Лабораторная работа №2
По курсу: "Анализ алгоритмов"
По теме: "Умножение матриц"

Студент: Кондрашова О.П.
Группа: ИУ7-55Б
Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Содержание

Введение **3**

1 Аналитическая часть **4**
1.1 Описание алгоритмов . . . 4
1.2 Стандартный алгоритм . . . 4
1.3 Алгоритм Винограда . . . 5
1.4 Модель вычислений . . . 5
1.5 Вывод . . . 5

2 Конструкторская часть **7**
2.1 Схемы алгоритмов . . . 7
2.2 Вывод . . . 9

3 Технологическая часть **10**
3.1 Требования к программному обеспечению . . . 10
3.2 Средства реализации . . . 10
3.3 Оценка трудоемкости . . . 11
3.4 Листинги функций . . . 12
3.5 Оптимизация алгоритма Винограда . . . 15
3.6 Вывод . . . 16

4 Исследовательская часть **17**
4.1 Постановка эксперимента . . . 17
4.2 Матрицы четного размера . . . 17
4.3 Матрицы нечетного размера . . . 18
4.4 Вывод . . . 18

Заключение **19**

Список литературы **20**

Введение

В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

Цели работы:

1. изучение трудоемкости алгоритмов умножения матриц и получение ее оценок;
2. получение навыка оптимизации алгоритма с целью снижения трудоемкости его выполнения на примере решения задачи умножения матриц;
3. экспериментальное подтверждение оценок трудоемкости.

1 Аналитическая часть

В данной части будут рассмотрены теоретические основы алгоритмова, а также составлена модель для вычисления трудоемкости..

1.1 Описание алгоритмов

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. 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 частности, умножение всегда выполнимо, если оба сомножителя — квадратные матрицы одного и того же порядка. Таким образом, из существования произведения AB вовсе не следует существование произведения BA.

Эти алгоритмы активно применяются во всех областях, применяющие линейную алгебру, такие как:

1. экономика;
2. физика;
3. компьютерная графика.

1.2 Стандартный алгоритм

Матрицей называют математический объект, эквивалентный двумерному массиву. Матрица является прямоугольной (в частных случаях – квадратной) таблицей, совокупностью строк и столбцов, на пересечении которых находятся элементы матрицы. Количество строк и столбцов является размерностью матриц. Для матриц определена операция умножения.

Пусть даны две матрицы – матрица A размером m на n:

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

и матрица B размером n на l:

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

Матрица C:

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

При этом:

$c_{i,j} = \sum\limits_{r=1}^{n} a_{i,r} \cdot b_{r,j}$ называется произведением матриц A и B.

Стандартный алгоритм умножений матриц A и B выполняет $m \cdot n \cdot l$ умножений и $m \cdot (n-1) \cdot l$ сложений.

Несмотря на очевидность и простоту, данный алгоритм не является «минимальным». Для более эффективного умножения матриц был разработан алгоритм Винограда.

1.3 Алгоритм Винограда

Очевидно, что каждый $c_{i,j}$ в результирующей матрице является скалярным произведением соответствующих столбца и строки. Такое умножение допускает предварительное вычисление части результата. Имеются два вектора: $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$.

Стандартное» перемножение:
$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4$

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

Данный алгоритм 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.

1.4 Модель вычислений

В рамках данной работы используется следующая модель вычислений:

1. Базовые операции имеют трудоемкость 1 (<, >, =, <=, =>, ==, +, -, *, /);

2. Оператор if имеет трудоемкость, равную трудоемкости тела оператора;

3. Оператор for имеет трудоемкость $F_{for} = 2 + N \cdot (F_{body} + F_{chek})$, где F_{body} – трудоемкость операций в теле цикла, а F_{check} – трудоемкость проверки условия.

1.5 Вывод

Были рассмотрены поверхностно алгоритмы классического умножения матриц и алгоритм Винограда, принципиальная разница которого — наличие предварительной обработки, а также уменьшение количества операций умножения.

2 Конструкторская часть

В данной части будут изложены математические основы алгоритмов.

