



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 по дисциплине "Анализ алгоритмов"

Тема Умножение матриц

Студент Романов А.В.

Группа ИУ7-53Б

Оценка (баллы) _____

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

Оглавление

Введение	2
1 Аналитическая часть	3
1.1 Стандартный алгоритм	3
1.2 Алгоритм Копперсмита – Винограда	3
1.3 Вывод	4
2 Конструкторская часть	5
2.1 Схемы алгоритмов	5
2.2 Модель вычислений	5
2.3 Трудоёмкость алгоритмов	6
2.3.1 Стандартный алгоритм умножения матриц	6
2.3.2 Алгоритм Копперсмита — Винограда	6
2.3.3 Оптимизированный алгоритм Копперсмита — Винограда	7
2.4 Вывод	7
3 Технологическая часть	14
3.1 Требование к ПО	14
3.2 Средства реализации	14
3.3 Реализация алгоритмов	14
3.4 Тестовые данные	16
3.5 Вывод	17
4 Исследовательская часть	18
4.1 Технические характеристики	18
4.2 Время выполнения алгоритмов	18
4.3 Вывод	19
Заключение	20
Литература	20

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 учётом 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.

Задачи лабораторной работы:

1. Изучение и реализация трёх алгоритмов умножения матриц: обычный, Копперсмита-Винограда, оптимизированный Копперсмита-Винограда;
2. Сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
3. Сравнительный анализ алгоритмов на основе экспериментальных данных.

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

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.

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 (1.4):

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

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырёх умножений - шесть, а вместо трёх 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, что позволит для каждого элемента выполнять лишь два умножения и пять сложений, складывая затем только лишь с 2 предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Из-за того, что операция сложения быстрее операции умножения в ЭВМ, на практике алгоритм должен работать быстрее стандартного.

1.3 Вывод

В данном разделе были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которого от классического алгоритма — наличие предварительной обработки, а также количество операций умножения.

2 | Конструкторская часть

2.1 Схемы алгоритмов

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц.

На рисунках 2.2, 2.3 и 2.4 представлена схема алгоритма Копперсмита — Винограда.

На рисунках 2.5 и 2.6 представлена схема оптимизированного алгоритма Копперсмита — Винограда.

Для алгоритма Копперсмита – Винограда худшим случаем являются матрицы с нечётным общим размером, а лучшим - с чётным, из-за того что отпадает необходимость в последнем цикле.

Данный алгоритм можно оптимизировать [1]:

- Предварительно получить строки столбцы соответствующих матриц;
- Замена вызова функции вычисления длины вектора на заранее вычисленное значение;
- Вынос конструкции if-then-else за пределы лямбда-функции;
- Ускорение работы для матриц с нечетной общей размерностью.

2.2 Модель вычислений

Для последующего вычисления трудоемкости введём модель вычислений:

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

$$+, -, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \quad (2.1)$$

2. Трудоемкость оператора выбора if условие then A else B рассчитывается, как (2.2).

$$f_{if} = f_{условия} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \quad (2.2)$$

3. Трудоемкость цикла рассчитывается, как (2.3).

$$f_{for} = f_{инициализации} + f_{сравнения} + N(f_{тела} + f_{инкремента} + f_{сравнения}) \quad (2.3)$$

4. Трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

2.3.1 Стандартный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма умножения матриц состоит из:

- Внешнего цикла по $i \in [1..A]$, трудоёмкость которого: $f = 2 + A \cdot (2 + f_{body})$;
- Цикла по $j \in [1..C]$, трудоёмкость которого: $f = 2 + C \cdot (2 + f_{body})$;
- Скалярного умножения двух векторов - цикл по $k \in [1..B]$, трудоёмкость которого: $f = 2 + 10B$;

Трудоёмкость стандартного алгоритма равна трудоёмкости внешнего цикла, можно вычислить ее, подставив циклы тела (2.4):

$$f_{base} = 2 + A \cdot (4 + C \cdot (4 + 10B)) = 2 + 4A + 4AC + 10ABC \approx 10ABC \quad (2.4)$$

2.3.2 Алгоритм Копперсмита — Винограда

Трудоёмкость алгоритма Копперсмита — Винограда состоит из:

1. Создания векторов rows и cols (2.5):

$$f_{create} = A + C; \quad (2.5)$$

2. Заполнения вектора rows (2.6):

$$f_{rows} = 3 + \frac{B}{2} \cdot (5 + 12A); \quad (2.6)$$

3. Заполнения вектора cols (2.7):

$$f_{cols} = 3 + \frac{B}{2} \cdot (5 + 12C); \quad (2.7)$$

4. Цикла заполнения матрицы для чётных размеров (2.8):

$$f_{cycle} = 2 + A \cdot (4 + C \cdot (11 + \frac{25}{2} \cdot B)); \quad (2.8)$$

5. Цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный (2.9):

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + A \cdot (4 + 14C), & \text{иначе.} \end{cases} \quad (2.9)$$

Итого, для худшего случая (нечётный размер матриц):

$$f_{wino_w} = A + C + 12 + 8A + 5B + 6AB + 6CB + 25AC + \frac{25}{2}ABC \approx 12.5 \cdot MNK \quad (2.10)$$

