



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 Алгоритм Коперсмита–Винограда

Студент Кононенко С.С.

Группа ИУ7-53Б

Оценка (баллы) _____

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

Оглавление

Введение	3
1 Аналитическая часть	4
1.1 Стандартный алгоритм	4
1.2 Алгоритм Копперсмита – Винограда	4
2 Конструкторская часть	6
2.1 Разработка алгоритмов	6
2.2 Трудоёмкость алгоритмов	6
2.2.1 Модель вычислений	6
2.2.2 Расчет трудоемкости	7
3 Технологическая часть	14
3.1 Требования к ПО	14
3.2 Средства реализации	14
3.3 Листинг кода	14
3.4 Тестирование функций	19
4 Исследовательская часть	20
4.1 Пример работы	20
4.2 Технические характеристики	20
4.3 Время выполнения алгоритмов	21
Заключение	25
Литература	26

Введение

Алгоритм Копперсмита — Винограда — алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом. В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2,3755})$, где n — размер стороны матрицы. Алгоритм Копперсмита — Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

На практике алгоритм Копперсмита — Винограда не используется, так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров. Поэтому пользуются алгоритмом Штрассена по причинам простоты реализации и меньшей константе в оценке трудоемкости.

Алгоритм Штрассена предназначен для быстрого умножения матриц. Он был разработан Фолькером Штрассеном в 1969 году и является обобщением метода умножения Карацубы на матрицы.

В отличие от традиционного алгоритма умножения матриц, алгоритм Штрассена умножает матрицы за время $\Theta(n^{\log_2 7}) = O(n^{2,81})$, что даёт выигрыш на больших плотных матрицах начиная, примерно, от 64×64 .

Несмотря на то, что алгоритм Штрассена является асимптотически не самым быстрым из существующих алгоритмов быстрого умножения матриц, он проще программируется и эффективнее при умножении матриц относительно малого размера.

Задачи лабораторной работы:

- реализовать классический алгоритм умножения матриц;
- реализовать алгоритм Копперсмита — Винограда;
- реализовать улучшенный Алгоритм Копперсмита — Винограда;
- рассчитать их трудоемкость;
- сравнить их временные характеристики экспериментально;
- на основании проделанной работы сделать выводы.

1 Аналитическая часть

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы

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

тогда матрица C

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

где

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

будет называться произведением матриц A и B . Стандартный алгоритм реализует данную формулу.

1.2 Алгоритм Копперсмита – Винограда

В результате умножения двух матриц, каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $V \cdot W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4$, что эквивалентно

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4. \quad (1.4)$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырех умножений - шесть, а вместо трех сложений - десять, 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, то для каждого элемента будет необходимо выполнить лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. Из-за того, что операция сложения быстрее операции умножения, алгоритм должен работать быстрее стандартного [1].

Вывод

Были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которых — наличие предварительной обработки, а также количество операций умножения.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2, 2.3 приведены схемы алгоритмов простого умножения матриц, умножения матриц по Копперсмит-Винограду и улучшенного умножения матриц по Копперсмит-Винограду соответственно.

Из рисунка 2.2 можно заметить, что для алгоритма Винограда худшим случаем являются матрицы с нечётным общим размером, а лучшим - с чётным, так как отпадает необходимость в последнем цикле.

Данный алгоритм можно оптимизировать:

- 1) заменой операции деления на 2 побитовым сдвигом на 1 вправо;
- 2) заменой выражения вида $a = a + \dots$ на $a += \dots$;
- 3) сделав в циклах по k шаг 2, избавившись тем самым от двух операций умножения на каждую итерацию.

2.2 Трудоёмкость алгоритмов

2.2.1 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1. $+$, $-$, $/$, $\%$, $==$, $!=$, $<$, $>$, $<=$, $>=$, $[]$, $*$, $++$, $--$ – трудоемкость 1.
2. Трудоемкость оператора выбора `if условие then A else B` рассчитывается, как

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \quad (2.1)$$

3. Трудоемкость цикла рассчитывается, как $f_{for} = f_{инициализации} + f_{сравнения} + N(f_{тела} + f_{инициализации} + f_{сравнения})$.
4. Трудоемкость вызова функции равна 0.

2.2.2 Расчет трудоемкости

Оценка трудоемкости проведена по схемам 2.1, 2.2, 2.3 для алгоритмов простого умножения матриц, умножения матриц по Копперсмит-Винограду и улучшенного умножения по Копперсмит-Винограду соответственно.