2.1 Схемы алгоритмов

Схема стандартного алгоритма:

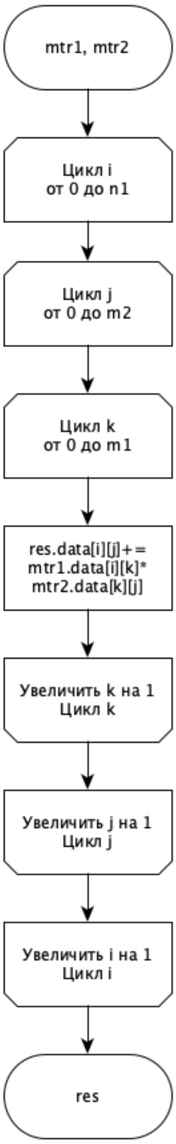

Рис. 1: Стандартный алгоритм

Схема алгоритма Винограда:

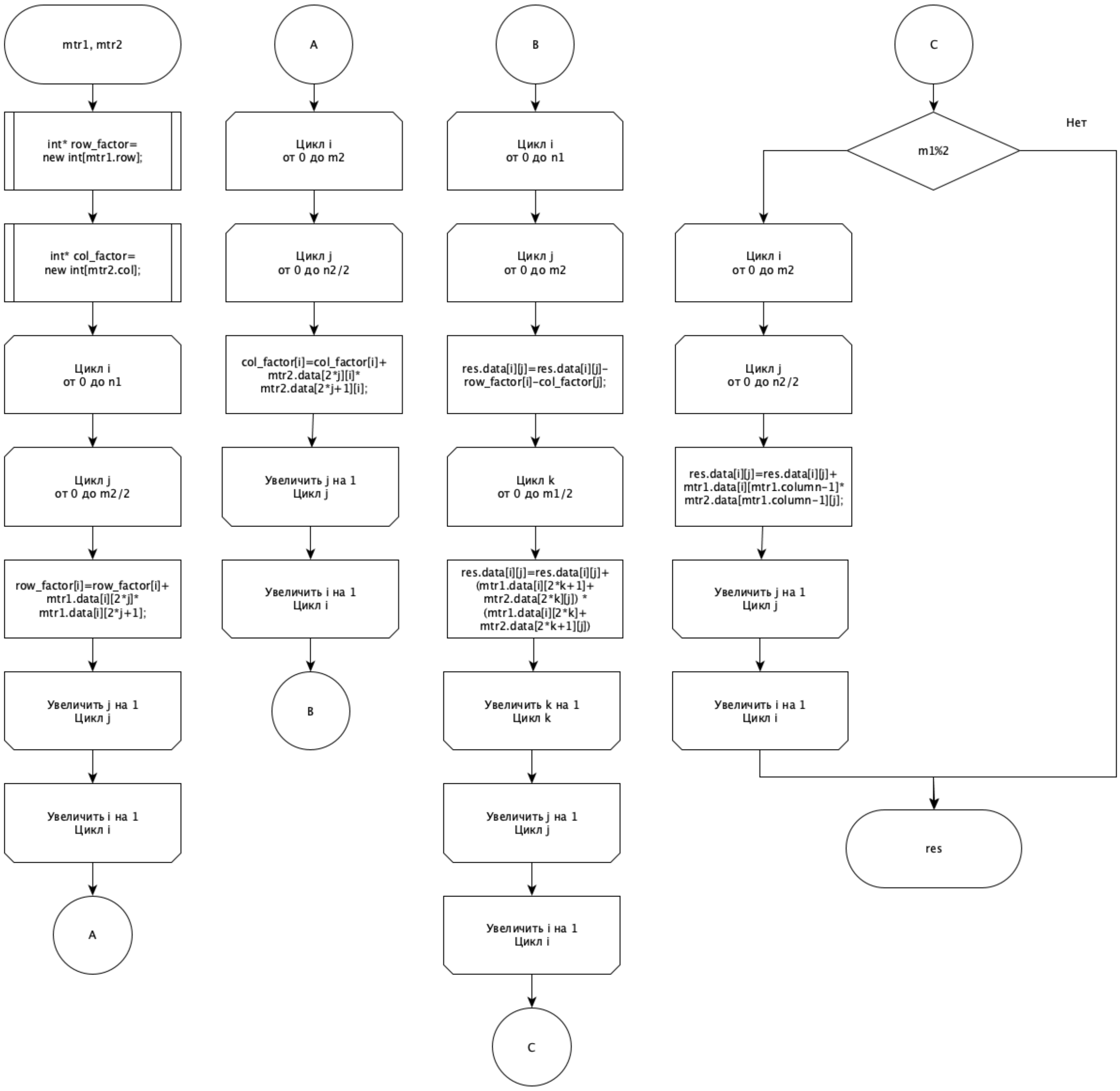

Рис. 2: Алгоритм Винограда

Схема оптимизированного алгоритма Виноградова:

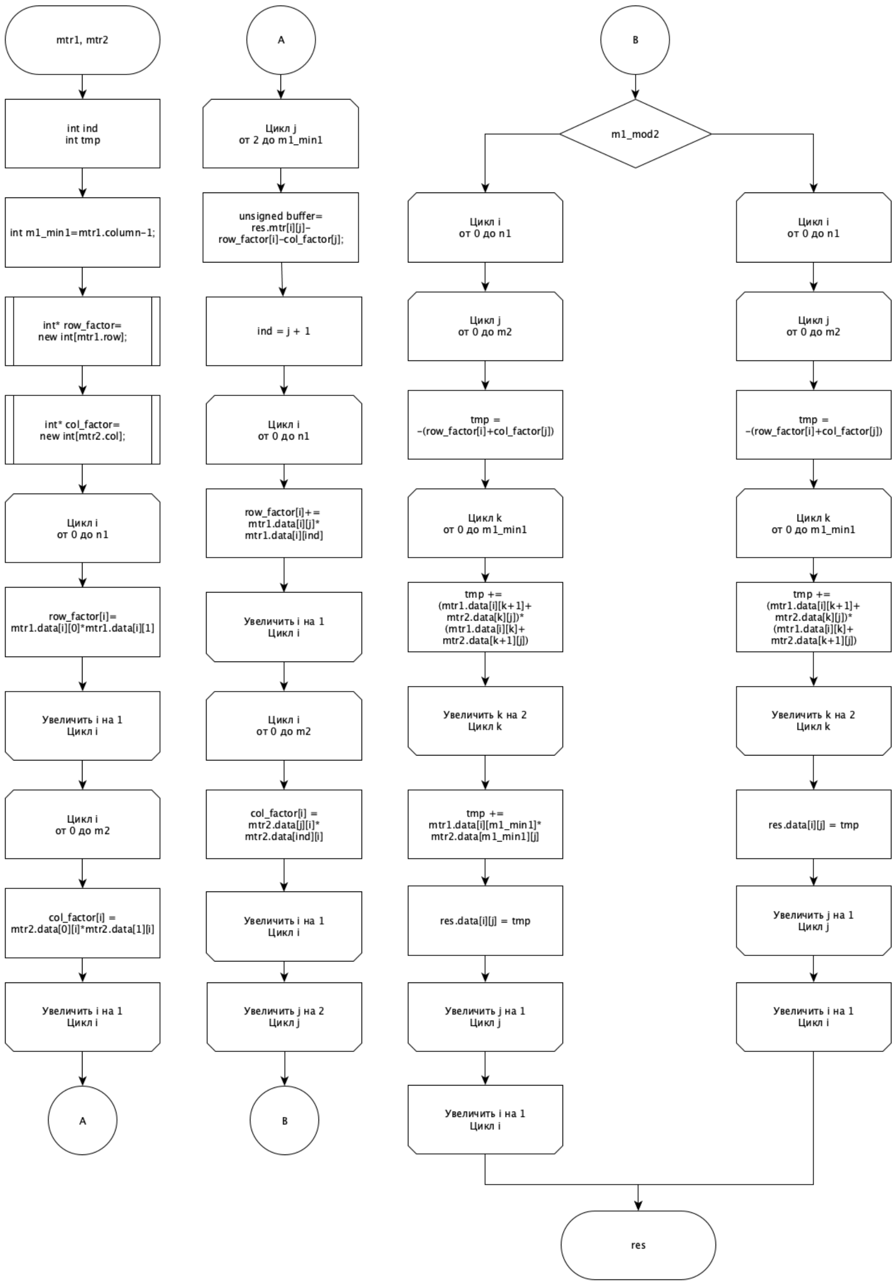

Рис. 3: Оптимизированный алгоритм Винограда

2.2 Вывод

В данной части были рассмотрены схемы алгоритмов.