Для лучшего случая (чётный размер матриц):

$$f_{wino_b} = A + C + 10 + 4A + 5B + 6AB + 6CB + 11AC + \frac{25}{2}ABC \approx 12.5 \cdot MNK \quad (2.11)$$

2.3.3 Оптимизированный алгоритм Копперсмита — Винограда

Трудоёмкость улучшенного алгоритма Копперсмита — Винограда состоит из:

1. Создания векторов `rows` и `cols` (2.12):

$$f_{init} = A + C; \quad (2.12)$$

2. Заполнения вектора `rows` (2.13):

$$f_{rows} = 2 + \frac{B}{2} \cdot (4 + 8A); \quad (2.13)$$

3. Заполнения вектора `cols` (2.14):

$$f_{cols} = 2 + \frac{B}{2} \cdot (4 + 8A); \quad (2.14)$$

4. Цикла заполнения матрицы для чётных размеров (2.15):

$$f_{cycle} = 2 + A \cdot (4 + C \cdot (8 + 9B)) \quad (2.15)$$

5. Цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный (2.16):

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + A \cdot (4 + 12C), & \text{иначе.} \end{cases} \quad (2.16)$$

Итого, для худшего случая (нечётный общий размер матриц) имеем (2.17):

$$f = A + C + 10 + 4B + 4BC + 4BA + 8A + 20AC + 9ABC \approx 9ABC \quad (2.17)$$

Для лучшего случая (чётный общий размер матриц) имеем (2.18):

$$f = A + C + 8 + 4B + 4BC + 4BA + 4A + 8AC + 9ABC \approx 9ABC \quad (2.18)$$

2.4 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы обоих алгоритмов умножения матриц. Оценены их трудоёмкости в лучшем и худшем случаях.