Вычисление трудоемкости алгоритма простого умножения матриц

Трудоёмкость алгоритма простого умножения матриц состоит из:

- 1) внешнего цикла по $i \in [1..M]$, трудоёмкость которого: $f = 2 + M \cdot (2 + f_{body})$;
- 2) цикла по $j \in [1..N]$, трудоёмкость которого: $f = 2 + N \cdot (2 + f_{body})$;
- 3) цикла по $k \in [1..K]$, трудоёмкость которого: $f = 2 + 10K$;

Учитывая, что трудоёмкость простого алгоритма равна трудоёмкости внешнего цикла, можно вычислить ее, подставив циклы тела (2.2):

$$f_{standard} = 2 + M \cdot (4 + N \cdot (4 + 10K)) = 2 + 4M + 4MN + 10MNK \approx 10MNK \quad (2.2)$$

Вычисление трудоёмкости алгоритма умножения матриц по Копперсмит-Винограду

Трудоёмкость алгоритма Копперсмита — Винограда состоит из:

- 1) создания и инициализации массивов MH и MV , трудоёмкость которого (2.3):

$$f_{init} = M + N; \quad (2.3)$$

- 2) заполнения массива MH , трудоёмкость которого (2.4):

$$f_{MH} = 3 + \frac{K}{2} \cdot (5 + 12M); \quad (2.4)$$

- 3) заполнения массива MV , трудоёмкость которого (2.5):

$$f_{MV} = 3 + \frac{K}{2} \cdot (5 + 12N); \quad (2.5)$$

- 4) цикла заполнения для чётных размеров, трудоёмкость которого (2.6):

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{25}{2} \cdot K)); \quad (2.6)$$

- 5) цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный, трудоёмкость которого (2.7):

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + M \cdot (4 + 14N), & \text{иначе.} \end{cases} \quad (2.7)$$

Итого, для худшего случая (нечётный общий размер матриц) имеем (2.8):

$$f = M + N + 12 + 8M + 5K + 6MK + 6NK + 25MN + \frac{25}{2}MNK \approx 12.5 \cdot MNK \quad (2.8)$$

Для лучшего случая (чётный общий размер матриц) имеем (2.9):

$$f = M + N + 10 + 4M + 5K + 6MK + 6NK + 11MN + \frac{25}{2}MNK \approx 12.5 \cdot MNK \quad (2.9)$$

Вычисление трудоёмкости улучшенного алгоритма умножения матриц по Копперсмит–Винограду

Трудоёмкость улучшенного алгоритма Копперсмита — Винограда состоит из:

- 1) создания и инициализации массивов MH и MV , трудоёмкость которого (2.10):

$$f_{init} = M + N; \quad (2.10)$$

- 2) заполнения массива MH , трудоёмкость которого (2.11):

$$f_{MH} = 2 + \frac{K}{2} \cdot (4 + 8M); \quad (2.11)$$

- 3) заполнения массива MV , трудоёмкость которого (2.12):

$$f_{MV} = 2 + \frac{K}{2} \cdot (4 + 8N); \quad (2.12)$$

- 4) цикла заполнения для чётных размеров, трудоёмкость которого (2.13):

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (8 + 9K)) \quad (2.13)$$

- 5) цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный, трудоёмкость которого (2.14):

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + M \cdot (4 + 12N), & \text{иначе.} \end{cases} \quad (2.14)$$

Итого, для худшего случая (нечётный общий размер матриц) имеем (2.15):

$$f = M + N + 10 + 4K + 4KN + 4KM + 8M + 20MN + 9MNK \approx 9MNK \quad (2.15)$$

Для лучшего случая (чётный общий размер матриц) имеем (2.16):

$$f = M + N + 8 + 4K + 4KN + 4KM + 4M + 8MN + 9MNK \approx 9MNK \quad (2.16)$$

Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов и проведена теоретическая оценка трудоемкости.

