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»

Отчет по лабораторной работе №2 по курсу "Анализ алгоритмов"

Тема Умножение матриц. Формула Винограда

Студент Котляров Н.А.

Группа ИУ7-51Б

Оценка (баллы)

Преподаватель Волкова Л.Л.

Москва — 2022 г.

Оглавление

Введение **3**

1 Аналитический раздел **4**

1.1 Стандартный алгоритм 4

1.2 Алгоритм Копперсмита - Винограда 4

2 Конструкторская часть **6**

2.1 Разработка алгоритмов 6

2.2 Модель вычислений (оценки трудоемкости) 9

2.3 Трудоёмкость алгоритмов 9

2.3.1 Трудоемкость стандартного алгоритма 9

2.3.2 Трудоемкость алгоритма Винограда 10

2.3.3 Трудоемкость оптимизированного алгоритма Винограда 11

3 Технологическая часть **12**

3.1 Требования к ПО 12

3.2 Средства реализации 12

3.3 Сведения о модулях программы 12

3.4 Реализация алгоритмов 13

3.5 Функциональные тесты 15

4 Исследовательская часть **17**

4.1 Технические характеристики 17

4.2 Демонстрация работы программы 17

4.3 Время выполнения реализаций алгоритмов 18

Заключение **23**

Список использованных источников **24**

Введение

В настоящее время компьютеры оперируют множеством типов данных. Одним из них являются матрицы. Для некоторых алгоритмов необходимо выполнять их перемножение. Данная операция является затратной по времени, имея сложность порядка $O(N^3)$. Поэтому Ш. Виноград создал свой алгоритм умножения матриц, который являлся асимптотически самым быстрым из всех.

Целью данной лабораторной работы является анализ алгоритмы перемножения матриц Винограда. Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить выбранные алгоритмы умножения матриц и способы их оптимизации;
- разработать оптимизированный алгоритм Винограда;
- составить схемы рассмотренных алгоритмов;
- реализовать алгоритмы умножения матриц;
- провести сравнительный анализ реализаций алгоритмов по затрачиваемым ресурсам (время и память);
- описать и обосновать полученные результаты.

1 Аналитический раздел

В данном разделе будут представлены описания алгоритмов умножения матриц стандартным способом, методом Винограда и его оптимизации.

1.1 Стандартный алгоритм

Даны две прямоугольные матрицы:

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}. \tag{1.1}$$

Индексы у матриц обозначают их размерность — матрица A размером $l \cdot m$, матрица B — $m \cdot n$ соответственно. Перемножать матрицы можно только когда вторая размерность A и первая размерность B совпадают. Тогда результатом умножения данных матриц будет являться матрица C размером $l \cdot n$:

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

где каждый элемент матрицы вычисляется следующим образом:

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1, l}; j = \overline{1, n}) \tag{1.3}$$

1.2 Алгоритм Копперсмита - Винограда

Асимптотика данного алгоритма является лучшей среди существующих, она составляет $O(N^{2,3755})$.

Пусть даны два вектора: $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Вектор V играет роль строки первой матрицы-множителя, вектор W — столбца второй матрицы-множителя. Их скалярное произведение равно 1.3

Это выражение эквивалентно следующему:

$$V \cdot W = S1 - S2, \text{ где} \tag{1.4}$$

$$S1 = (u_1 + w_2) \cdot (u_2 + w_1) + (u_3 + w_4) \cdot (u_4 + w_3)$$

$$S2 = u_1 \cdot u_2 + u_3 \cdot u_4 + w_1 \cdot w_2 + w_3 \cdot w_4$$

В случае, если размерность умножаемых векторов нечетна (например, равна 5), к произведению векторов добавляется слагаемое $v_5 \cdot w_5$.

Преимуществом данного метода вычисления является то, что слагаемые $v_1v_2, v_3v_4, w_1w_2, w_3w_4$ можно рассчитать для каждой строки и каждого столбца заранее, тем самым сократить количество операций умножения и привести к менее затратной операции сложения.

Также как варианты оптимизации можно использовать побитовые сдвиги, += , использовать буферы.

Вывод

В данном разделе были рассмотрены принципы работы алгоритмов умножения матриц. Полученных знаний достаточно для разработки выбранных алгоритмов. На вход алгоритмам будут подаваться две матрицы элементов и их размерности. Реализуемое ПО будет работать в пользовательском режиме (вывод произведения матриц, рассчитанного тремя методами), а также в экспериментальном (проведение замеров времени выполнения реализаций алгоритмов).

2 Конструкторская часть

В этом разделе будут приведены схемы алгоритмов и вычисления трудоемкости данных алгоритмов.

