2 + m(2 + 2 + t(2 + 21))) = 11nmk + 4nm + 4n + 4
- Вычисление матрицы с нечётной размерностью - 2 + 2 + n(2 + 2 + m(2 + 15 + 2 + t(2 + 21))) = 11nmk + 19nm + 4n + 4
- Худший случай - 11nmk + 4nm + 13nt + 13mt + 8n + 6 $\sim$ O(n^3)
- Лучший случай - 11nmk + 19nm + 13nt + 13mt + 8n + 6 $\sim$ O(n^3)

2.4 Вывод

В данном разделе был формализован и описан процесс вычисления произведения двух матриц, а также рассмотрены схемы алгоритма Винограда и стандартного алгоритма перемножения матриц.

3 Технологический раздел

В данном разделе будут рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы.

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать 3 алгоритма перемножения двух матриц – стандартный, Винограда и улучшенный Винограда. Пользователь должен иметь возможность произвести вычисления для матриц, размер которых он вводит, а также иметь возможность сравнить время работы этих алгоритмов.

3.2 Средства реализации

Для реализации поставленной задачи был использован язык программирования C++[2]. Проет был выполнен в среде XCode[3]. Для измерения процессорного времени была использована ассемблерная инструкция rdtsc[4].

3.3 Листинг кода

На основе схем алгоритмов, представленных в конструкторском разделе, в соответствии с указанными требованиями к реализации с использованием средств языка C++ было разработано программное обеспечение, содержащее реализации выбранных алгоритмов. В данном пункте приведён листиннг этих реализаций.

Листинг 1. Код стандартного алгоритма

```
const size_t nA = matrixA.size();
const size_t nB = matrixB.size();

if (nA == 0 || nB == 0)
{
cout << "Wrong dimension of the matrix" << endl;
return vector<vector<int>>();
}

const size_t mA = matrixA[0].size();
const size_t mB = matrixB[0].size();

if (mA != nB || mA == 0 || mB == 0)
{
cout << "Wrong dimension of the matrix" << endl;
return vector<vector<int>>();
}
```

```
vector<vector<int>> matrixRes(nA);
for (int i = 0; i < nA; ++i)
{
matrixRes[i].assign(mB, 0);
}

for (size_t row = 0; row < nA; ++row)
{
for (size_t column = 0; column < mB; ++column)
{
for (size_t k = 0; k < mA; ++k)
{
matrixRes[row][column] += matrixA[row][k] *
    matrixB[k][column];
}
}
}

return matrixRes;
```

Листинг 2. Код алгоритма Винограда

```
const size_t nA = matrixA.size();
const size_t nB = matrixB.size();
if (nA == 0 || nB == 0)
{
cout << "Wrong dimension of the matrix" << endl;
return vector<vector<int>>();
}

const size_t mA = matrixA[0].size();
const size_t mB = matrixB[0].size();
if (mA != nB || mA == 0 || mB == 0)
{
cout << "Wrong dimension of the matrix" << endl;
return vector<vector<int>>();
}

vector<vector<int>> matrixRes(nA);
for (int i = 0; i < nA; ++i)
{
matrixRes[i].assign(mB, 0);
}

vector<int> rowFactor(nA);
for (int i = 0; i < nA; ++i)
{
for (int j = 0; j < nB / 2; ++j)
{
```

```
        rowFactor[i] = rowFactor[i] + (matrixA[i][2 * j]
            * matrixA[i][2 * j + 1]);
        }
        }

        vector<int> columnFactor(mB);
        for (int i = 0; i < mB; ++i)
        {
        for (int j = 0; j < mA / 2; ++j)
        {
        columnFactor[i] = columnFactor[i] + (matrixB[2 *
            j][i] * matrixB[2 * j + 1][i]);
        }
        }

        for (int i = 0; i < nA; ++i)
        {
        for (int j = 0; j < mB; ++j)
        {
        matrixRes[i][j] = -rowFactor[i] - columnFactor[j
            ];
        for (int k = 0; k < mA / 2; ++k)
        {
        matrixRes[i][j] = matrixRes[i][j] + ((matrixA[i
            ][2 * k] + matrixB[2 * k + 1][j]) * (matrixA[i
            ][2 * k + 1] + matrixB[2 * k][j]));
        }
        }
        }

        if (mA % 2 != 0)
        {
        for (int i = 0; i < nA; ++i)
        {
        for (int j = 0; j < mB; ++j)
        {
        matrixRes[i][j] = matrixRes[i][j] + (matrixA[i][
            mA - 1] * matrixB[nB - 1][j]);
        }
        }
        }

        return matrixRes;
```