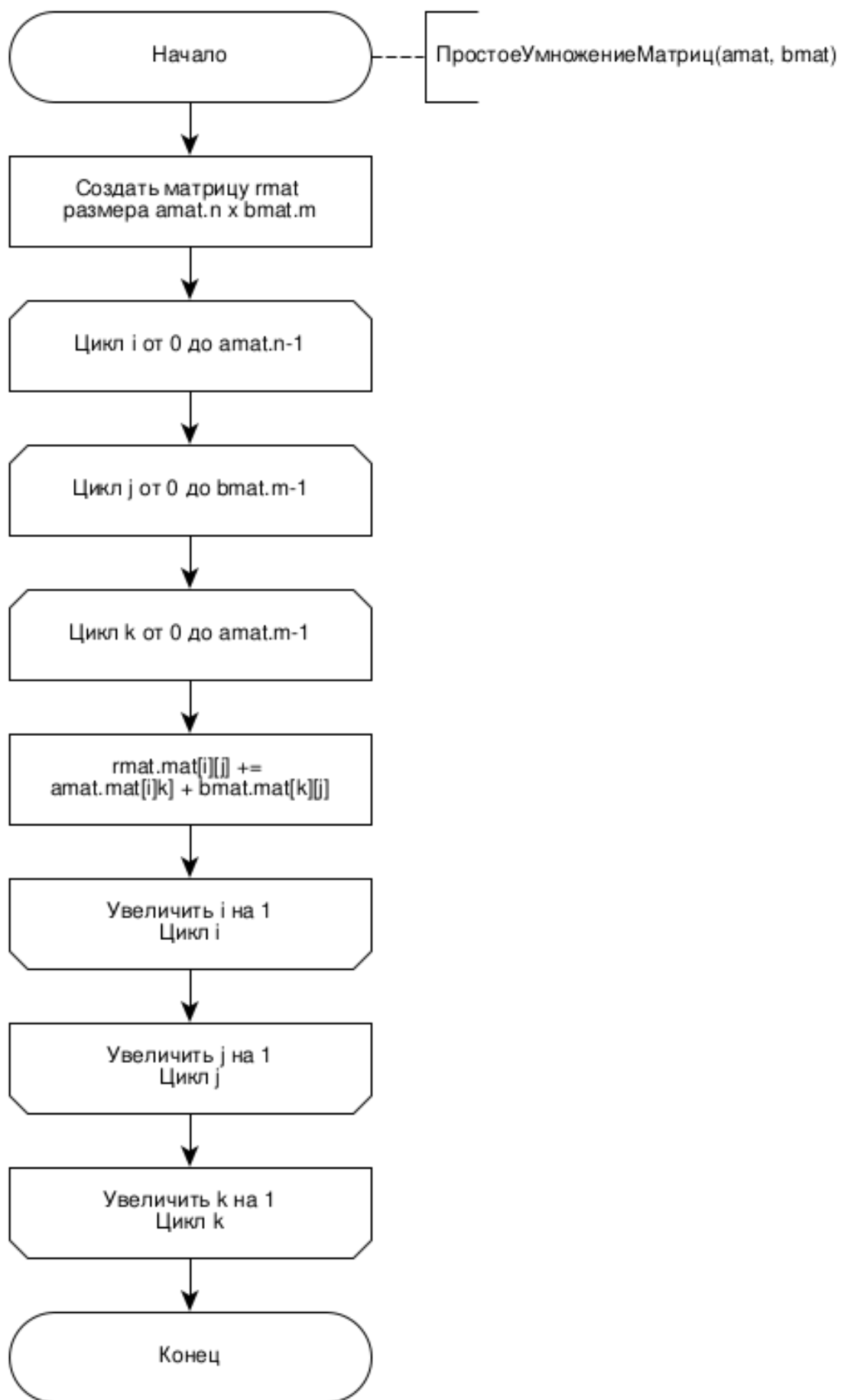


Рисунок 2.1 – Схема алгоритма простого умножения матриц

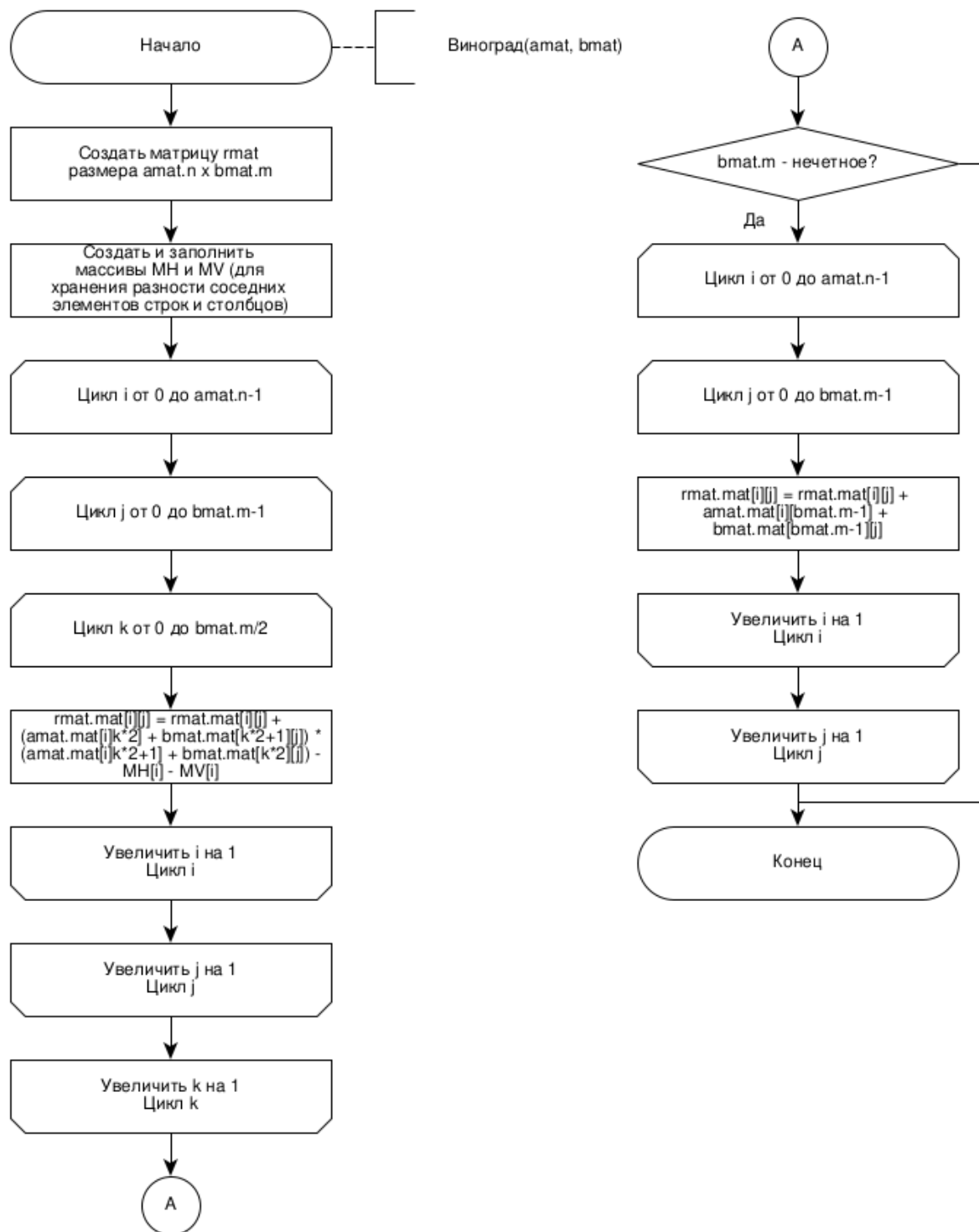


Рисунок 2.2 – Схема алгоритма