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены схемы алгоритмов стандартного умножения матриц, умножения Винограда и оптимизированного умножения Винограда.

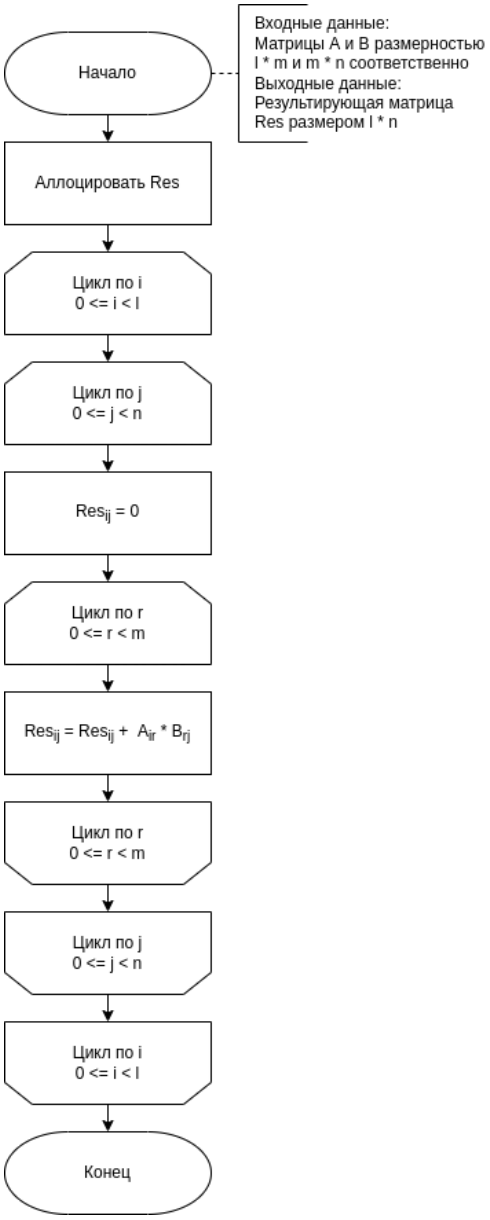

Рисунок 2.1 – Схема алгоритма стандартного умножения матриц

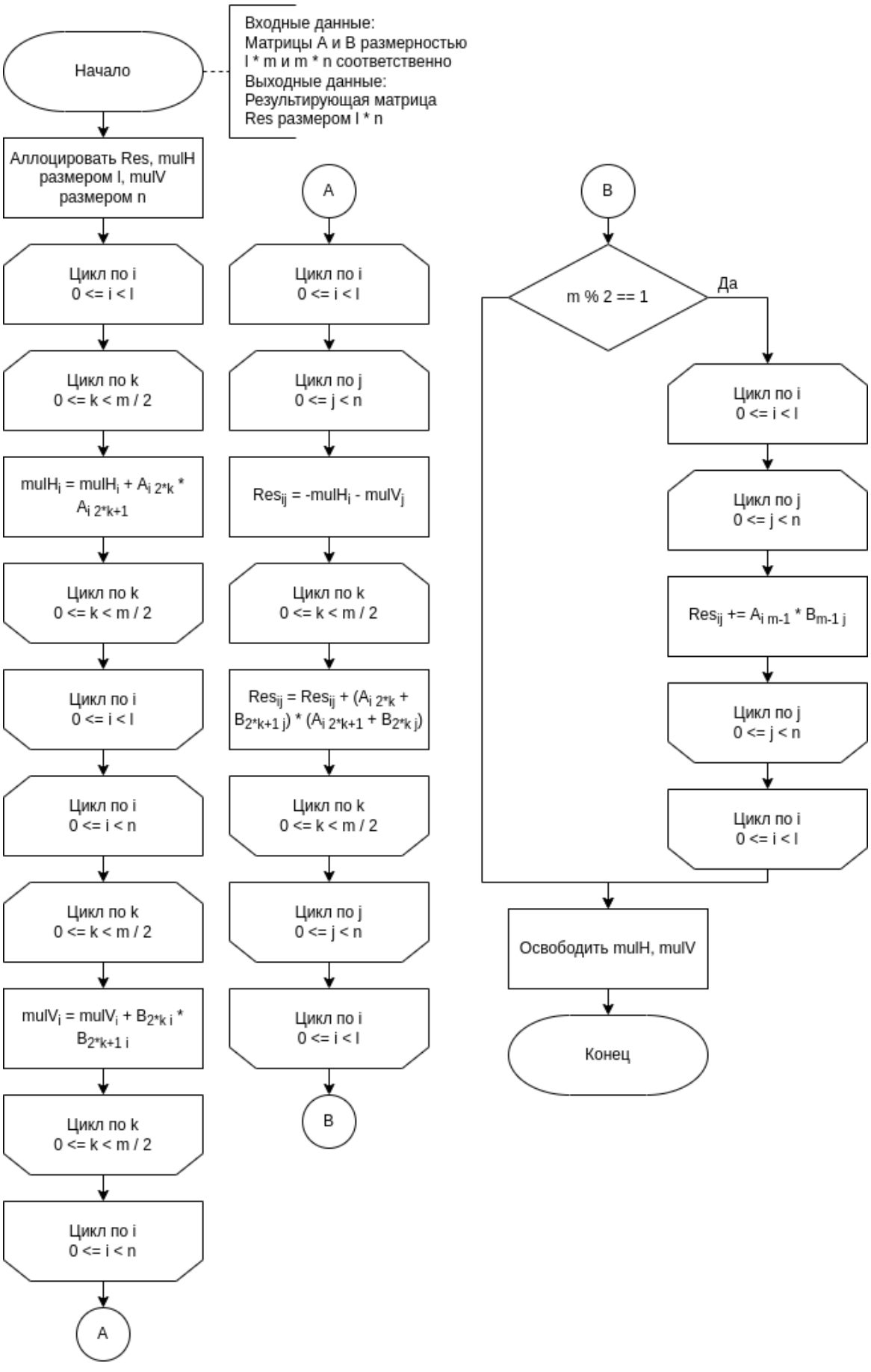

Рисунок 2.2 – Схема алгоритма умножения матриц Винограда

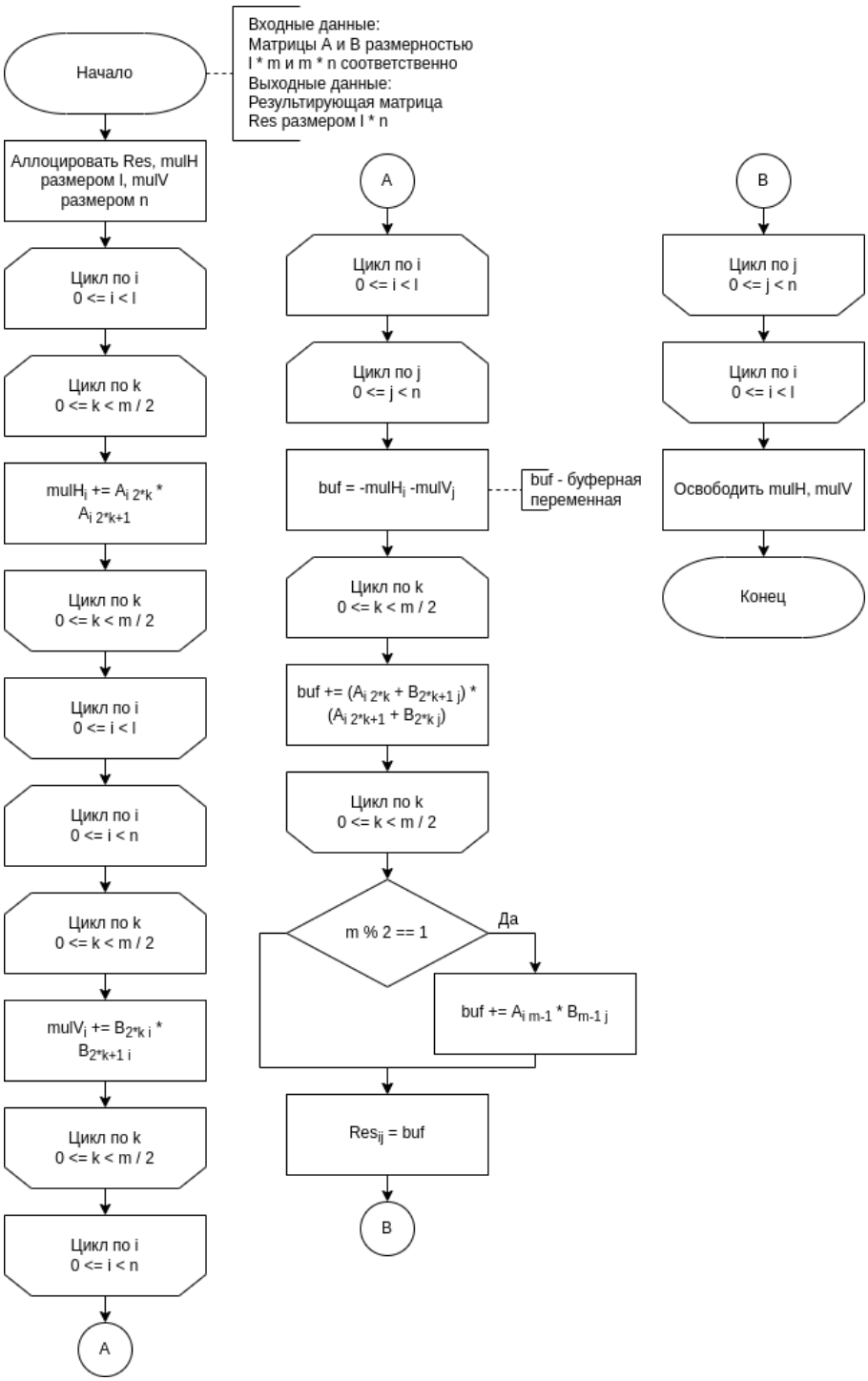

Рисунок 2.3 – Схема оптимизированного алгоритма умножения матриц Винограда

2.2 Модель вычислений (оценки трудоемкости)

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1. Операции из списка (2.1) имеют трудоемкость 1:

$$+, ++, + =, -, --, - =, ! =, <, >, <=, >=, <<, >>, [] \tag{2.1}$$

2. Операции из списка (2.2) имеют трудоемкость 2:

$$/, / =, *, * =, \%, \% = \tag{2.2}$$

3. Трудоемкость оператора выбора `if условие then A else B` рассчитывается как

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4. Трудоемкость цикла с N итерациями рассчитывается, как (2.4);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

5. Трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

2.3.1 Трудоемкость стандартного алгоритма