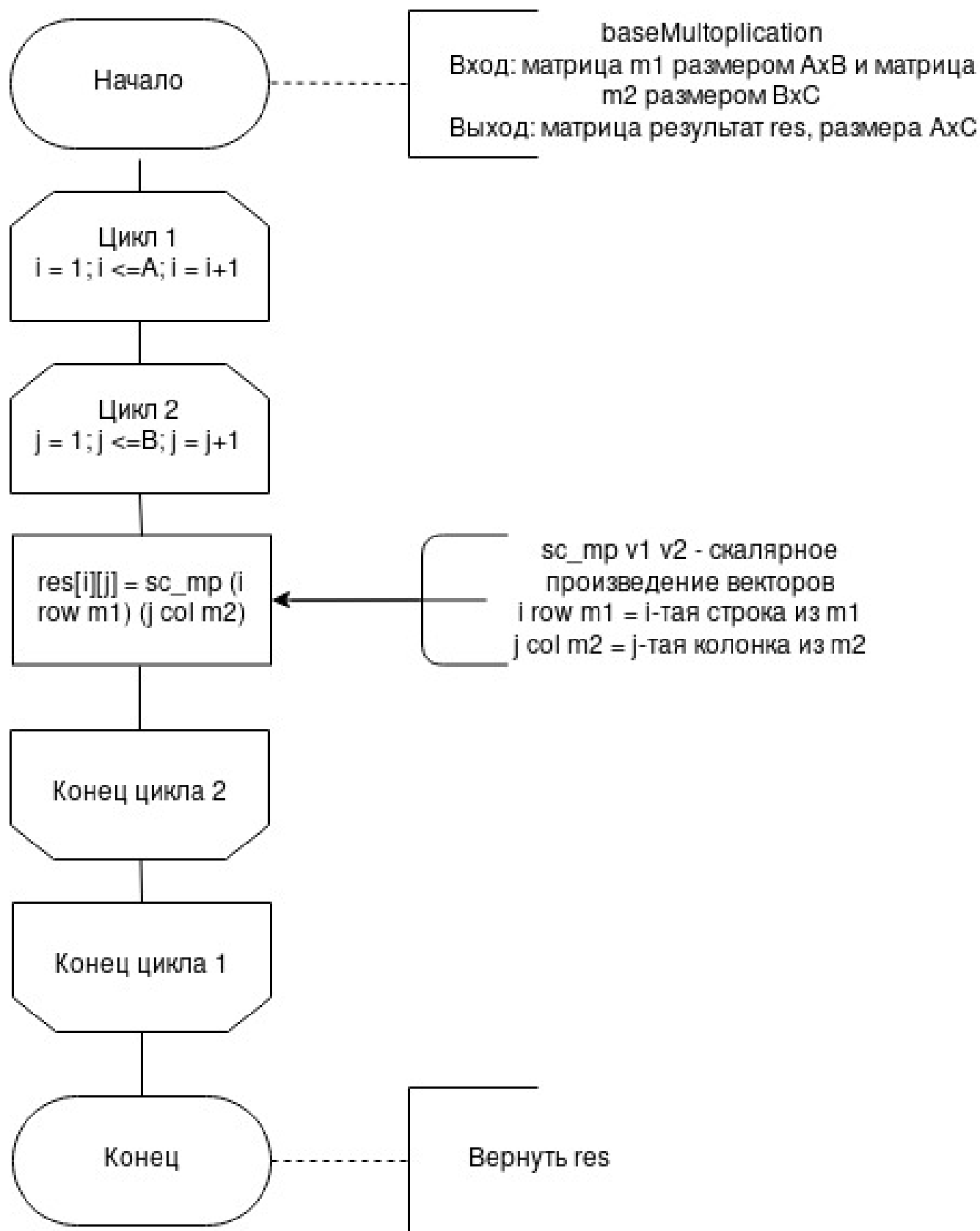


Рис. 2.1: Схема стандартного алгоритма умножения матриц

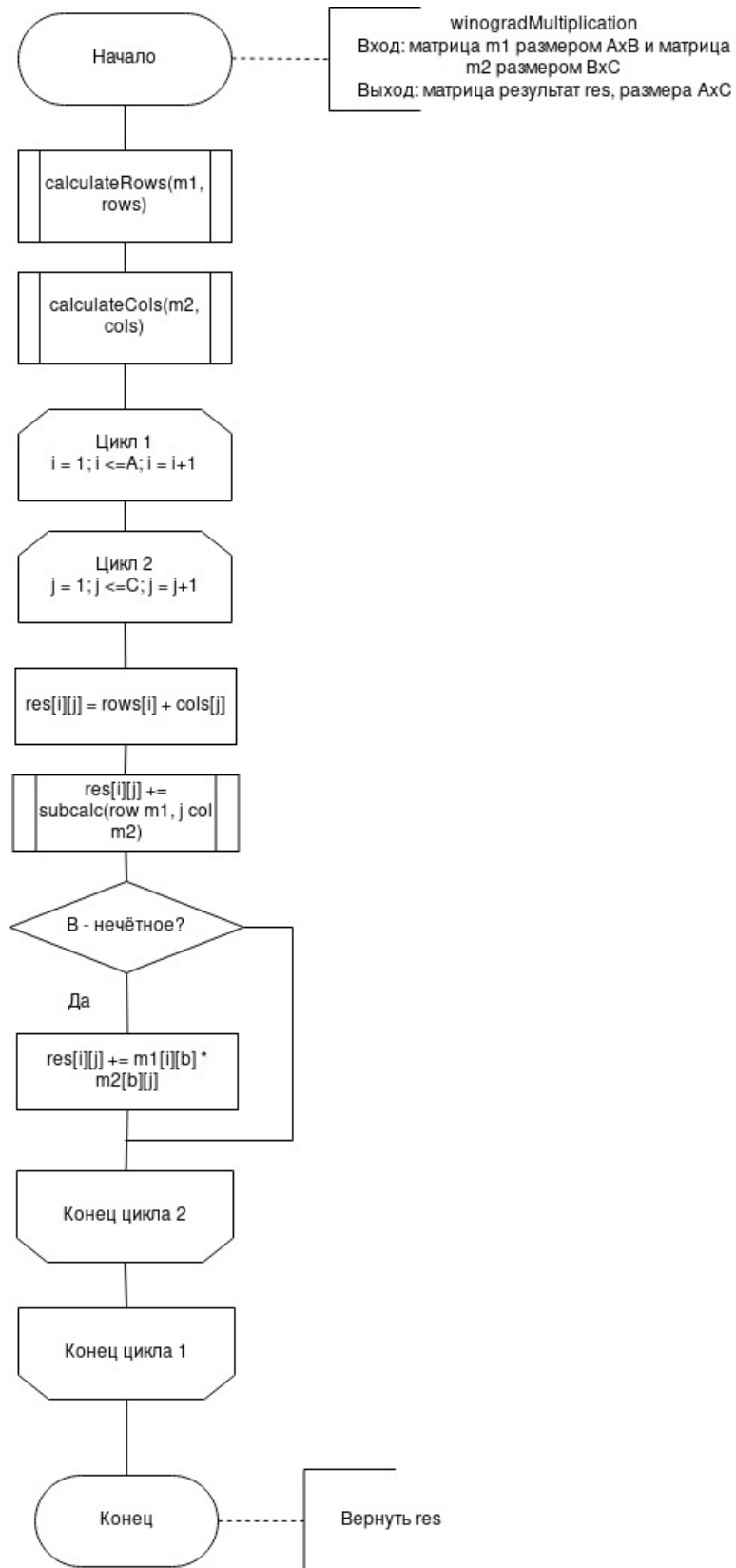


Рис. 2.2: Схема алгоритма Копперсмита – Винограда

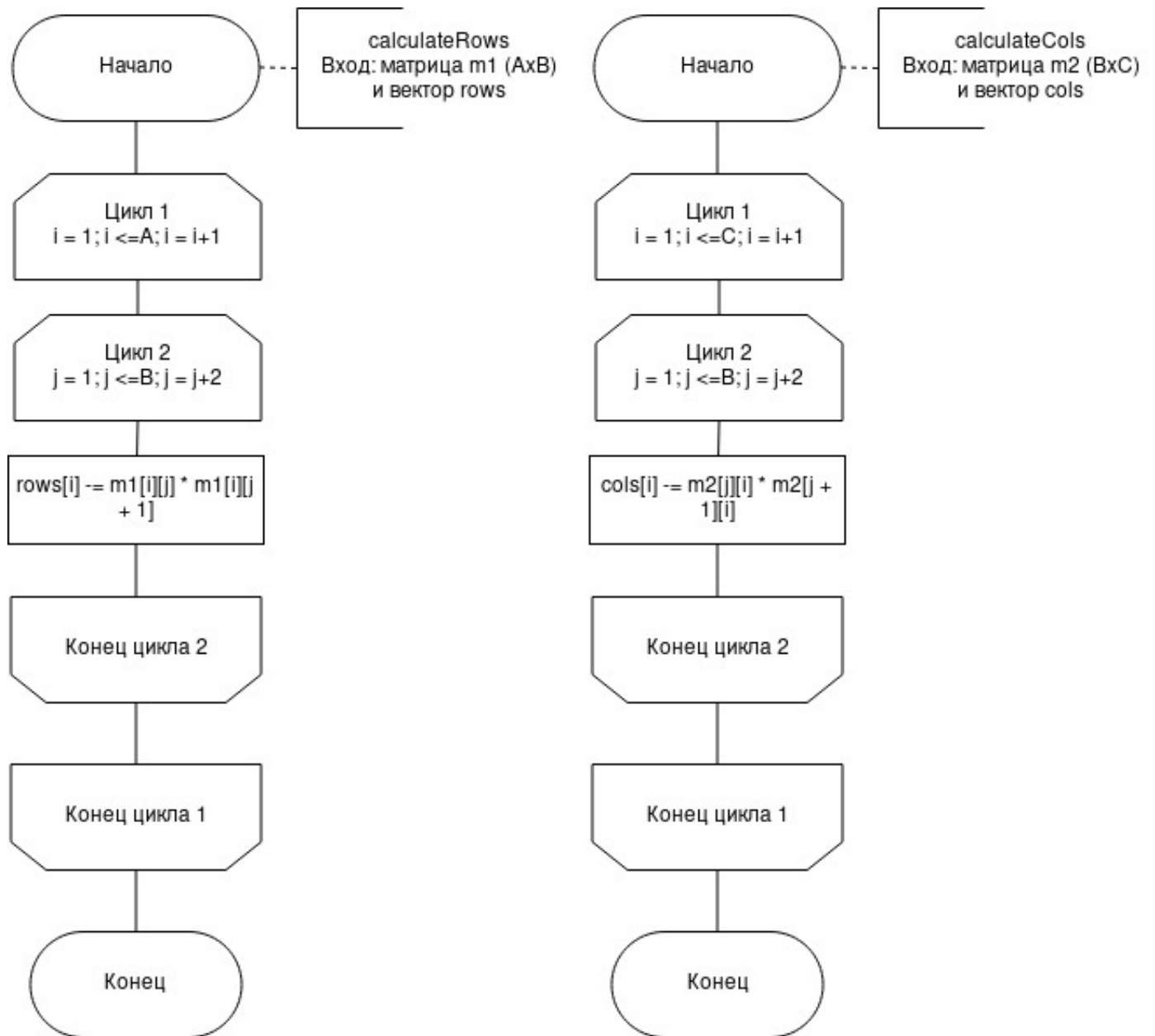


Рис. 2.3: Схема функций алгоритма Копперсмита – Винограда