умножения матриц по Копперсмит-Винограду

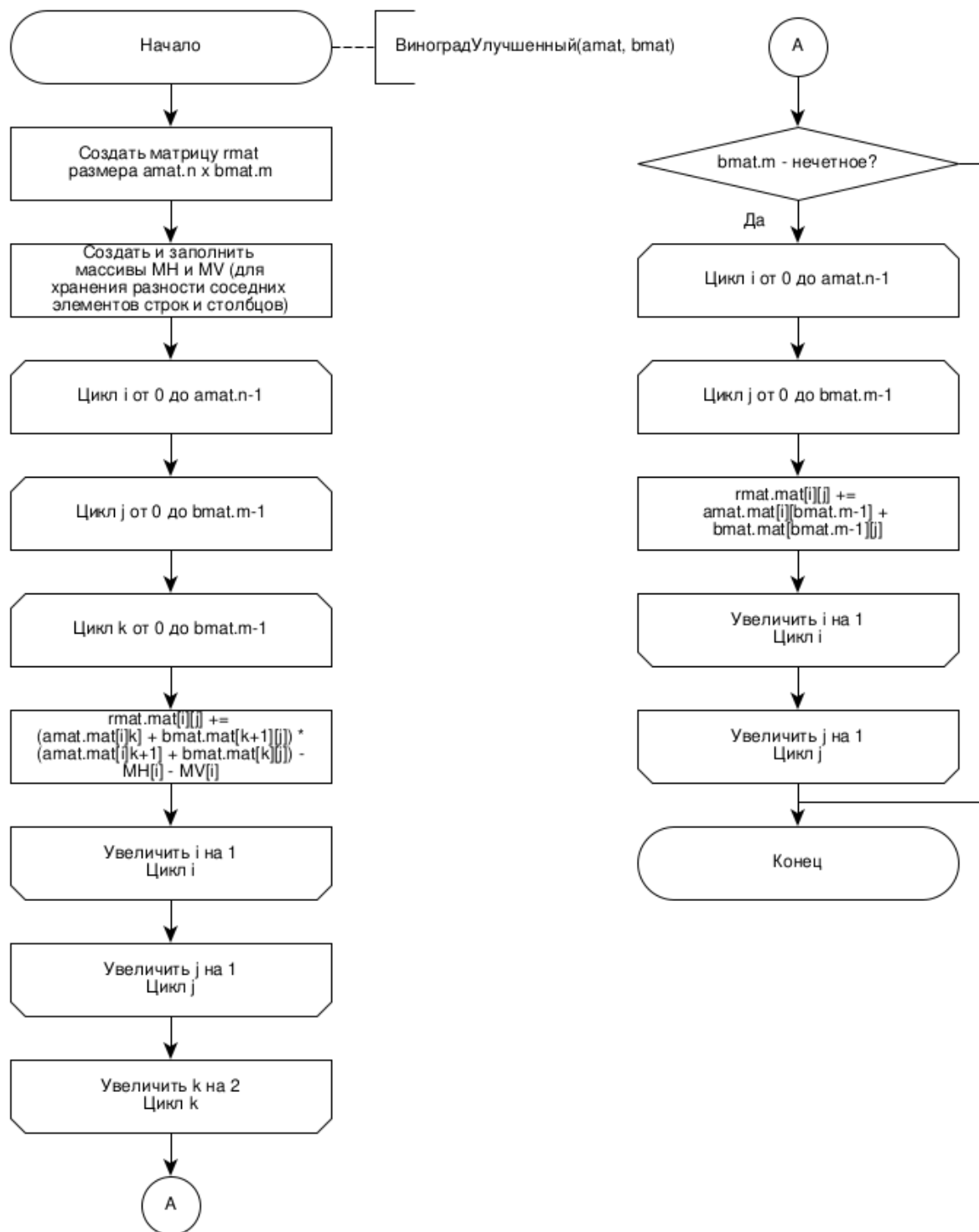


Рисунок 2.3 – Схема алгоритма улучшенного умножения матриц по Копперсмиту–Винограду