3 Технологическая часть

В этом разделе будут изложены требования к программному обеспечению и листинги алгоритмов.

3.1 Требования к программному обеспечению

Входные данные: matr1 - первая матриц, matr2 - вторая матрица. Выходные данные: произведение двух матриц.

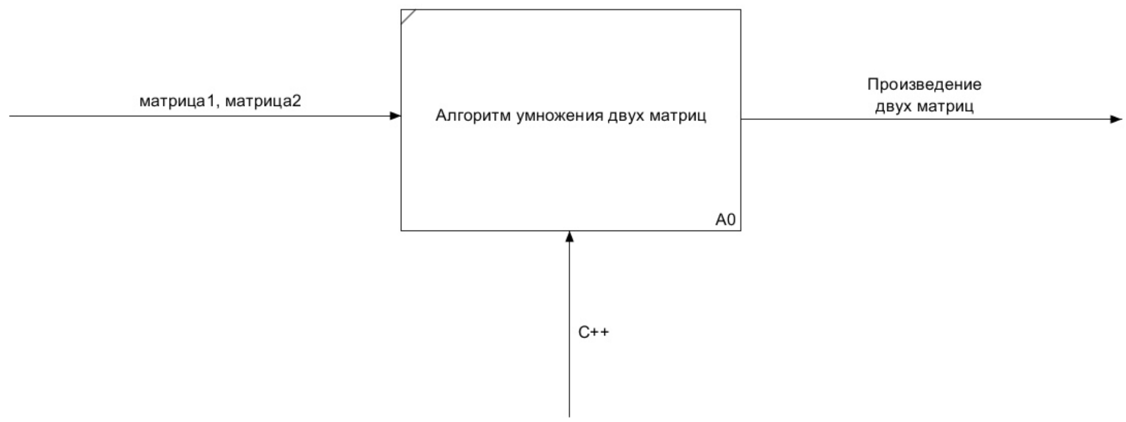

Рис. 4: IDEF0 диаграмма умножения двух матриц

3.2 Средства реализации

Данная программа разработана на языке C++, поддерживаемом многими операционными системами. Проект выполнен в среде Xcode.

В программе отсутствует проверка на ввод пустых или некорректных данных.

Для замера процессорного времени используется функция, возвращающая количество тиков.

Листинг 1: Функция замера количества тиков

```
unsigned long long tick()
{
    unsigned long long d;
    __asm__ __volatile__ ("rdtsc" : "=A" (d));
    return d;
}
```

3.3 Оценка трудоемкости

Трудоемкость стандартного алгоритма:
$F = 2 + N \cdot (2 + 2 + M \cdot (2 + 2 + K \cdot (2 + 8))) = 10MNK + 4NM + 4N + 2$

Трудоемкость алгоритма Винограда:
Лучший случай (m четное):
$F = 2 + 6N + 15N\frac{M}{2} + 2 + 6K + 15M\frac{K}{2} + 2 + 2N + 11NK + 13MNK + 2 = 13MNK + 7.5M(N + K) + 11NK + 8N + 6K + 8$
Худший случай (m нечетное) $(int)m/2 = (m - 1)/2$:
$F = 2+6N+15N\frac{M-1}{2}+2+6K+15(M-1)\frac{K}{2}+2+2N+11NK+26NK\frac{M-1}{2}+2+4N+15NK = 8N+12N+6K+(N+K)15\frac{M-1}{2}+26NK+13NMK-13NK = 13NMK + 7.M(N + K) + 13NK + 4.5N - 1.5K + 8$