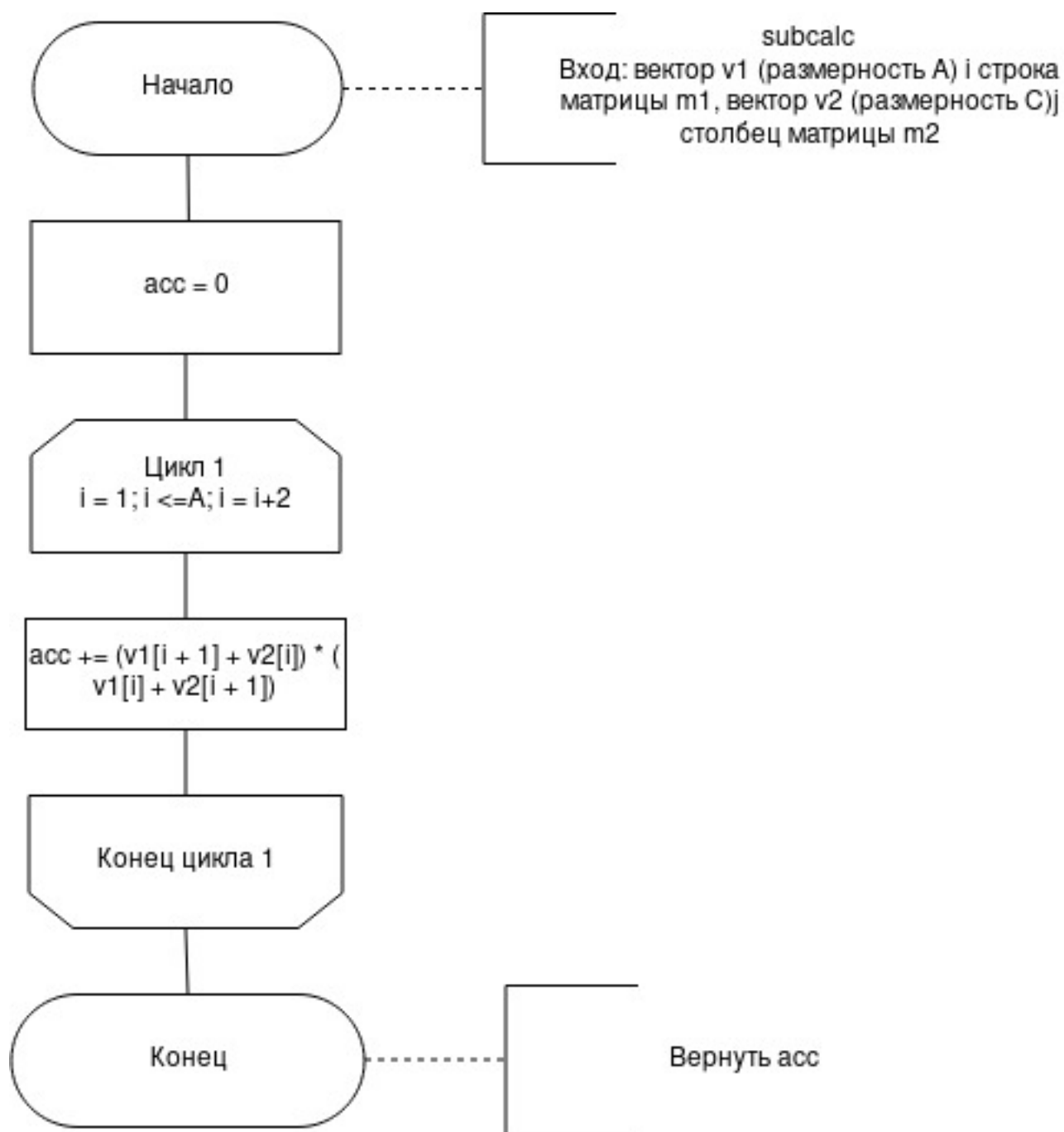


Рис. 2.4: Схема функций алгоритма Копперсмита – Винограда

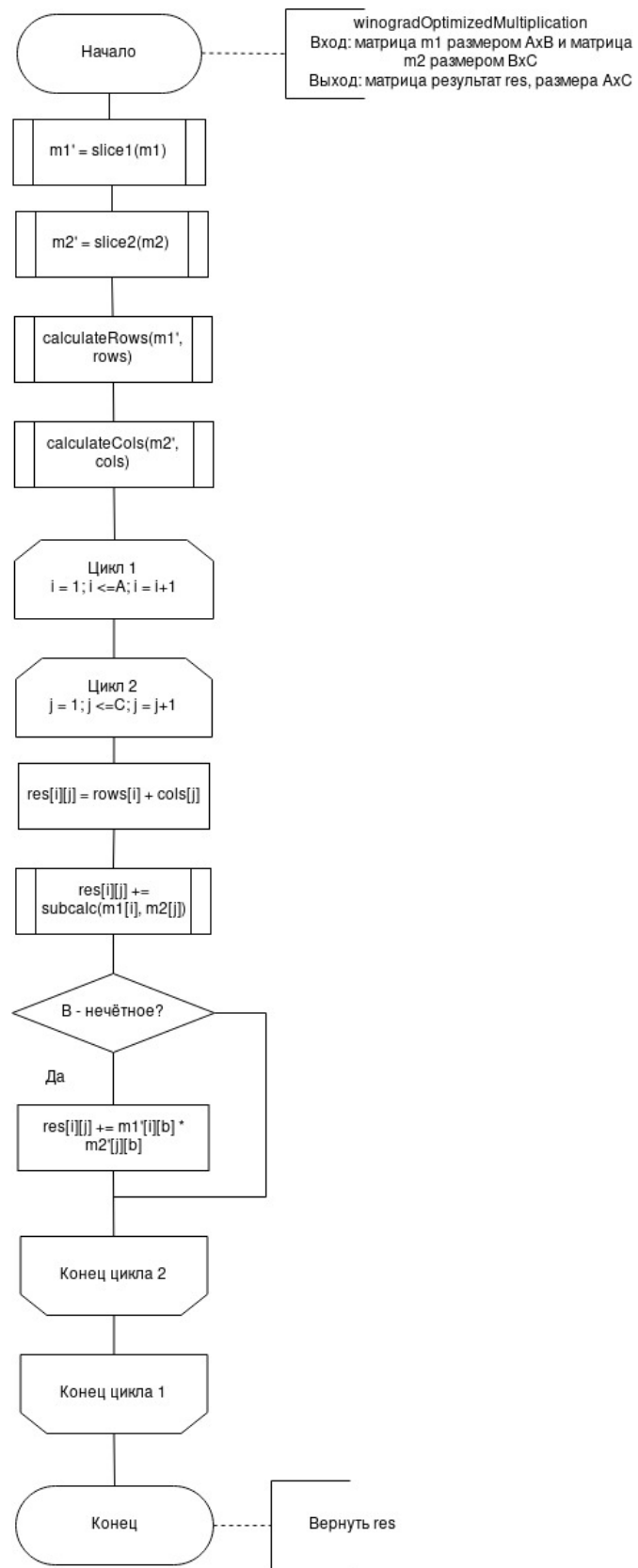


Рис. 2.5: Схема оптимизированного алгоритма Копперсмита – Винограда

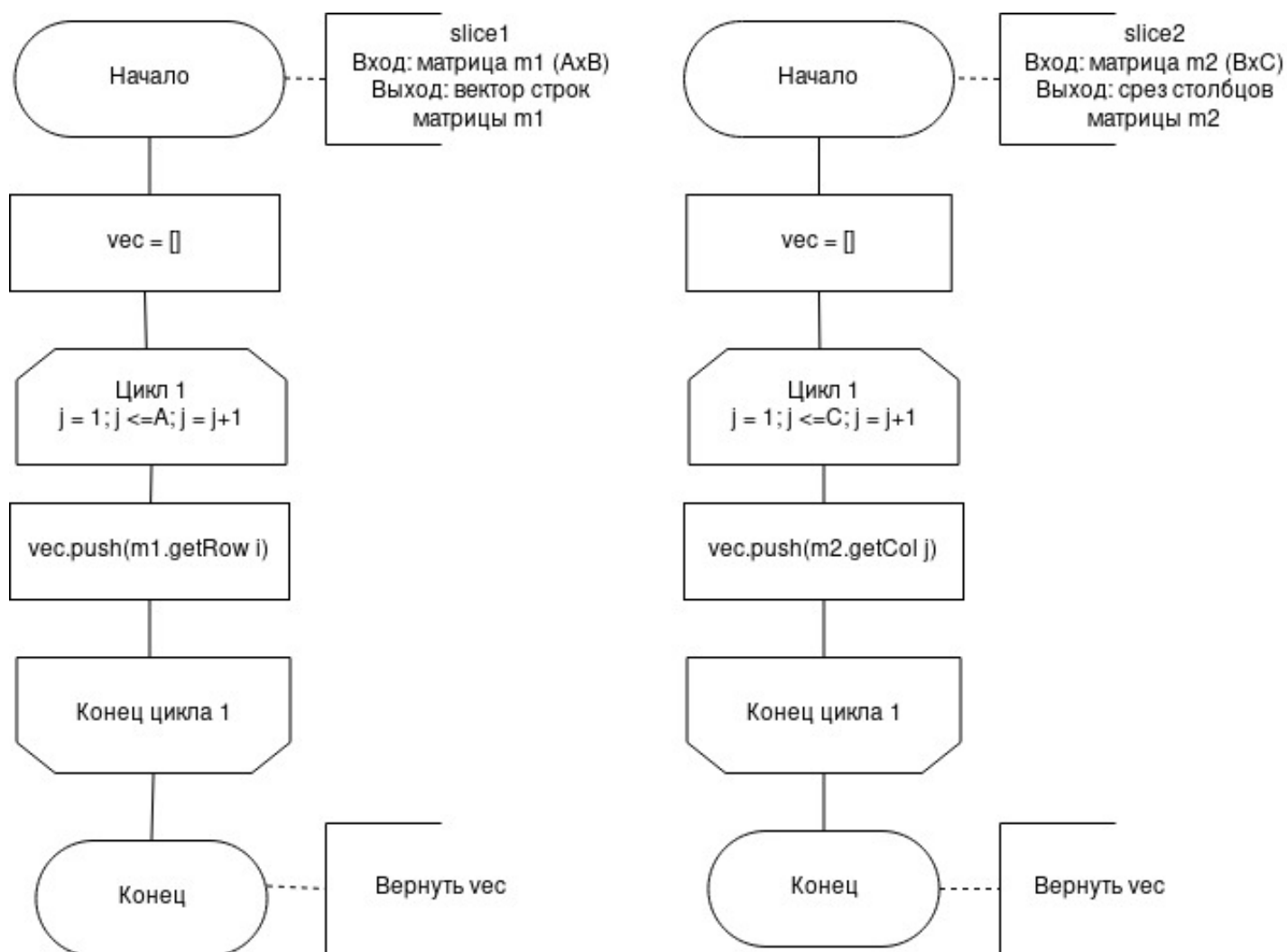


Рис. 2.6: Схема функций оптимизированного алгоритма Копперсмита – Винограда

3 | Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

- На вход ПО получает размеры 2 матриц, а также их элементы;
- На выходе — ПО печатает матрицу, которая является результатом умножения входных матриц.

3.2 Средства реализации

Для реализации ПО я выбрал язык программирования Haskell [2]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка программирования.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.4 приведена реализация алгоритмов перемножения матриц.