3 Технологическая часть

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

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаются размеры матриц (целые числа) и сами матрицы (элементы которых - целые числа), которые нужно перемножить;
- на выходе — результаты умножения матриц алгоритмами простого умножения матриц, умножения матриц по Копперсмити–Винограду и улучшенного умножения матриц по Копперсмити–Винограду.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран многопоточный язык Golang [2]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка. Помимо этого, встроенные средства языка предоставляют высокоточные средства тестирования разработанного ПО.

3.3 Листинг кода

В листингах 3.1, 3.2, 3.3 приведены реализации алгоритмов простого умножения матриц, умножения матриц по Копперсмити–Винограду и улучшенного умножения матриц по Копперсмити–Винограду соответственно.

В листинге 3.4 приведены реализации вспомогательных функций.

Листинг 3.1 – Алгоритм простого умножения матриц

```
1 func SimpleMult(amat, bmat MInt) MInt {
```

```

2   rmat := formResMat(amat.n, bmat.m)
3
4   for i := 0; i < rmat.n; i++ {
5       for j := 0; j < rmat.m; j++ {
6           for k := 0; k < amat.m; k++ {
7               rmat.mat[i][j] += amat.mat[i][k] * bmat.mat[k][j]
8           }
9       }
10  }
11
12  return rmat
13 }

```

Листинг 3.2 – Алгоритм умножения матриц по Копперсмиту–Винограду

```

1 func WinogradMult(amat, bmat MInt) MInt {
2     rmat := formResMat(amat.n, bmat.m)
3
4     rowcf := precomputeWinogradRows(amat)
5     colcf := precomputeWinogradCols(bmat)
6
7     for i := 0; i < rmat.n; i++ {
8         for j := 0; j < rmat.m; j++ {
9             for k := 0; k < rmat.m/2; k++ {
10                rmat.mat[i][j] +=
11                    (amat.mat[i][k*2]+bmat.mat[k*2+1][j])*(amat.mat[i][k*2+1]+bmat.mat[k*2][j])
12                    -
13                    rowcf.mat[i][k] - colcf.mat[k][j]
14            }
15        }
16    }
17
18    if rmat.m%2 != 0 {
19        for i := 0; i < rmat.n; i++ {
20            for j := 0; j < rmat.m; j++ {
21                rmat.mat[i][j] += amat.mat[i][amat.m-1] * bmat.mat[bmat.n-1][j]
22            }
23        }
24    }
25
26    return rmat
27 }
28
29 func precomputeWinogradRows(mat MInt) MInt {
30     s := mat.m / 2
31     cf := formResMat(mat.n, s)
32
33     for i := 0; i < mat.n; i++ {
34         for j := 0; j < mat.m/2; j++ {

```

```

33         cf.mat[i][j] = mat.mat[i][j*2] * mat.mat[i][j*2+1]
34     }
35 }
36
37 return cf
38 }
39
40 func precomputeWinogradCols(mat MInt) MInt {
41     s := mat.n / 2
42     cf := formResMat(s, mat.m)
43
44     for i := 0; i < mat.n/2; i++ {
45         for j := 0; j < mat.m; j++ {
46             cf.mat[i][j] = mat.mat[i*2][j] * mat.mat[i*2+1][j]
47         }
48     }
49
50     return cf
51 }

```

Листинг 3.3 – Алгоритм улучшенного умножения матриц по
Копперсмиту–Винограду

```

1 func WinogradMultImp(amat, bmat MInt) MInt {
2     rmat := formResMat(amat.n, bmat.m)
3
4     rowcf := precomputeWinogradRowsImp(amat)
5     colcf := precomputeWinogradColsImp(bmat)
6
7     for i := 0; i < rmat.n; i++ {
8         for j := 0; j < rmat.m; j++ {
9             for k := 0; k < rmat.m-1; k += 2 {
10                 l := k/2 + k%2
11                 rmat.mat[i][j] +=
12                     (amat.mat[i][k]+bmat.mat[k+1][j])*(amat.mat[i][k+1]+bmat.mat[k][j]) -
13                     rowcf.mat[i][l] - colcf.mat[l][j]
14             }
15         }
16
17         if rmat.m%2 != 0 {
18             k := amat.m - 1
19             for i := 0; i < rmat.n; i++ {
20                 for j := 0; j < rmat.m; j++ {
21                     rmat.mat[i][j] += amat.mat[i][k] * bmat.mat[k][j]
22                 }
23             }
24         }
25     }
26 }