Листинг 3. Код улучшенного алгоритма Винограда

```
        const size_t nA = matrixA.size();
        const size_t nB = matrixB.size();
        if (nA == 0 || nB == 0)
        {
```

```
cout << "Wrong dimension of the matrix" << endl;
return vector<vector<int>>();
}

const size_t mA = matrixA[0].size();
const size_t mB = matrixB[0].size();
if (mA != nB || mA == 0 || mB == 0)
{
cout << "Wrong dimension of the matrix" << endl;
return vector<vector<int>>();
}

vector<vector<int>> matrixRes(nA);
for (int i = 0; i < nA; ++i)
{
matrixRes[i].assign(mB, 0);
}

const size_t d = nB / 2;

vector<int> rowFactor(nA);
for (int i = 0; i < nA; ++i)
{
for (int j = 0; j < d; ++j)
{
rowFactor[i] += (matrixA[i][(j<<1)] * matrixA[i
    ][(j<<1)|1]);
}
}

vector<int> columnFactor(mB);
for (int i = 0; i < mB; ++i)
{
for (int j = 0; j < d; ++j)
{
columnFactor[i] += (matrixB[(j << 1)][i] *
    matrixB[(j << 1) | 1][i]);
}
}

if (nB % 2 == 0)
{
for (int i = 0; i < nA; ++i)
{
for (int j = 0; j < mB; ++j)
{
matrixRes[i][j] = -rowFactor[i] - columnFactor[j
    ];
for (int k = 0; k < d; ++k)
{
```

```
        matrixRes[i][j] += ((matrixA[i][(k << 1)] +
            matrixB[(k << 1) | 1][j])* (matrixA[i][(k <<
            1) | 1] + matrixB[(k << 1)][j]));
        }
        }
        }
        }
        else
        {
        for (int i = 0; i < nA; ++i)
        {
        for (int j = 0; j < mB; ++j)
        {
        matrixRes[i][j] = -rowFactor[i] - columnFactor[j]
            + (matrixA[i][nB - 1] * matrixB[nB - 1][j]);
        for (int k = 0; k < d; ++k)
        {
        matrixRes[i][j] += ((matrixA[i][(k << 1)] +
            matrixB[(k << 1) | 1][j]) * (matrixA[i][(k <<
            1) | 1] + matrixB[(k << 1)][j]));
        }
        }
        }
        }

        return matrixRes;
```

3.4 Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки, а также был представлен листинг реализаций выбранных алгоритмов.

4 Экспериментальный раздел

В данном разделе будет проведено исследование временных затрат разработанного программного обеспечения, вместе с подробным сравнительным анализом реализованных алгоритмов на основе экспериментальных данных.

4.1 Сравнительное исследование

Замеры времени выполнялись на квадратных матрицах размера от 100x100 до 1000x1000 с шагом 100 для четной совпадающей размерности матриц (табл. 4.1.1, рис. 4.1.1), и от 101x101 до 1001x1001 для нечетной (табл. 4.1.2, рис. 4.1.2). Числа в матрицах генерировались случайным образом. Все замеры были произведены на процессоре 2,7 GHz Intel Core i5 с паматью 8 ГБ 1867 MHz DDR3.

Таблица 4.1.1. Сравнение времени работы алгоритмов в тактах процессора для чётной размерности матриц

Размерность матрицы	Стандартный	Винограда	Улучшенный Винограда
100	52034082	49414527	43259365
200	447583410	389320119	316222732
300	1415892680	1405172852	1281383494
400	4128338247	4445154675	3398253821
500	7960538147	7762218724	6867809251
600	13859504924	14474345166	12646795828
700	28854830638	26278279010	22268806196
800	45547351819	43995713775	37176720077
900	86996857986	85336754062	73473300350
1000	111941707495	110928724001	95510769695