Трудоемкость оптимизированного алгоритма Винограда:
Лучший случай (m четное) $mid = m/2$:
$F = mid(8 + 9N + 9K) + 3 + 4N + 12NK + 16NK \cdot mid = 3 + 4N + 12NK + mid(8+9N+9K+16NK) = 3+4N+12NK+0.5M(8+9N+9K+16NK) = 8MNK + 4.5MK + 12NK + 4M + 4N + 3$
Худший случай (m нечетное) $mid = (m - 1)/2$::
$F = mid(8 + 9N + 9K) + 3 + 4N + 18NK + 16NK \cdot mid = 3 + 4N + 18NK + mid(8+9N+9K+16NK) = 3+4N+18NK+\frac{M-1}{2}\cdot(8+9N+9K+16NK) = 3+4N+18NK+4M+4.5NM+4.5MK+8MNK-4-4.5N-4.5K-8NK = 8MNK + 4.5NM + 4.5MK + 10NK + 0.75N - 2.25K + 4M - 1$

3.4 Листинги функций

Ниже приведены листинги функций, реализующих алгоритмы умножения.

Листинг 2: Стандартный алгоритм

```
Matrix Matrix::classic_mult(const Matrix &mtr1, const Matrix &mtr2)
{
    Matrix res(mtr1.row, mtr2.column);

    for (int i = 0; i < mtr1.row; ++i)
    {
        for (int j = 0; j < mtr2.column; ++j)
        {
            data[i][j] = 0;
            for (int k = 0; k < mtr2.column; ++k)
            {
                res.data[i][j] = res.data[i][j] + mtr1.data[i][k] * mtr2.data[k][j];
            }
        }
    }

    return res;
}
```

Листинг 3: Алгоритм Винограда

```
Matrix Matrix::vinograd_mult(const Matrix &mtr1, const Matrix &mtr2)
{
    Matrix res(mtr1.row, mtr2.column);

    int row_factor[mtr1.row];
    int col_factor[mtr2.column];

    for(int i = 0; i < mtr1.row; i++)
    {
        row_factor[i] = 0;
        for(int j = 0; j < mtr1.column / 2; j++)
        {
            row_factor[i] = row_factor[i] + mtr1.data[i][2 * j] * mtr1.data[i][2 * j + 1];
        }
    }

    for(int i = 0; i < mtr2.column; i++)
    {
        col_factor[i] = 0;
        for(int j = 0; j < mtr2.row / 2; j++)
        {
            col_factor[i] = col_factor[i] + mtr2.data[2 * j][i] * mtr2.data[2 * j + 1][i];
        }
```

```
    }

    for(int i = 0; i < mtr1.row; i++)
    {
        for(int j = 0; j < mtr2.column; j++)
        {
            res.data[i][j] = -row_factor[i] - col_factor[j];
            for(int k = 0; k < mtr1.column / 2; k++)
            {
                res.data[i][j] = res.data[i][j] +
                                 (mtr1.data[i][2 * k + 1] + mtr2.data[2 * k][j]) *
                                 (mtr1.data[i][2 * k] + mtr2.data[2 * k + 1][j]);
            }
        }
    }

    if(mtr1.column % 2)
    {
        for(int i = 0; i < mtr1.row; i++)
            for(int j = 0; j < mtr2.column; j++)
                res.data[i][j] = res.data[i][j] + mtr1.data[i][mtr1.column - 1] * mtr2.data[mtr1.column - 1][j];
    }

    return res;
}
```