```



```

25
26     return rmat
27 }
28
29 func precomputeWinogradRowsImp(mat MInt) MInt {
30     s := mat.m / 2
31     cf := formResMat(mat.n, s)
32
33     for i := 0; i < mat.n; i++ {
34         for j := 0; j < mat.m-1; j += 2 {
35             cf.mat[i][j/s+j%2] = mat.mat[i][j] * mat.mat[i][j+1]
36         }
37     }
38
39     return cf
40 }
41
42 func precomputeWinogradColsImp(mat MInt) MInt {
43     s := mat.n / 2
44     cf := formResMat(s, mat.m)
45
46     for i := 0; i < mat.n-1; i += 2 {
47         for j := 0; j < mat.m; j++ {
48             cf.mat[i/s+i%2][j] = mat.mat[i][j] * mat.mat[i+1][j]
49         }
50     }
51
52     return cf
53 }

```

Листинг 3.4 – Вспомогательные функции

```

1 package matrix
2
3 import (
4     "fmt"
5     "math/rand"
6 )
7
8 // ReadMatrix used to read matrix with n rows and m columns.
9 func ReadMatrix(n, m int) MInt {
10     mat := formResMat(n, m)
11
12     for i := 0; i < mat.n; i++ {
13         for j := 0; j < mat.m; j++ {
14             fmt.Scanf("%d", &mat.mat[i][j])
15         }
16     }
17

```

```

18     return mat
19 }
20
21 // ReadNum used to read a word through EOL symbol.
22 func ReadNum() int {
23     var num int
24
25     fmt.Scanln(&num)
26
27     return num
28 }
29
30 func randomFill(mat MInt, max int) {
31     for i := 0; i < mat.n; i++ {
32         for j := 0; j < mat.m; j++ {
33             mat.mat[i][j] = rand.Intn(max)
34         }
35     }
36 }
37
38 func formResMat(n, m int) MInt {
39     var rmat MInt
40
41     rmat.n = n
42     rmat.m = m
43     rmat.mat = make([] []int, rmat.n)
44
45     for i := range rmat.mat {
46         rmat.mat[i] = make([]int, rmat.m)
47     }
48
49     return rmat
50 }

```

3.4 Тестирование функций

В таблице 3.1 приведены функциональные тесты для функций, реализующих простой алгоритм умножения матриц, алгоритм умножения матриц по Копперсмитту–Винограду и улучшенный алгоритм умножения матриц по Копперсмитту–Винограду. Все тесты пройдены успешно.

Таблица 3.1 – Тестирование функций

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 6 & 12 & 18 \\ 6 & 12 & 18 \\ 6 & 12 & 18 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 3 & 6 \\ 3 & 6 \end{pmatrix}$
(2)	(2)	(4)
$\begin{pmatrix} 1 & -2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 6 \\ 4 & 12 & 18 \\ 4 & 12 & 18 \end{pmatrix}$
(1 2)	(1 2)	Не могут быть перемножены

