Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт
по лабораторной работе № 2

Название: Алгоритмы умножения матриц

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б _____ М. Р. Саркисян
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Оглавление

Введение

В ходе данной лабораторной работы необходимо изучить и применить алгоритмы умножения матриц (стандартный алгоритм умножения матриц, алгоритм Винограда и улучшенный алгоритм Винограда), рассчитать их трудоемкость, а также получить навыки в улучшении алгоритмов.

1 Аналитическая часть

В данном разделе будут формализованы цель и задачи, а также рассмотрено описание алгоритмов умножения матриц.

1.1 Цель и задачи

Целью данной лабораторной работы является изучение алгоритмов умножения матриц: стандартного алгоритма умножения матриц и алгоритма умножения матриц Винограда.

Задачи:

1. Изучить алгоритмы умножения матриц (стандартный алгоритм умножения матриц и алгоритм Винограда).
2. Улучшить алгоритм Винограда.
3. Дать теоретическую оценку трудоемкости стандартного алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда.
4. Реализовать три алгоритма умножения матриц на одном из языков программирования.
5. Провести сравнительный анализ реализаций алгоритмов.

1.2 Описание алгоритмов

Матрицей A размера $[N \times M]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, которая представляет собой совокупность N строк и M столбцов, на пересечении которых находятся элементы[1].

Операция умножения, одна из основных операций над матрицами, выполнима только в том случае, если число столбцов в первой матрице совпадает с числом строк во второй. В частности, умножение всегда выполнимо, если оба сомножителя - квадратные матрицы одного и того же порядка.

Алгоритмы умножения матриц применяются во многих областях, сколь бы то ни было связанных с линейной алгеброй, таких как:

- физика;
- экономика;
- компьютерная графика.

1.2.1 Классический алгоритм умножения матриц

Пусть даны две прямоугольные матрицы A и B размерности m $\times$ n и n $\times$ l соответственно:

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

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

Тогда матрица C размерностью m $\times$ l:

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

в которой:

$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j}$

называется произведением матриц A и B.

Пример:

$$A = \begin{bmatrix} 3 & 1 & 4 \\ 0 & -2 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 3 \\ 1 & 1 \\ -3 & 2 \end{bmatrix}$$

$$AB = \begin{bmatrix} -11 & 18 \\ -5 & 0 \\ 1 & 4 \end{bmatrix}$$

1.2.2 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[3].

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$. Их скалярное произведение можно найти по формуле 1.1:

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.1}$$

Это равенство можно переписать в виде 1.2:

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{1.2}$$

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.

1.3 Вывод

Итак, в этом разделе были поставлены цель и задачи, рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, указаны их формулы и кратко описаны особенности применения.

2 Конструкторская часть

В данном разделе будут рассмотрены требования к ПО и схемы алгоритмов.

2.1 Требования к программному обеспечению

Входные данные: две матрицы и их размерности. Выходные данные: результат умножения введенных матриц.

На рисунке 2.1 представлена IDEF0-диаграмма, описывающая алгоритм нахождения произведения двух матриц.