Трудоемкость стандартного алгоритма состоит из следующих составляющих:

- трудоемкость цикла по i $\in$ [1, L] $f = 2 + L(2 + f_{body})$;
- трудоемкость цикла по j $\in$ [1, N] $f = 2 + 3 + N(2 + f_{body})$;
- трудоемкость цикла по r $\in$ [1, M] $f = 2 + M(2 + 12) = 2 + 14M$.

Итого, трудоемкость стандартного алгоритма равна
$f = 2+L(4+N(7+14M)) = 2+4L+7LN+14LNM \approx 14LNM \approx O(N^3)$

2.3.2 Трудоемкость алгоритма Винограда

Трудоемкость алгоритма Винограда состоит из следующих составляющих:

- аллокация и инициализация векторов mulH и mulV — $f_{HV} = L+N+2+L(2+3+\frac{1}{2}M(3+14))+2+N(2+3+\frac{1}{2}M(3+14)) = 2+6L+6N+\frac{17}{2}LM+\frac{17}{2}NM$;
- цикл по i $\in$ [1, L] $f_L = 2 + L(2 + f_{body})$;
- цикл по j $\in$ [1, N] $f_N = 2 + N(2 + 7 + f_{body})$;
- цикл по k $\in$ [1, M/2] $f_{M/2} = 3 + \frac{1}{2}M(3 + 28) = 3 + \frac{31}{2}M$;
- проверка размеров на нечетность

$$f_{if} = 3 + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.5}$$

Итого, трудоемкость алгоритма Винограда для лучшего случая, который наступает при четной размерности M умножаемых матриц, равна
$f = 4 + 10L + 6N + \frac{17}{2}LM + \frac{17}{2}NM + 12LN + \frac{31}{2}LNM \approx 15,5LNM \approx O(N^3)$