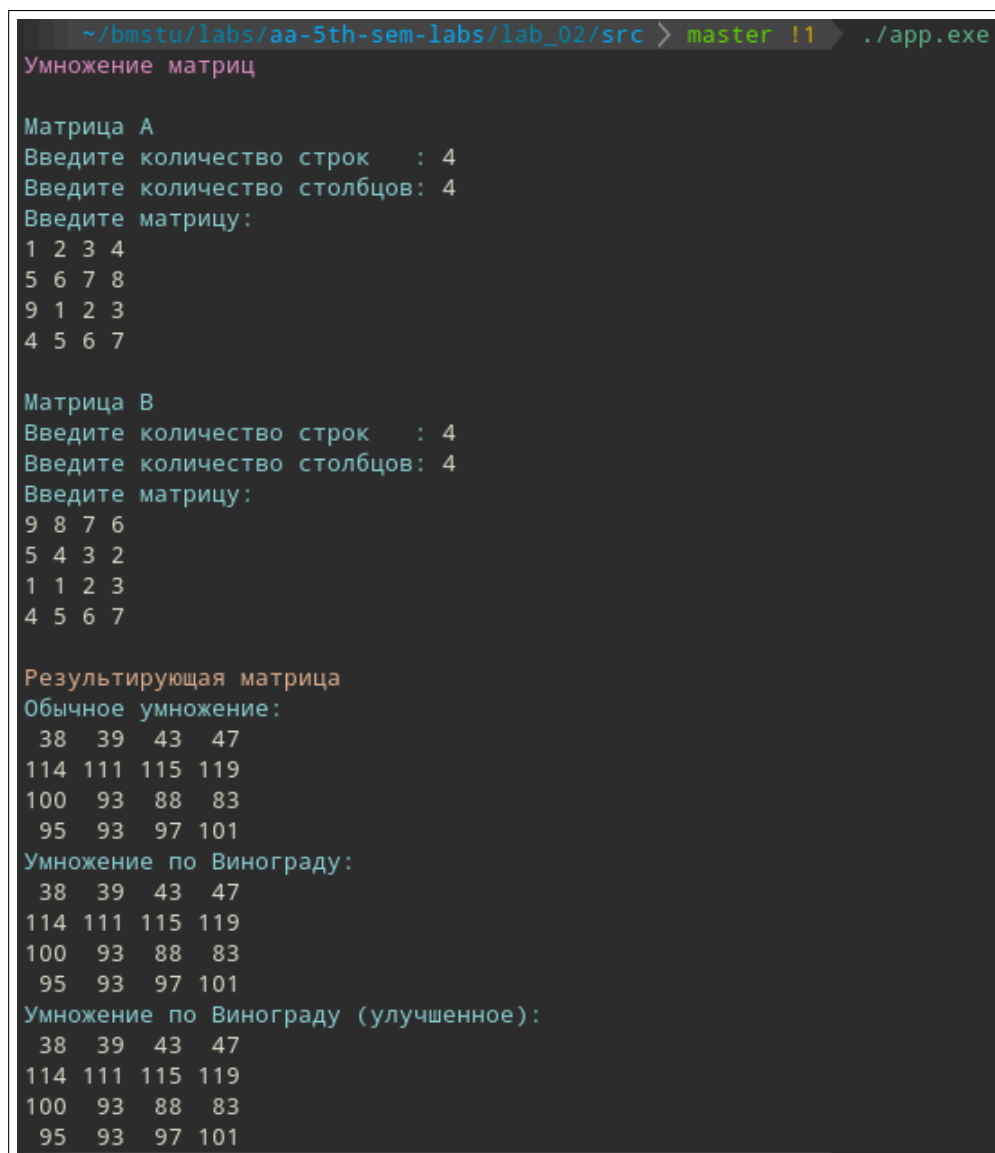
Вывод

Были разработаны и протестированы реализации алгоритмов: простой алгоритм умножения матриц, алгоритм умножения матриц по Копперсмитту–Винограду и улучшенный алгоритм умножения матриц по Копперсмитту–Винограду.

4 Исследовательская часть

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.



```
~/bmstu/labs/aa-5th-sem-labs/lab_02/src > master !1 > ./app.exe
Умножение матриц

Матрица A
Введите количество строк : 4
Введите количество столбцов: 4
Введите матрицу:
1 2 3 4
5 6 7 8
9 1 2 3
4 5 6 7

Матрица B
Введите количество строк : 4
Введите количество столбцов: 4
Введите матрицу:
9 8 7 6
5 4 3 2
1 1 2 3
4 5 6 7

Результирующая матрица
Обычное умножение:
38 39 43 47
114 111 115 119
100 93 88 83
95 93 97 101
Умножение по Винограду:
38 39 43 47
114 111 115 119
100 93 88 83
95 93 97 101
Умножение по Винограду (улучшенное):
38 39 43 47
114 111 115 119
100 93 88 83
95 93 97 101
```

Рисунок 4.1 – Демонстрация работы алгоритмов умножения матриц

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Manjaro [3] Linux [4] 20.1 64-битная.
- Память: 16 ГБ.
- Процессор: AMD Ryzen™ 7 3700U [5] ЦПУ @ 2.30ГГц

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

4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи написания «бенчмарков» [6], предоставляемых встроенными в Go средствами. Для такого рода тестирования не нужно самостоятельно указывать количество повторов. В библиотеке для тестирования существует константа N , которая динамически варьируется в зависимости от того, был ли получен стабильный результат или нет.

В листинге 4.1 пример реализации бенчмарка.

Листинг 4.1 – Реализация бенчмарка

```
1 package matrix
2
3 import (
4     "testing"
5 )
6
7 // Simple multiplication benchmarks.
8
9 func BenchmarkSimple100(b *testing.B) {
10     N := 100
11     amat := formResMat(N, N)
12     randomFill(amat, 100)
13     bmat := formResMat(N, N)
14     randomFill(bmat, 100)
15
16     for i := 0; i < b.N; i++ {
17         SimpleMult(amat, bmat)
18     }
19 }
```

Результаты замеров приведены в таблицах 4.1, 4.2. На рисунках 4.2, 4.3 приведены графики, иллюстрирующие зависимость времени работы алгоритмов умножения от размеров матриц.