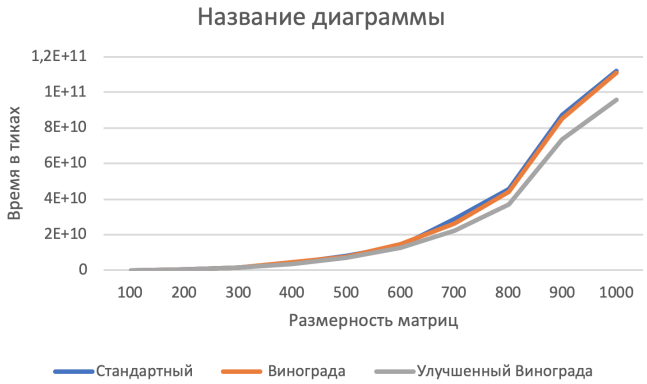

Рисунок 4.1.1. – график зависимости времени работы алгоритмов для чётной размерности матриц

Таблица 4.1.2. Сравнение времени работы алгоритмов в тактах процессора для нечётной размерности матриц

Размерность матрицы	Стандартный	Виноградa	Улучшенный Виноградa
101	49880669	48262417	43648946
201	411923317	400398552	324912995
301	1658989459	2007837600	1740148774
401	4818884551	4796086190	4080594021
501	8354925265	8565477002	7330399626
601	15401080668	14673340612	12858391267
701	25217048226	27080987646	23027375044
801	34700307829	35618403773	32345278321
901	94207348398	93458447718	115969050293
1001	120365201656	118430073117	115969050293

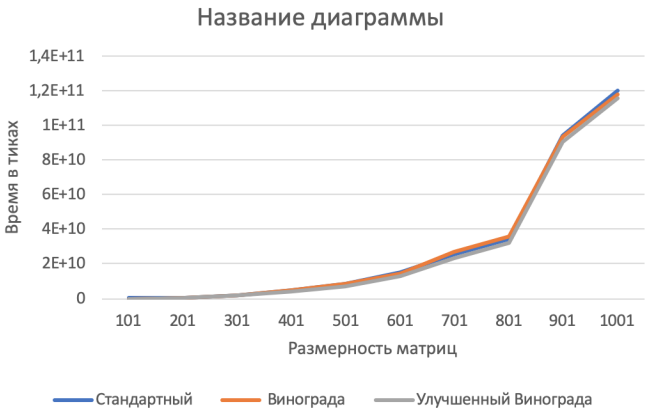

Рисунок 4.1.2. – график зависимости времени работы алгоритмов для нечётной размерности матриц

Из данных графиков можно сделать вывод, что алгоритм Винограда и классический алгоритм работают приерно одинаково. Результаты показали, что оптимизация улучшила показатели алгоритма Винограда по времени.

Так же сотоит отметить, что в классическом алгоритме разница времени между выполнением умножения матриц размером, отличающимся на единицу, незначительна, тогда как в алгоритме Винограда разница существенна из-за дополнительной проверки на чётность размерности с последующими вычсилениями при нёчетном количестве элементов.

4.2 Вывод

В данном разделе было проведено исследование временных затрат разработанного программного обеспечения, вместе с подробным сравнительным анализом реализованных алгоритмов на основе экспериментальных данных.

Заключение

В ходе выполнения данной лабораторной работы были изучены и реализованы алгоритмы перменожения матриц. В аналитическом разделе было дано описание стандартного алгоритма и алгоритма Винограда. В конструкторском разделе был формализован и описан процесс вычисления пороизвдения двух матриц, разработаны алгоритмы. В технологическом разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы. В экспериментальном разеде было проведено исследование временных затрат.

Список литературы

[1] Кибернетический сборник. Новая серия. Вып. 25. Сб. статей 1983 — 1985 гг.: Пер. с англ. — М.: Мир, 1988 — В.Б. Алекссев. Сложность умножения матриц. Обзор.

[2] ISO/IEC JTC1 SC22 WG21 N 3690 «Programming Languages — C++» [Электронный ресурс]. – Режим доступа: https://devdocs.io/cpp/, свободный. (Дата обращения: 29.09.2019 г.)

[3] Apple «Apple Developer Documentation» [Электронный ресурс]. – Режим доступа: https://developer.apple.com/documentation/, свободный. (Дата обращения: 29.09.2019 г.)

[4] Microsoft «rdtsc» [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/cpp/intrinsics/rdtsc?view=vs-2019, свободный. (Дата обращения: 29.09.2019 г.)