Для худшего случая, который наступает при нечетной размерности M умножаемых матриц, равна
$f = 6 + 14L + 6N + \frac{17}{2}LM + \frac{17}{2}NM + 25LN + \frac{31}{2}LNM \approx 15,5LNM \approx O(N^3)$

2.3.3 Трудоемкость оптимизированного алгоритма Винограда

Трудоемкость оптимизированного алгоритма Винограда состоит из следующих составляющих:

- аллокация и инициализация векторов mulH и mulV — $f_{HV} = L + N + 2 + L(2 + 3 + \frac{1}{2}M(3 + 13)) + 2 + N(2 + 3 + \frac{1}{2}M(3 + 13)) = 2 + 6L + 6N + 8LM + 8NM$;
- цикл по i $\in$ [1, L] $f_L = 2 + L(2 + f_{body})$;
- цикл по j $\in$ [1, N] $f_N = 2 + N(2 + 5 + f_{body} + f_{if} + 3)$;
- цикл по k $\in$ [1, M/2] $f_{M/2} = 3 + \frac{1}{2}M(3 + 23) = 3 + 13M$;
- проверка размеров на нечетность

$$f_{if} = 3 + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.6}$$

Вывод

Были разработаны схемы всех трех алгоритмов умножения матриц. Для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- ПО принимает две целочисленные матрицы размерностью не больше 700;
- ПО возвращает результирующую матрицу — произведение двух входных матриц.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования Си [1].