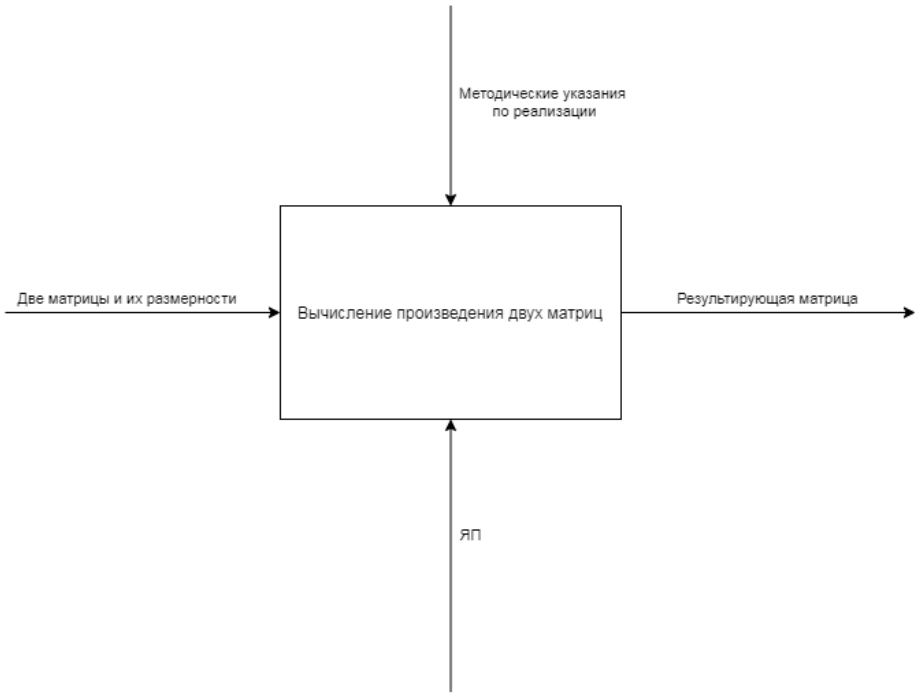


Рис. 2.1: IDEF0-диаграмма алгоритма нахождения произведения двух матриц

Тестирование будет производиться по стратегии чёрного ящика, что означает отсутствие знания о внутреннем коде тестируемого объекта.

2.2 Схемы алгоритмов

На рисунках 2.2, 2.3 и 2.4 представлены схемы алгоритмов умножения матриц.