Листинг 4: Оптимизированный алгоритм Винограда

```
Matrix Matrix::vinograd_mult_optm(const Matrix &mtr1, const Matrix &mtr2)
{
    int ind;
    int tmp;
    Matrix res(mtr1.row, mtr2.column);

    int row_factor[mtr1.row];
    int col_factor[mtr2.column];

    int m1_min1 = mtr1.column - 1;

    for(int i = 0; i < mtr1.row; i++)
    {
        row_factor[i] = mtr1.data[i][0] * mtr1.data[i][1];
    }
    for(int i = 0; i < mtr2.column; i++)
    {
        col_factor[i] = mtr2.data[0][i] * mtr2.data[1][i];
    }

    for(int j = 2; j < m1_min1; j += 2)
    {
        ind = j + 1;
        for(int i = 0; i < mtr1.row; i++)
        {
            row_factor[i] += mtr1.data[i][j] * mtr1.data[i][ind];
        }
        for(int i = 0; i < mtr2.column; i++)
        {
            col_factor[i] += mtr2.data[j][i] * mtr2.data[ind][i];
        }
    }

    if(mtr1.column % 2)
    {
        for(int i = 0; i < mtr1.row; i++)
        {
            for(int j = 0; j < mtr2.column; j++)
            {
                tmp = -(row_factor[i] + col_factor[j]);
                for(int k = 0; k < m1_min1; k += 2)
                {
                    tmp += (mtr1.data[i][k+1] + mtr2.data[k][j]) * (mtr1.data[i][k] + mtr2.data[k+1][j]);
                }
                tmp += mtr1.data[i][m1_min1] * mtr2.data[m1_min1][j];
                res.data[i][j] = tmp;
            }
```

```
        }
    }

    else
    {
        for(int i = 0; i < mtr1.row; i++)
        {
            for(int j = 0; j < mtr2.column; j++)
            {
                tmp = -(row_factor[i] + col_factor[j]);
                for(int k = 0; k < m1_min1; k += 2)
                {
                    tmp += (mtr1.data[i][k+1] + mtr2.data[k][j
                        ]) * (mtr1.data[i][k] + mtr2.data[k
                        +1][j]);
                }
                res.data[i][j] = tmp;
            }
        }
    }

    return res;
}
```

3.5 Оптимизация алгоритма Винограда

Стандартный алгоритм Винограда может быть оптимизирован с помощью следующих модификаций:

1. конструкции вида $a = a + c$ заменяются на конструкции вида $a+ = c$;

Листинг 5: Составное присваивание

```
for (int i = 0; i <= first_matrix.rows; i++)
    {
        for (int j = 0; j <= (first_matrix.columns -
            m1_mod2); j += 2)
        {
            row_factor[i] += first_matrix.matrix[i][j] *
                first_matrix.matrix[i][j + 1];
        }
    }
```

2. заранее вычисляем переменные и выражения, которые часто используются;

Листинг 6: Вынос первой итерации из цикла

```
int m1_min1 = mtr1.column - 1;
```

3. внутри тройного цикла накапливать результат в буфер, а вне цикла сбрасывать буфер в ячейку памяти;

Листинг 7: Запись в буфер

```
if(mtr1.column % 2)
    {
        for(int i = 0; i < mtr1.row; i++)
        {
            for(int j = 0; j < mtr2.column; j++)
            {
                tmp = -(row_factor[i] + col_factor[j]);
                for(int k = 0; k < m1_min1; k += 2)
                {
                    tmp += (mtr1.data[i][k + 1] + mtr2.
                        data[k][j]) * (mtr1.data[i][k] +
                        mtr2.data[k + 1][j]);
                }
                tmp += mtr1.data[i][m1_min1] * mtr2.data[
                    m1_min1][j];
                res.data[i][j] = tmp;
            }
        }
    }
```

4. перенос вложенного цикла последнего цикла внутрь основного цикла для нечетных размерностей;

Листинг 8: Запись в буфер

```
else
    {
        for(int i = 0; i < mtr1.row; i++)
        {
            for(int j = 0; j < mtr2.column; j++)
            {
                tmp = -(row_factor[i] + col_factor[j]);
                for(int k = 0; k < m1_min1; k += 2)
                {
                    tmp += (mtr1.data[i][k + 1] + mtr2.
                        data[k][j]) * (mtr1.data[i][k] +
                        mtr2.data[k + 1][j]);
                }
                res.data[i][j] = tmp;
            }
        }
    }
```

3.6 Вывод

Так как оценка дается по самому быстрому растущему слагаемому, в нашем случае это куб линейного размера матриц. В классическом алгоритме коэффициент равен 10, у Винограда — 13, в улучшенном — 8. Можно сделать вывод о том, что улучшенный метод Винограда выигрывает по скорости работы у классического алгоритма умножения матриц.

4 Исследовательская часть

В данной части представлены результаты исследования быстродействия алгоритмов.

4.1 Постановка эксперимента

Были проведены исследования зависимости времени работы трех алгоритмов от размеров перемножаемых матриц. Замеры времени проводились отдельно для матриц четной размерности и матриц нечетной размерности. Для эксперимента использовались матрицы размера от 100 до 1000 с шагом 100 для четной размерности и размера от 101 до 1001 с шагом 100 для нечетной размерности. Временные замеры проводятся путём многократного проведения эксперимента и деления результирующего времени на количество итераций эксперимента.