В данном языке есть все требующиеся для данной лабораторной инструменты разработки.

Процессорное время работы реализаций алгоритмов было замерено с помощью функции tick() из библиотеки sys/time.h [2].

3.3 Сведения о модулях программы

Программа состоит из четырех модулей:

1) main.c — главный файл программы, в котором вызываются остальные файлы;
2) interface.c — файл программы, в котором располагается меню;
3) algos.c — файл программы, в котором располагаются коды всех алгоритмов;

4) data.c — файл работой с памятью.

3.4 Реализация алгоритмов

В листингах 3.1, 3.2, 3.3 представлены реализации алгоритмов умножения матриц: стандартный, Винограда, оптимизированный Винограда.

Листинг 3.1 – Реализация стандартного алгоритма умножения матриц

```
1  int **multStand(int ** m1, int ** m2, int l, int m, int n)
2  {
3      int **res = allocateMatrix(l, n);
4      if (res) {
5          for (int i = 0; i < l; i++) {
6
7              for (int j = 0; j < n; j++) {
8                  res[i][j] = 0;
9                  for (int r = 0; r < m; r++)
10                 res[i][j] = res[i][j] + m1[i][r] * m2[r][j];
11             }
12         }
13     }
14     else
15     printf("Error␣!\n");
16     return res;
17 }
```

Листинг 3.2 – Реализация алгоритма умножения матриц Винограда

```
1  int **multVin(int **m1, int **m2, int l, int m, int n)
2  {
3      int **res = allocateMatrix(l, n);
4      int *mulH = calloc (l, sizeof(int));
5      int *mulV = calloc (n, sizeof(int));
6      if (res && mulH && mulV) {
7          for (int i = 0; i < l; i++)
8          for (int k = 0; k < m / 2; k++)
9          mulH[i] = mulH[i] + m1[i][2 * k] * m1[i][2 * k + 1];
10         for (int i = 0; i < n; i++)
11         for (int k = 0; k < m / 2; k++)
12         mulV[i] = mulV[i] + m2[2 * k][i] * m2[2 * k + 1][i];
```

```
13          for (int i = 0; i < l; i++)
14          for (int j = 0; j < n; j++) {
15              res[i][j] = - mulH[i] - mulV[j];
16              for (int k = 0; k < m / 2; k++)
17              res[i][j] = res[i][j] + (m1[i][2 * k] + m2[2 * k +
                    1][j]) * (m1[i][2 * k + 1] + m2[2 * k][j]);
18          }
19          if (m % 2 == 1)
20          for (int i = 0; i < l ; i++)
21          for (int j = 0; j < n ; j++)
22          res[i][j] = res[i][j] + m1[i][m - 1] * m2 [m - 1][j];
23          free(mulH);
24          free(mulV);
25      }
26      else
27      printf("Error!\n");
28      return res;
29  }
```

Листинг 3.3 – Реализация оптимизированного алгоритма умножения матриц Винограда

```
1  int **multVinOptimize(int **m1, int **m2, int l, int m, int n)
2  {
3      int **res = allocateMatrix(l, n);
4      int *mulH = calloc(l, sizeof(int));
5      int *mulV = calloc(n, sizeof(int));
6      if (res && mulH && mulV) {
7          int optm = m / 2;
8          int f;
9          for (int i = 0; i < l; i++)
10         for (int k = 0; k < optm; k++) {
11             f = k << 1;
12             mulH[i] += m1[i][f] * m1[i][f + 1];
13         }
14         for (int i = 0; i < n; i++)
15         for (int k = 0; k < optm; k++) {
16             f = k << 1;
17             mulV[i] += m2[f][i] * m2[f + 1][i];
18         }
19         for (int i = 0; i < l; i++)
20         for (int j = 0; j < n; j++) {
```

```
21              res[i][j] = - mulH[i] - mulV[j];
22              for (int k = 0; k < optm; k++) {
23                  f = k << 1;
24                  res[i][j] += (m1[i][f] + m2[f+ 1][j]) * (m1[i][f
                        + 1] + m2[f][j]);
25              }
26          }
27          if (m % 2 == 1)
28          for (int i = 0; i < l ; i++)
29          for (int j = 0; j < n ; j++)
30          res[i][j] += m1[i][m - 1] * m2 [m - 1][j];
31          free(mulH);
32          free(mulV);
33      }
34      else
35      printf("Error!\n");
36      return res;
37  }
```

3.5 Функциональные тесты

В таблице 3.1 приведены тестовые случаи для алгоритмов умножения матриц. Случаи 1–2 — умножение однострочных/одностолбцовых матриц, 3–5 — квадратных матриц одного размера, 6 — тест на некорректный размер.