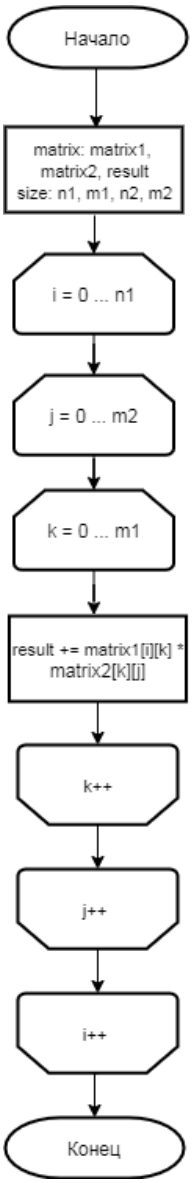


Рис. 2.2: Схема классического алгоритма умножения матриц

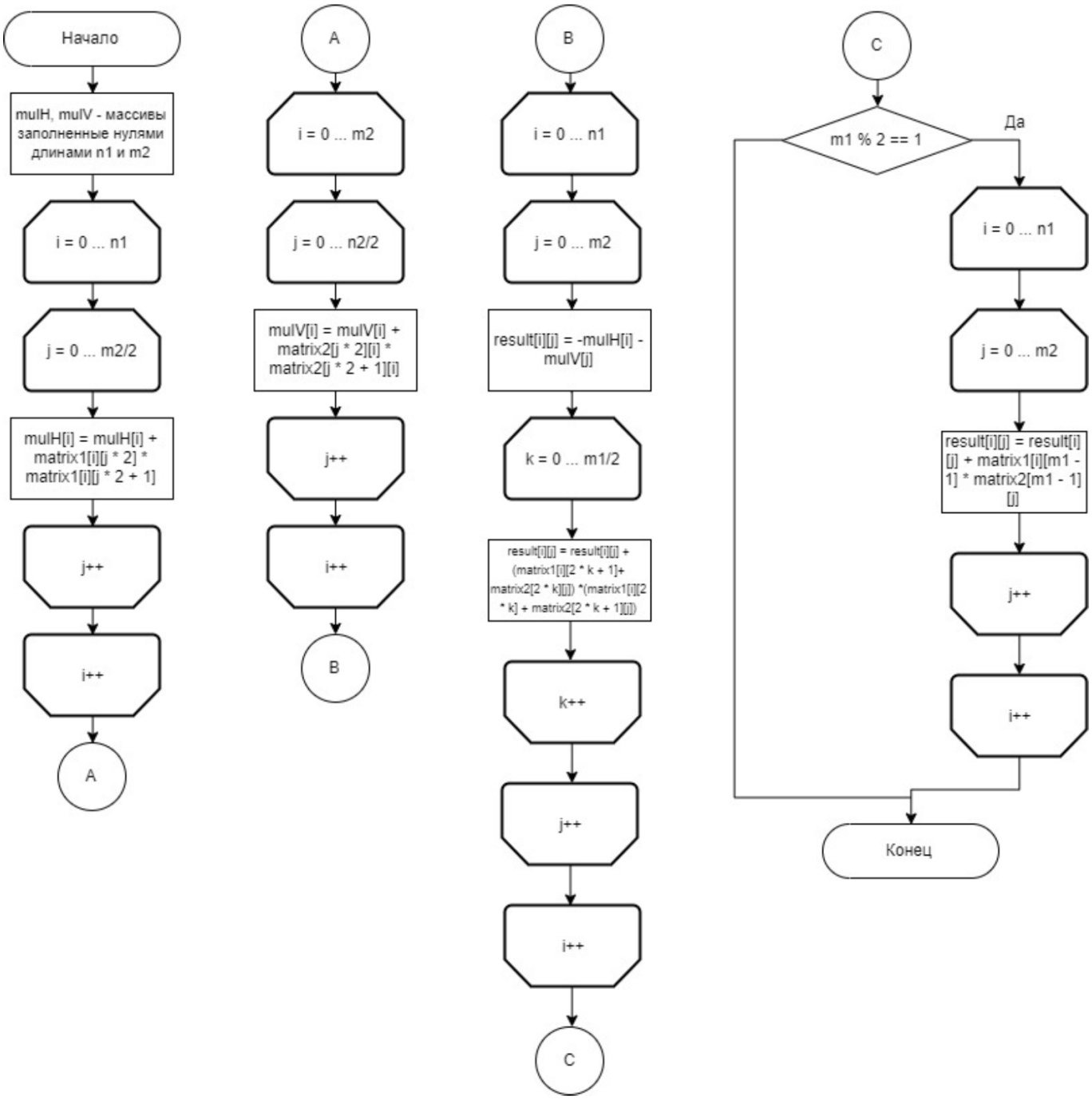


Рис. 2.3: Схема алгоритма Винограда

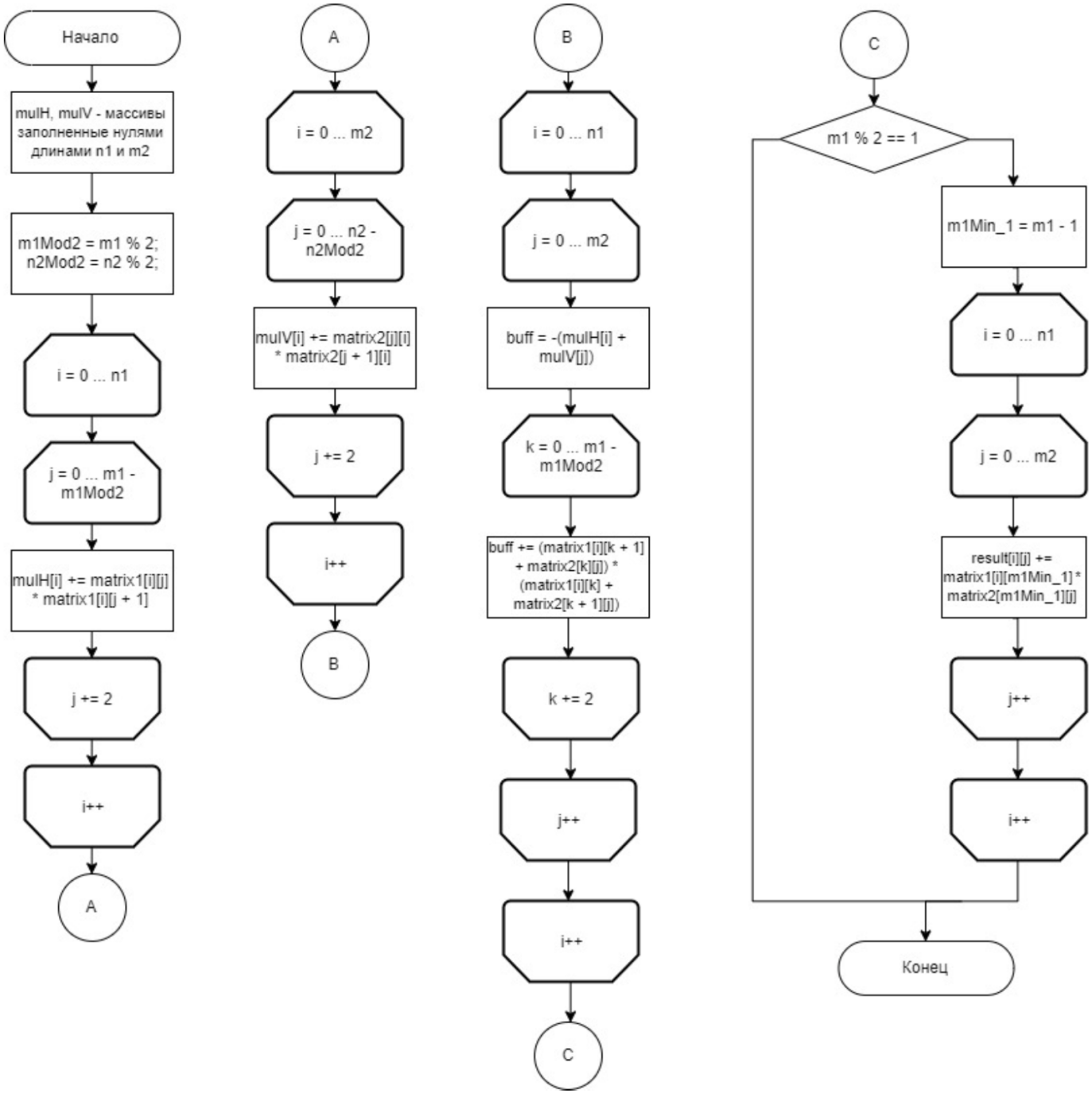


Рис. 2.4: Схема оптимизированного алгоритма Винограда

2.3 Трудоемкость алгоритмов

Модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [];
- оценка трудоемкости цикла for от 0 до N с шагом 1 $F_{for} = 2 + N \cdot (2 + F_{body})$, где F_{body} - тело цикла;
- стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

Оценим трудоемкость алгоритмов по коду программы.

2.3.1 Классический алгоритм

Трудоемкость классического алгоритма:

$$10MNQ + 4MQ + 4M + 2$$

2.3.2 Алгоритм Винограда

Трудоемкость алгоритма Винограда:

Первый цикл: $15/2 \cdot MN + 5 \cdot M + 2$

Второй цикл: $15/2 \cdot MN + 5 \cdot M + 2$

Третий цикл: $13 \cdot MNQ + 12 \cdot MQ + 4 \cdot M + 2$

Условный переход: $\begin{bmatrix} 2 & \text{, в случае невыполнения условия} \\ 15 \cdot QM + 4 \cdot M + 2 & \text{, в случае выполнения условия} \end{bmatrix}$

Итого: $15/2 \cdot MN + 5 \cdot M + 2 + 15/2 \cdot MN + 5 \cdot M + 2 + 13 \cdot MNQ + 12 \cdot MQ + 4 \cdot M + 2 + \begin{bmatrix} 2 & \text{, в случае невыполнения условия} \\ 15 \cdot QM + 4 \cdot M + 2 & \text{, в случае выполнения условия} \end{bmatrix}$

2.3.3 Оптимизированный алгоритм Винограда

Трудоемкость алгоритма Винограда:

Первый цикл: $11/2 \cdot MN + 4 \cdot M + 2$

Второй цикл: $11/2 \cdot MN + 4 \cdot M + 2$

Третий цикл: $17/2 \cdot MNQ + 9 \cdot MQ + 4 \cdot M + 2$

Условный переход: $\begin{bmatrix} 1 & \text{, в случае невыполнения условия} \\ 10 \cdot QM + 4 \cdot M + 2 & \text{, в случае выполнения условия} \end{bmatrix}$

Итого: $11/2 \cdot MN + 4 \cdot M + 2 + 11/2 \cdot MN + 4 \cdot M + 2 + 17/2 \cdot MNQ + 9 \cdot MQ + 4 \cdot M + 2 + \begin{bmatrix} 1 & \text{, в случае невыполнения условия} \\ 10 \cdot QM + 4 \cdot M + 2 & \text{, в случае выполнения условия} \end{bmatrix}$

2.4 Вывод

Так как оценка трудоемкости дается по самому быстро растущему слагаемому, мы рассматриваем куб линейного размера матриц. В классическом варианте алгоритма данный коэффициент равен 10, в алгоритме Винограда - 13, в оптимизированном (улучшенном) алгоритме Винограда - 8.5. Отсюда можно сделать вывод, что улучшенный алгоритм Винограда выигрывает по скорости работы у неулучшенного и классического аналогов.

3 | Технологическая часть

В данном разделе будут рассмотрены средства реализации и представлен листинг кода, а также проведено тестирование по стратегии чёрного ящика.

3.1 Средства реализации

Для реализации программы мной был выбран C++[2], так как это язык программирования, содержащий средства создания эффективных программ практически любого назначения. Он также был достаточно подробно изучен мной на предыдущем курсе.

3.2 Листинг кода

В листингах 3.1, 3.2, 3.3 представлена реализация алгоритмов умножения матриц.

Листинг 3.1: Функция классического умножения матриц

```
1 int standardMulti(int** A, int** B, int** res, int N_a, int M_a, int
    M_b)
2 {
3   for (int i = 0; i < N_a; i++)
4     for (int j = 0; j < M_b; j++)
5     {
6       res[i][j] = 0;
7       for (int k = 0; k < M_a; k++)
8         res[i][j] += A[i][k] * B[k][j];
```

```
    }

  return OK;
}
```

Листинг 3.2: Функция алгоритма Винограда

```
int vinogradMulti(int** A, int** B, int** result, int N_a, int M_a, int
    M_b)
{
  int* mulH = new int[N_a];
  int* mulV = new int[M_b];
  if (!mulH || !mulV)
  {
    return FAIL;
  }
  else
  {
    for (int i = 0; i < N_a; i++)
    {
      mulH[i] = 0;
      for (int k = 0; k < M_a / 2; k++)
        mulH[i] = mulH[i] + A[i][2 * k] * A[i][2 * k + 1];
    }

    for (int j = 0; j < M_b; j++)
    {
      mulV[j] = 0;
      for (int k = 0; k < M_a / 2; k++)
        mulV[j] = mulV[j] + B[2 * k][j] * B[2 * k + 1][j];
    }

    for (int i = 0; i < N_a; i++)
      for (int j = 0; j < M_b; j++)
      {
        result[i][j] = 0 - mulH[i] - mulV[j];
        for (int k = 0; k < M_a / 2; k++)
          result[i][j] = result[i][j] + \
          (A[i][2 * k] + B[2 * k + 1][j]) * \
```

```
32            (A[i][2 * k + 1] + B[2 * k][j]);
33        }
34
35      if (M_a % 2 == 1)
36        for (int i = 0; i < N_a; i++)
37          for (int j = 0; j < M_b; j++)
38            result[i][j] = result[i][j] + \
39            A[i][M_a - 1] * B[M_a - 1][j];
40    }
41
42    return OK;
43  }
```