Листинг 3.1: Функция умножения матриц обычным способом

```
1 baseMultiplication :: (Num a) => Matrix a -> Matrix a -> Matrix a
2 baseMultiplication m1 m2 = M.fromList (M.nrows m1) (M.ncols m2) $ V.toList $
  _multp m1 m2 1
3 where
4   _multp m1 m2 i
5   | i == M.nrows m1 + 1 = V.fromList []
6   | otherwise = P.foldl (\acc j ->
7     V.zipWith (+) acc $ V.zipWith (*) (V.fromList $ P.take (M.ncols m2) $
8       repeat (M.getElem i j m1)) (M.getRow j m2))
9     (V.fromList $ P.take (M.ncols m2) $ repeat 0) [1..M.ncols m1] V.++
10    _multp m1 m2 (i + 1)
```

Листинг 3.2: Функция умножения матриц с транспонированием

```
1 baseTMultiplication :: (Num a) => Matrix a -> Matrix a -> Matrix a
```

```

1 baseTMultiplication m1 m2 = M.fromList (M.nrows m1) (M.ncols m2) $ V.toList
  $ _multp m1 (M.transpose m2) 1
2
3 where
4   _multp m1 m2 i
5   | i == M.nrows m1 + 1 = V.fromList []
6   | otherwise = P.foldl (\acc j ->
7     V.snoc acc $ V.sum (V.zipWith (*) (M.getRow i m1) (M.getRow j m2)))
8     (V.fromList []) [1..M.nrows m2]
9     V.++ _multp m1 m2 (i + 1)

```

Листинг 3.3: Функция умножения матриц по Винограду

```

1 winogradMultiplication :: (Num a) => Matrix a -> Matrix a -> Matrix a
2 winogradMultiplication m1 m2 = res
3 where
4   a = M.nrows m1
5   b = M.ncols m1
6   c = M.ncols m2
7
8   rows = V.generate a $ \i -> precalc $ M.getRow (i + 1) m1
9   cols = V.generate c $ \j -> precalc $ M.getCol (j + 1) m2
10
11   precalc v = P.foldl (\acc i ->
12     acc - V.unsafeIndex v i * V.unsafeIndex v (i + 1)) 0 [0, 2 .. V.length v
13       - 2]
14
15   res = M.matrix a c $ \(i, j) ->
16     V.unsafeIndex rows (i - 1) + V.unsafeIndex cols (j - 1)
17     + subcalc (M.getRow i m1) (M.getCol j m2)
18     + if odd b then M.unsafeGet i b m1 * M.unsafeGet b j m2 else 0
19
20   subcalc v1 v2 = P.foldl (\acc i ->
21     acc + (V.unsafeIndex v1 (i + 1) + V.unsafeIndex v2 (i))
22     * (V.unsafeIndex v1 (i) + V.unsafeIndex v2 (i + 1))) 0 [0, 2 .. V.
23       length v1 - 2]

```

Листинг 3.4: Функция нахождения расстояния Дамерау-Левенштейна матрично

```

1 winogradOptimizedMultiplication :: (Num a) => Matrix a -> Matrix a -> Matrix
  a
2 winogradOptimizedMultiplication m1 m2 = res
3 where
4   a = M.nrows m1
5   b = M.ncols m1
6   c = M.ncols m2
7
8   m1' = V.generate a $ \i -> M.getRow (i + 1) m1
9   m2' = V.generate c $ \j -> M.getCol (j + 1) m2
10
11   rows = V.generate a $ \i -> precalc $ V.unsafeIndex m1' i
12   cols = V.generate c $ \j -> precalc $ V.unsafeIndex m2' j

```



```

13
14   precalc v = P.foldl (\acc i ->
15     acc - V.unsafeIndex v i * V.unsafeIndex v (i + 1)) 0 [0, 2 .. b - 2]
16
17   res = if odd b
18     then M.matrix a c $ \(i, j) ->
19       let v1 = V.unsafeIndex m1' (i - 1)
20         v2 = V.unsafeIndex m2' (j - 1)
21       in V.unsafeIndex rows (i - 1) + V.unsafeIndex cols (j - 1) + subcalc
22         v1 v2 + V.last v1 * V.last v2
23     else M.matrix a c $ \(i, j) ->
24       let v1 = V.unsafeIndex m1' (i - 1)
25         v2 = V.unsafeIndex m2' (j - 1)
26       in V.unsafeIndex rows (i - 1) + V.unsafeIndex cols (j - 1) + subcalc
27         v1 v2
28
29   subcalc v1 v2 = P.foldl (\acc i ->
30     acc + (V.unsafeIndex v1 (i + 1) + V.unsafeIndex v2 (i))
31       * (V.unsafeIndex v1 (i) + V.unsafeIndex v2 (i + 1))) 0 [0, 2 .. b
32       - 2]

```

3.4 Тестовые данные

В таблице 3.1 приведены тесты для функций, реализующих стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда. Все тесты пройдены успешно.