Таблица 3.1 – Тестовые случаи

N	Матрица 1	Матрица 2	Результат
1	$\begin{pmatrix} 1 & 1 & 1 \\ 5 & 5 & 5 \\ 2 & 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 15 \\ 6 \end{pmatrix}$
2	$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
3	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$
4	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 4 \end{pmatrix}$
5	$\begin{pmatrix} 1 & -2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 6 \\ 4 & 12 & 18 \\ 4 & 12 & 18 \end{pmatrix}$
6	$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \end{pmatrix}$	Некорректный размер

Вывод

На основе схем из конструкторского раздела были написаны реализации требуемых алгоритмов, а также проведено их тестирование.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система Manjaro xfce [3] Linux [4] x86_64;
- память 8 ГБ;
- мобильный процессор AMD Ryzen™ 7 3700U @ 2.3 ГГц [5].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы: в демонстрации был выбран вариант перемножения случайных матриц корректного размера.

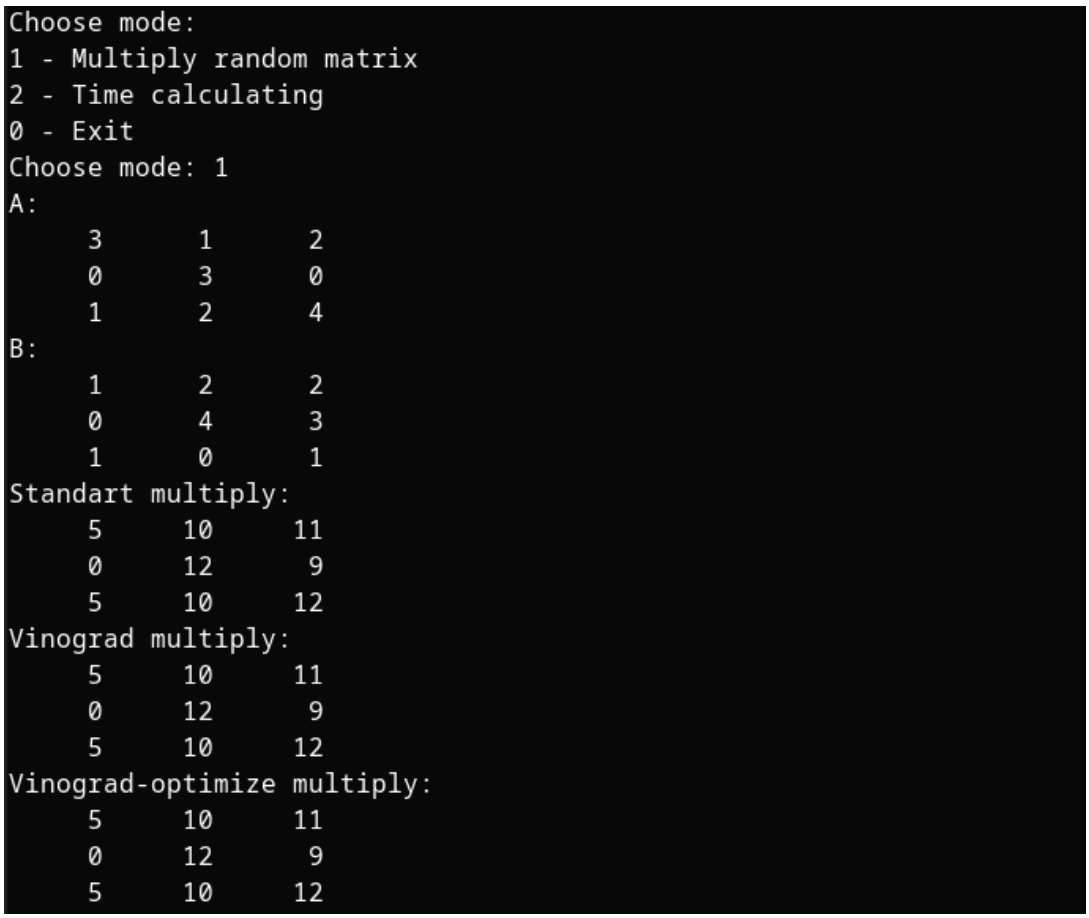

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения реализаций алгоритмов

В таблицах 4.1–4.3 представлены замеры процессорного времени работы алгоритмов на квадратных матрицах размером от 50 до 550 элементов. Такая размерность выбрана, потому что на больших размерах массива происходит переполнение типа счетчика тиков. Каждое значение было получено усреднением по 10 замерам. Примем за лучший случай четную размерность N, за худший — нечетную. Время приведено в тиках процессора.