Листинг 3.3: Функция улучшенного алгоритма Винограда

```
1  int vinogradOptiMulti(int** A, int** B, int** result, int N_a, int M_a,
       int M_b)
2  {
3    int* mulH = new int[N_a];
4    int* mulV = new int[M_b];
5    if (!(mulH && mulV))
6    {
7      return FAIL;
8    }
9    else
10   {
11     int buf;
12     for (int i = 0; i < N_a; i++)
13     {
14       buf = 0;
15       for (int k = 1; k < M_a; k += 2)
16         buf -= A[i][k - 1] * A[i][k];
17       mulH[i] = buf;
18     }
19
20     for (int j = 0; j < M_b; j++)
21     {
22       buf = 0;
23       for (int k = 1; k < M_a; k += 2)
```

```
24          buf -= B[k - 1][j] * B[k][j];
25        mulV[j] = buf;
26      }
27
28      for (int i = 0; i < N_a; i++)
29        for (int j = 0; j < M_b; j++)
30        {
31          buf = mulH[i] + mulV[j];
32          for (int k = 1; k < M_a; k += 2)
33            buf += (A[i][k - 1] + B[k][j]) * (A[i][k] + B[k - 1][j]);
34
35          if (M_a % 2)
36            buf += A[i][M_a - 1] * B[M_a - 1][j];
37
38          result[i][j] = buf;
39
40        }
41    }
42
43    return OK;
44  }
```

Что было изменено в улучшенной версии алгоритма Винограда:

- избавление от деления в цикле;
- замена сложения с повторяющимся операндом на +=;
- накопление результата в буфер, вне цикла - сброс буфера в ячейку матрицы.

3.3 Тестирование

Тестирование производилось по стратегии чёрного ящика, что означает отсутствие знания о внутреннем коде тестируемого объекта.

Результаты тестирования приведены в таблице 3.1.

Таблица 3.1: Результаты тестирования

№	n1, m1	n2, m2	matrix1	matrix2	expected result	got result
1	0 0	0 0				
2	1 1	0 0	4			
3	0 0	1 1		1		
4	3 2	1 1	1 3 2 3 2 3	1		
5	1 1	1 2	5	2 3	10 15	10 15
6	2 2	2 2	1 2 0 1	-1 2 1 3	1 8 1 3	1 8 1 3

3.4 Вывод

В данном разделе рассматривались средства реализации и представлены листинги кода вышеописанных алгоритмов с последующим их тестированием.

4 Исследовательская часть

В данном разделе будет представлено сравнение реализаций алгоритмов по времени.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

В ходе выполнения лабораторной работы было произведено две серии замеров:

- для матриц размером $N \times N$, $N \in \{100, 200, 300, 400, 500\}$;
- для матриц размером $N \times N$, $N \in \{101, 201, 301, 401, 501\}$.

Для матриц каждой размерности замер времени был произведен на 50 итерациях, а в качестве результата взято усредненное время работы алгоритма. Результаты замеров времени приведены в таблицах 4.1 и 4.2, графически представлены на рисунках 4.1 и 4.2.

Таблица 4.1: Результаты первой серии замеров процессорного времени в мсек. для реализаций алгоритмов умножения матриц

N	Классический алг.	Алг. Винограда	Опт. алг. Винограда
100	3.74	2.68	2.08
200	34.08	24.18	21.15
300	119.62	86.34	68.12
400	353.3	218.53	169.32
500	597.42	508.37	342.76

Таблица 4.2: Результаты второй серии замеров процессорного времени в мсек. для реализаций алгоритмов умножения матриц

N	Классический алг.	Алг. Винограда	Опт. алг. Винограда
101	3.76	2.72	2.03
201	34.22	25.01	19.56
301	120.76	87.18	85.02
401	315.42	213.26	171.42
501	599.99	453.66	339.84