Таблица 4.1 – Время работы реализаций алгоритмов при чётных размерах матриц

Размер матрицы	Время, нс		
	С	КВ	УКВ
100	2169144	2424132	1811016
200	17680401	22032658	15485909
300	65838464	76854395	55530661
400	210892129	207347115	162146803
500	364922866	350837776	327348293
600	666673788	655599427	631239232
700	1369979155	1332273841	1207834761
800	3205123980	3108582931	2921219674

Таблица 4.2 – Время работы реализаций алгоритмов при нечётных размерах матриц

Размер матрицы	Время, нс		
	С	КВ	УКВ
101	2259672	2796926	2018691
201	20071583	21688038	18481951
301	74128394	77705821	61183913
401	221489215	218382194	158158291
501	371283268	372238416	361183214
601	672286913	671599427	648239232
701	1382138491	1384273841	1324143862
801	3291482918	3389582931	3217248194

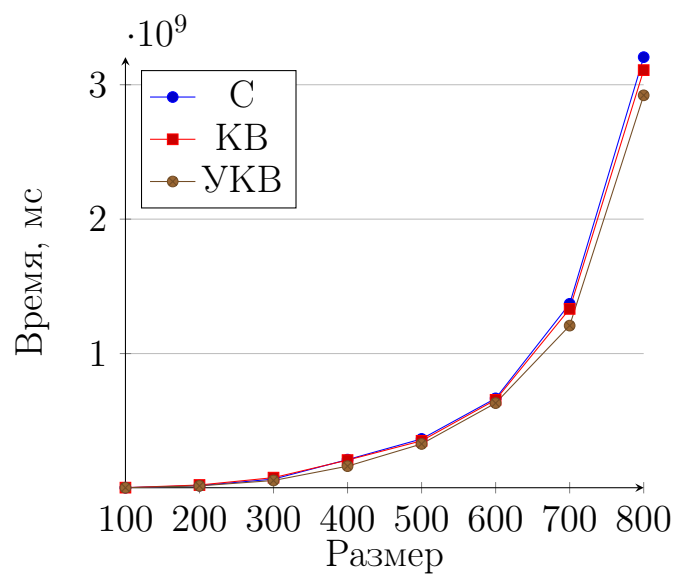


Рисунок 4.2 – Зависимость времени работы реализаций алгоритмов от чётного размера квадратной матрицы

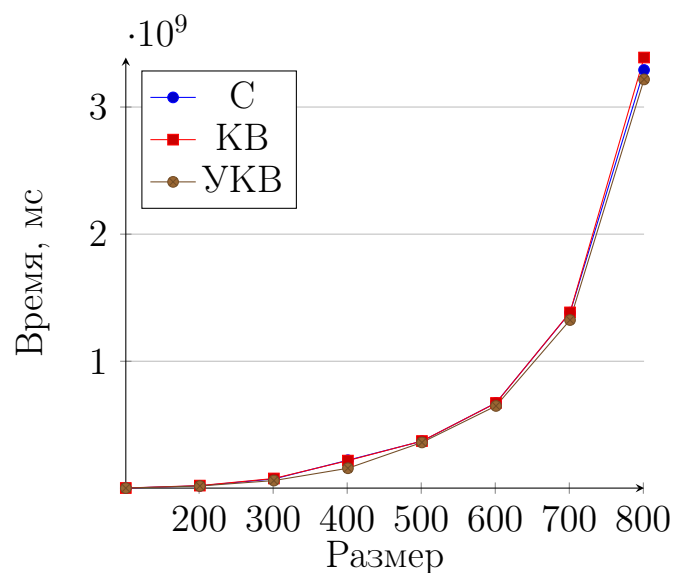


Рисунок 4.3 – Зависимость времени работы реализаций алгоритмов от нечётного размера квадратной матрицы

Вывод

Время работы реализации алгоритма Копперсмита–Винограда незначительно меньше времени работы реализации простого алгоритма умножения, однако при размере 800×800 время вычислений реализации алгоритма Копперсмита–Винограда на 0.3 секунды меньше, нежели у реализации простого алгоритма (что составляет в данном случае примерно 10%).

Заключение

В ходе выполнения работы была достигнута цель выполнены все поставленные задачи:

- реализовать классический алгоритм умножения матриц;
- реализовать алгоритм Копперсмита — Винограда;
- реализовать улучшенный Алгоритм Копперсмита — Винограда;
- рассчитать их трудоемкость;
- сравнить их временные характеристики экспериментально;
- на основании проделанной работы сделать выводы.

Экспериментально были установлены различия в производительности различных алгоритмов умножения матриц. Улучшенный алгоритм Винограда имеет меньшую сложность, нежели простой алгоритм перемножения матриц, что при размерах матриц 800×800 дало практический выигрыш во времени в 10%.

Литература

- [1] Погорелов Д. А. Таразанов А. М. Волкова Л. Л. Оптимизация классического алгоритма Винограда для перемножения матриц // Журнал №1. 2019. Т. 49.
- [2] The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/> (дата обращения: 11.09.2020).
- [3] Manjaro – enjoy the simplicity [Электронный ресурс]. Режим доступа: <https://manjaro.org/> (дата обращения: 14.09.2020).
- [4] Linux – Википедия [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Linux> (дата обращения: 14.09.2020).
- [5] Мобильный процессор AMD Ryzen™ 7 3700U с графикой Radeon™ RX Vega 10 [Электронный ресурс]. Режим доступа: <https://www.amd.com/ru/products/apu/amd-ryzen-7-3700u> (дата обращения: 14.09.2020).
- [6] testing – The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/pkg/testing/> (дата обращения: 12.09.2020).