Таблица 4.1 – Замеры времени работы реализаций алгоритмов для лучшего случая

N	Стандартный	Винограда	оптимизированный Винограда
50	1000000	1000000	1000000
100	15000000	12000000	10000000
150	63000000	52000000	40000000
200	185000000	152000000	120000000
250	418000000	341000000	270000000
300	844000000	693000000	545000000
350	1572000000	1297000000	1023000000
400	2630000000	2168000000	1709000000
450	4187000000	3449000000	2723000000
500	6434000000	5462000000	4481000000
550	9540000000	7948000000	6432000000

Таблица 4.2 – Замеры времени работы реализаций алгоритмов для худшего случая

N	Стандартный	Винограда	оптимизированный Винограда
51	1000000	1000000	1000000
101	18000000	12000000	9000000
151	66000000	51000000	40000000
201	186000000	149000000	117000000
251	443000000	358000000	283000000
301	864000000	707000000	557000000
351	1563000000	1282000000	1011000000
401	2678000000	2188000000	1717000000
451	4491000000	3719000000	2912000000
501	6669000000	5498000000	4318000000
551	9705000000	7972000000	6261000000

На рисунке 4.2 представлены зависимости времени работы различных реализаций алгоритмов от линейного размера квадратных матриц с четной размерностью. В результате эксперимента было получено, что на квадратных матрицах размером до 550 оптимизированный алгоритм Винограда работает на 40 % быстрее стандартного и на 25 % быстрее классического алгоритма Винограда.

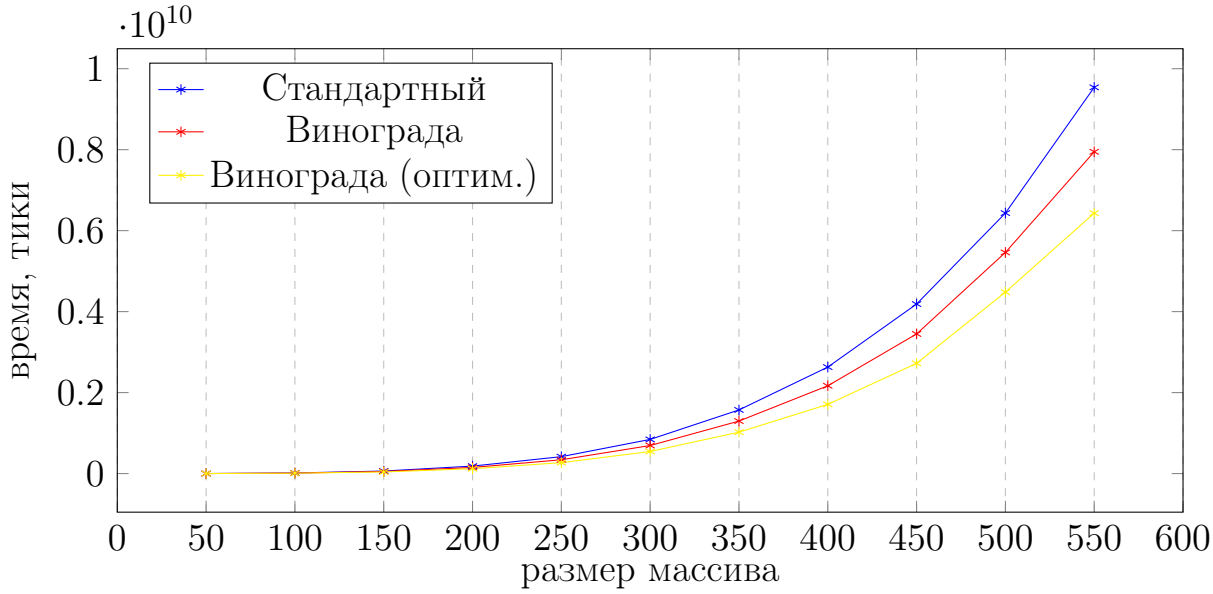

Рисунок 4.2 – Сравнение времени работы реализаций алгоритмов для лучшего случая

На рисунке 4.3 представлены зависимости времени работы различных реализаций алгоритмов от линейного размера квадратных матриц с нечетной размерностью. В результате эксперимента было получено, что на квадратных матрицах размером до 551 реализации алгоритмов показывают примерно такие же результаты.