Рис. 4.1: Результаты первой серии замеров процессорного времени

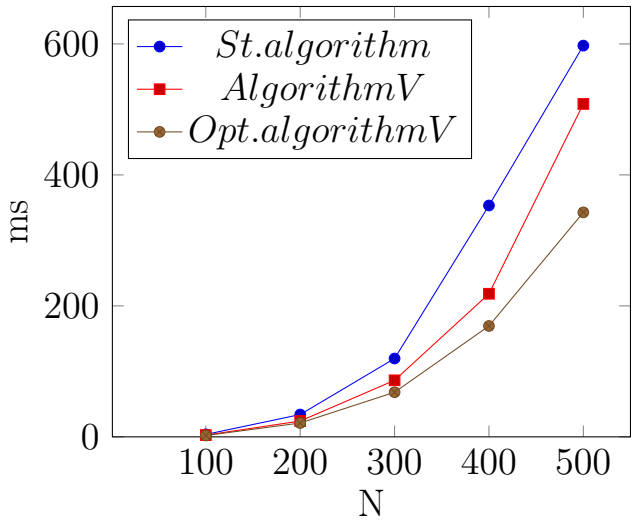
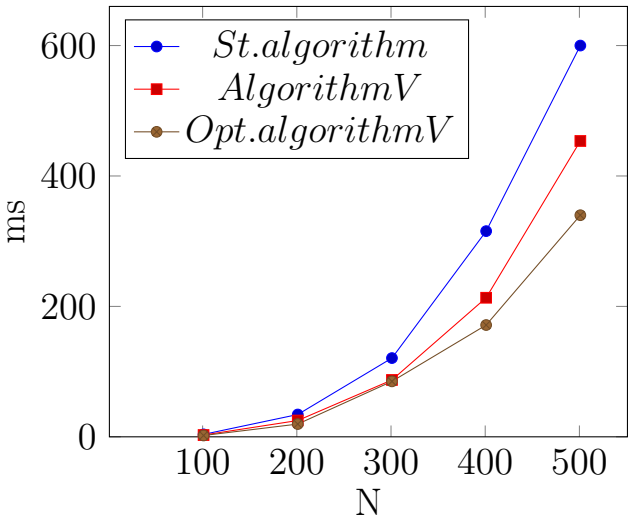


Рис. 4.2: Результаты второй серии замеров процессорного времени



4.2 Вывод

Несмотря на одинаковый характер трудоемкости алгоритмов, результаты проведенного эксперимента отвечают ожиданиям: оптимизированный алгоритм Винограда имеет наибольшую скорость работы, алгоритм Винограда без оптимизаций работает медленее оптимизированного, но быстрее стандартного алгоритма.

Эксперимент показал, что оптимизированный алгоритм Винограда работает в среднем в 2 раза быстрее, чем стандартный алгоритм умножения на тех же входных данных, и примерно в 1.5 быстрее, чем алгоритм Винограда без оптимизаций.

Заключение

В ходе лабораторной работы мной были изучены алгоритмы умножения матриц: стандартный, алгоритм Винограда и улучшенный алгоритм Винограда. Была произведена реализация вышеуказанных алгоритмов.

После этого было сделано сравнение полученных алгоритмов, вследствие которого было установлено, что при малых размерах матрицы алгоритм Винограда и его оптимизированная версия сравнимы со стандартным алгоритмом, но являются ощутимо лучше при больших размерах матрицы. При этом улучшенный алгоритм Винограда является в 1,5-2 раза более оптимальным по трудоемкости и времени по сравнению со стандартным алгоритмом умножения матриц.

Нельзя не заметить, что подобное достигается путём использования дополнительной памяти. Если ресурсы конкретной задачи не ограничены, можно считать улучшенную версию алгоритма Винограда наилучшей для вычисления плотных матриц любого размера.

Список использованных источников

1. И. В. Белоусов. Матрицы и определители, учебное пособие по линейной алгебре, 2006.

2. Руководство по языку C++ [Электронный ресурс], - режим доступа: https://docs.micr ru/cpp/cpp/cpp-language-reference?view=vs-2019

3. Дж. Макконнел. Основы современных алгоритмов. 2-е дополненное издание. – М.: Техносфера, 2006. – 267с.