Первая матрица	Вторая матрица	Ожидаемый результат
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 6 & 12 & 18 \\ 6 & 12 & 18 \\ 6 & 12 & 18 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 2 \\ 1 & 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 5 & 10 \\ 5 & 10 \end{pmatrix}$
(2)	(2)	(4)
$\begin{pmatrix} 1 & -2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 6 \\ 4 & 12 & 18 \\ 4 & 12 & 18 \end{pmatrix}$

Таблица 3.1: Тестирование функций

3.5 Вывод

В данном разделе были разработаны исходные коды четырёх алгоритмов перемножения матриц: обычный алгоритм, алгоритм с транспонированием, алгоритм Копперсмита — Винограда, оптимизированный алгоритм Копперсмита — Винограда.

4 | Исследовательская часть

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система: Debian [3] Linux [4] 11 «bullseye» 64-bit.
- Оперативная память: 12 GB.
- Процессор: Intel(R) Core(TM) i5-3550 CPU @ 3.30GHz [5].

4.2 Время выполнения алгоритмов

Время выполнения алгоритм замерялось с помощью применения технологии профайлинга [6]. Данный инструмент даёт детальное описание количества вызовов и количества времени CPU, занятого каждой функцией.

В таблицах 4.1 и 4.2 представлены замеры времени работы для каждого из алгоритмов на чётных размерах матриц. Здесь и далее: С — стандартный алгоритм, ТС — стандартный алгоритм с транспонированием, КВ — алгоритм Копперсмита – Винограда, ОКВ – оптимизированный алгоритм Копперсмита – Винограда.

Таблица 4.1: Таблица времени выполнения алгоритмов при чётных размерах (в секундах)

Размер матрицы	С	ТС	КВ	ОКВ
100	0.482	0.120	0.169	0.158
200	3.998	0.947	1.312	1.253
300	13.526	3.364	4.478	4.063
400	34.330	7.521	11.319	9.909
500	NaN	14.913	22.427	19.357
600	NaN	26.448	39.292	33.265

Таблица 4.2: Таблица времени выполнения алгоритмов при нечётных размерах (в наносекундах)

Размер матрицы	С	ТС	КВ	ОКВ
101	0.493	0.126	0.190	0.178
201	4.025	1.009	1.411	1.301
301	13.629	3.465	4.562	4.120
401	34.710	7.810	11.450	10.080
501	NaN	15.123	22.651	19.612
601	NaN	26.768	39.512	33.729

4.3 Вывод

Реализация умножения матриц с помощью алгоритма Копперсмита – Винограда в среднем выполняется в 3.4 раза быстрее, чем умножение обычным способом. Но, не смотря на это, алгоритм матриц с предварительным транспонированием матрицы выполняется быстрее чем любой рассматриваемый алгоритм. Такой алгоритм выполняется быстрее 3.6 раза чем обычное умножение, в 1.5 раза быстрее чем алгоритм Копперсмита – Винограда и в 1.3 раза быстрее чем его оптимизированная версия. Можно сделать вывод, что это происходит из-за кэширования соседних элементов перемножаемых строк. Кроме этого, для такого алгоритма не нужна дополнительная память, и, следовательно, нет нужды в дополнительном времени для ее выделения.

Заключение

В рамках данной лабораторной работы:

1. Были изучены и реализованы 3 алгоритма перемножения матриц: обычный, Копперсмита – Винограда, оптимизированный Копперсмита – Винограда;
2. Был произведён анализ трудоёмкости алгоритмов на основе теоретических расчётов и выбранной модели вычислений;
3. Был сделан сравнительный анализ алгоритмов на основе экспериментальных данных.

На основании анализа трудоёмкости алгоритмов в выбранной модели вычислений было показано, что улучшенный алгоритм Винограда имеет меньшую сложность, нежели простой алгоритм перемножения матриц. На основании замеров времени исполнения алгоритмов, был сделан вывод, что алгоритм Копперсмита – Винограда в среднем в 3.5 раза быстрее чем обычный алгоритм умножения матриц. Кроме этого, я решил добавить в сравнение алгоритм с предварительным транспонированием матрицы. Оказалось, что такая реализация быстрее алгоритма Копперсмита – Винограда в 1.5 раза и обгоняет классический алгоритм умножения в 3.6 раза.

Литература

- [1] Реализация алгоритма умножения матриц по Винограду на языке Haskell. Анисимов Н.С, Строганов Ю.В. [Электронный ресурс]. Режим доступа: <https://cyberleninka.ru/article/n/realizatsiya-algoritma-umnozheniya-matrits-po-vinogradu-na-yazyke-haskell/viewer>. Дата обращения: 01.10.2020.
- [2] The Haskell purely functional programming language [Электронный ресурс]. Режим доступа: <https://haskell.org/>. Дата обращения: 16.09.2020.
- [3] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: <https://www.debian.org/>. Дата обращения: 20.09.2020.
- [4] Linux – Getting Started [Электронный ресурс]. Режим доступа: <https://linux.org>. Дата обращения: 20.09.2020.
- [5] Процессор Intel® Core™ i5-3550 [Электронный ресурс]. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/65516/intel-core-i5-3550-processor-6m-cache-up-to-3-70-ghz.html>. Дата обращения: 20.09.2020.
- [6] Haskell profiling [Электронный ресурс]. Режим доступа: https://downloads.haskell.org/ghc/8.8.1/docs/html/user_guide/index.html. Дата обращения: 20.09.2020.