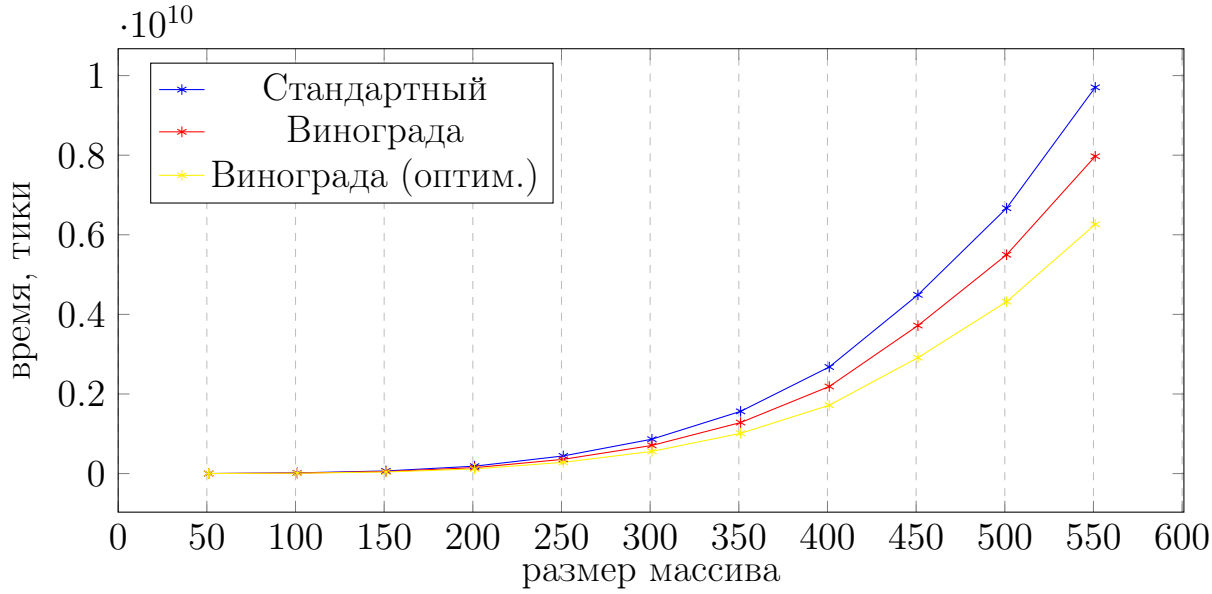

Рисунок 4.3 – Сравнение времени работы реализаций алгоритмов для худшего случая

Вывод

Получены следуюшие результаты для реализации стандартного алгоритма умножения матриц:

- в лучшем случае она на 15 % медленнее реализации алгоритма Винограда и на 30 % медленнее реализации оптимизированного алгоритма Винограда;
- в худшем случае она на 18 % медленнее реализации алгоритма Винограда и на 35 % медленнее реализации оптимизированного алгоритма Винограда.

Также оптимизированный алгоритм Винограда дает выигрыш по затрачиваемому времени на 20 % относительно стандартного алгоритма Винограда независимо от класса эквивалентности данных.

Заключение

В ходе выполнения данной лабораторной работы достигнута поставленная цель:

Были решены все задачи:

- изучены выбранные алгоритмы умножения матриц и способы оптимизации;
- разработан оптимизированный алгоритм Винограда;
- составлены схемы рассмотренных алгоритмов;
- реализованы разработанные алгоритмы умножения матриц;
- проведен сравнительный анализ алгоритмов по затрачиваемым ресурсам (время и память);
- описаны и обоснованы полученные результаты.

В результате исследований можно прийти к выводу, что реализация стандартного алгоритма является самой медленной из рассмотренных. Хотя сложность всех алгоритмов равняется $O(N^3)$, реализация оптимизированного алгоритма Винограда работает быстрее в среднем на 25 % при любых входных данных. По используемой памяти алгоритмы практически не отличаются, за исключением использования двух дополнительных массивов в алгоритме Винограда.

Список использованных источников

[1] Справочник по языку C [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/cpp/c-language/c-language-reference?view=msvc-170 (дата обращения: 04.09.2021).

[2] <sys/time.h> [Электронный ресурс]. Режим доступа: https://pubs.opengroup.org/onlinepubs/7908799/xsh/systime.h.html (дата обращения: 04.09.2021).

[3] Manjaro [Электронный ресурс]. Режим доступа: https://manjaro.org/ (дата обращения: 03.10.2022).

[4] Linux – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Linux (дата обращения: 04.10.2022).

[5] Мобильный процессор AMD Ryzen™ 7 3700U [Электронный ресурс]. Режим доступа: https://www.amd.com/ru/products/apu/amd-ryzen-7-3700u (дата обращения: 04.10.2022).