4.2 Матрицы четного размера

Время работы алгоритмов на матрицах четной размерности:

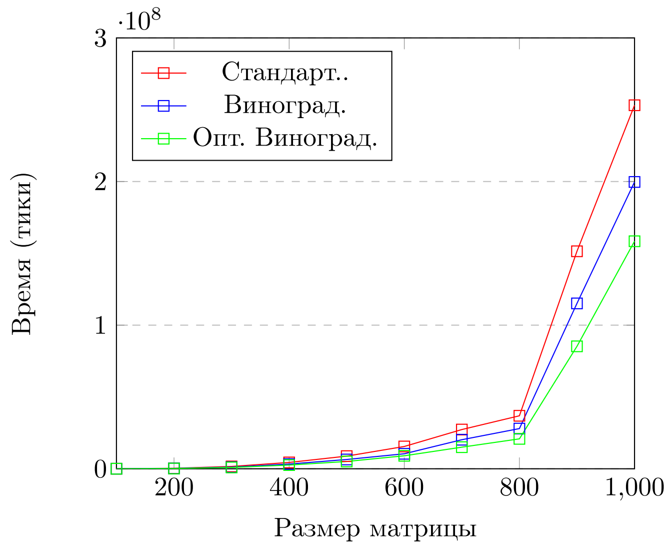

Рис. 5: График времени работы алгоритмов на матрицах четной размерности

4.3 Матрицы нечетного размера

Время работы алгоритмов на матрицах нечетной размерности:

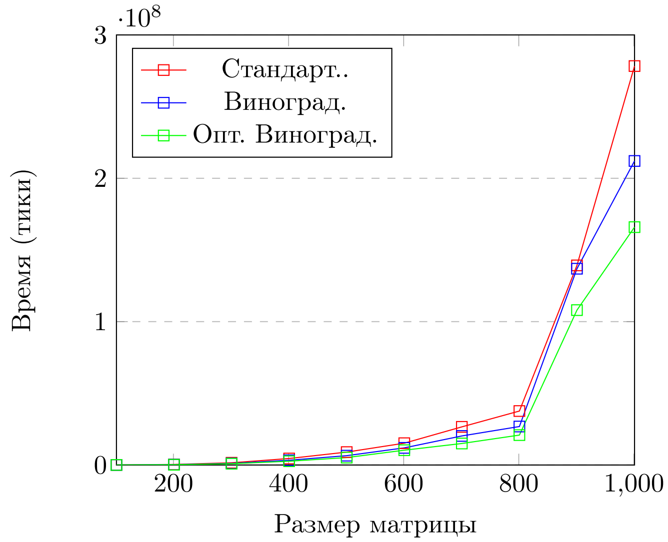

Рис. 6: График времени работы алгоритмов на матрицах нечетной размерности

4.4 Вывод

Алгоритм Виноградова начинает выигрывать в быстродействии у других алгоритмов только в случае, когда матрицы имеют размерность, не позволяющую хранить их целиком в памяти компьютера. Стандартный алгоритм умножения матриц работает дольше, чем среализация алгоритма Винограда, в то время как оптимизированная версия алгоритма Винограда работает быстрее их обоих.

Заключение

В данной работе были исследованы алгоритмы умножения матриц. Для каждого алгоритма была вычислена трудоемкость данного алгоритма в выбранной модели вычислений. Для реализации каждого алгоритма был произведен сравнительный анализ алгоритмов.

В ходе работы были получены следующие навыки:

1. изучение и подсчет трудоемкости алгоритмов;
2. алгоритм Винограда был оптимизирован с целью снижения трудоемкости;
3. оценки трудоемкости были экспериментально подтверждены.

Список литературы

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

[2] Henry Cohn, Robert Kleinberg, Balazs Szegedy, and Chris Umans. Group-theoretic Algorithms for Matrix Multiplication. arXiv:math.GR/0511460. Proceedings of the 46th Annual Symposium on Foundations of Computer Science, 23-25 October 2005, Pittsburgh, PA, IEEE Computer Society, pp. 379—388..

[3] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation, 9:251-280, 1990.