



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

Студент Романов С.К.

Группа ИУ7-55Б

Оценка (баллы) _____

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

Москва — 2022 г.

Оглавление

ВВЕДЕНИЕ	2
1 Аналитическая часть	4
1.1 Стандартный алгоритм	4
1.2 Алгоритм Копперсмита – Винограда	4
1.3 Оптимизированный алгоритм Копперсмита – Винограда	5
2 Конструкторская часть	6
2.1 Схемы алгоритмов	6
2.2 Модель вычислений	16
2.3 Трудоёмкость алгоритмов	16
2.3.1 Стандартный алгоритм умножения матриц	16
2.3.2 Алгоритм Копперсмита — Винограда	17
2.3.3 Оптимизированный алгоритм Копперсмита — Винограда	18
3 Технологическая часть	20
3.1 Требования к программному обеспечению	20
3.2 Средства реализации	20
3.3 Листинги кода	20
3.3.1 Классический алгоритм перемножения матриц	21
3.3.2 Алгоритм перемножения матриц Копперсмита—Винограда	22
3.3.3 Оптимизированный алгоритм перемножения матриц Копперсмита—Винограда	23
3.4 Тестовые данные	25
4 Исследовательская часть	26
4.1 Пример выполнения	26
4.2 Время выполнения алгоритмов	27
ЗАКЛЮЧЕНИЕ	30

ВВЕДЕНИЕ

Алгоритм Копперсмита—Винограда[1] — 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. На текущий момент сложность алгоритма составляет $O(n^{2,3728596})$.

Цель лабораторной работы:

- Изучение и исследование особенностей оптимизации сложных вычислений.

Задачи лабораторной работы:

1. Изучить и реализовать алгоритмы перемножения матриц.
 - Классический;
 - Копперсмита—Винограда;
 - Копперсмита—Винограда с оптимизациями согласно варианту;
2. Создать ПО, реализующее алгоритмы, указанные в варианте.
3. Провести анализ затрат работы программы по времени и по памяти, выявить влияющие на них характеристики.
4. Создать отчёт, содержащий:
 - актуальность исследования;
 - характеристики предложенной реализации (по времени и памяти);
 - краткие рекомендации об особенностях применения оптимизаций (важно помнить об улучшениях, которые использует компилятор).
 - результаты тестирования;
 - Выводы.

Для достижения поставленных целей и задач необходимо:

1. Изучить теоретические основы алгоритма Копперсмита—Винограда;

2. Реализовать выше обозначенный алгоритм.
3. Изучить и реализовать возможные пути оптимизации.
4. Провести экспериментальное исследование.

В ходе работы будут затронуты следующие темы:

1. Оптимизация вычислений.
2. Алгоритмы перемножения матриц.
3. Оценка реализаций алгоритмов.

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):

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \quad (1.4)$$

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 Оптимизированный алгоритм Копперсмита – Винограда

Алгоритм полностью повторяет логику обычного алгоритма. Оптимизация заключается в упрощении некоторых вычислений со стороны языка программирования, такие как:

- Предварительно получить строки столбцы соответствующих матриц;
- заменить операцию $x = x + k$; на $x += k$;
- заменить умножение на 2 на побитовый сдвиг;

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, 2.4 и 2.5 представлена схема алгоритма Копперсмита—Винограда.

На рисунках 2.6, 2.7, 2.8 и 2.9 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.

Согласно варианту, алгоритм можно оптимизировать следующим образом:

- Предварительно получить строки столбцы соответствующих матриц;
- заменить операцию $x = x + k$; на $x += k$;
- заменить умножение на 2 на побитовый сдвиг;

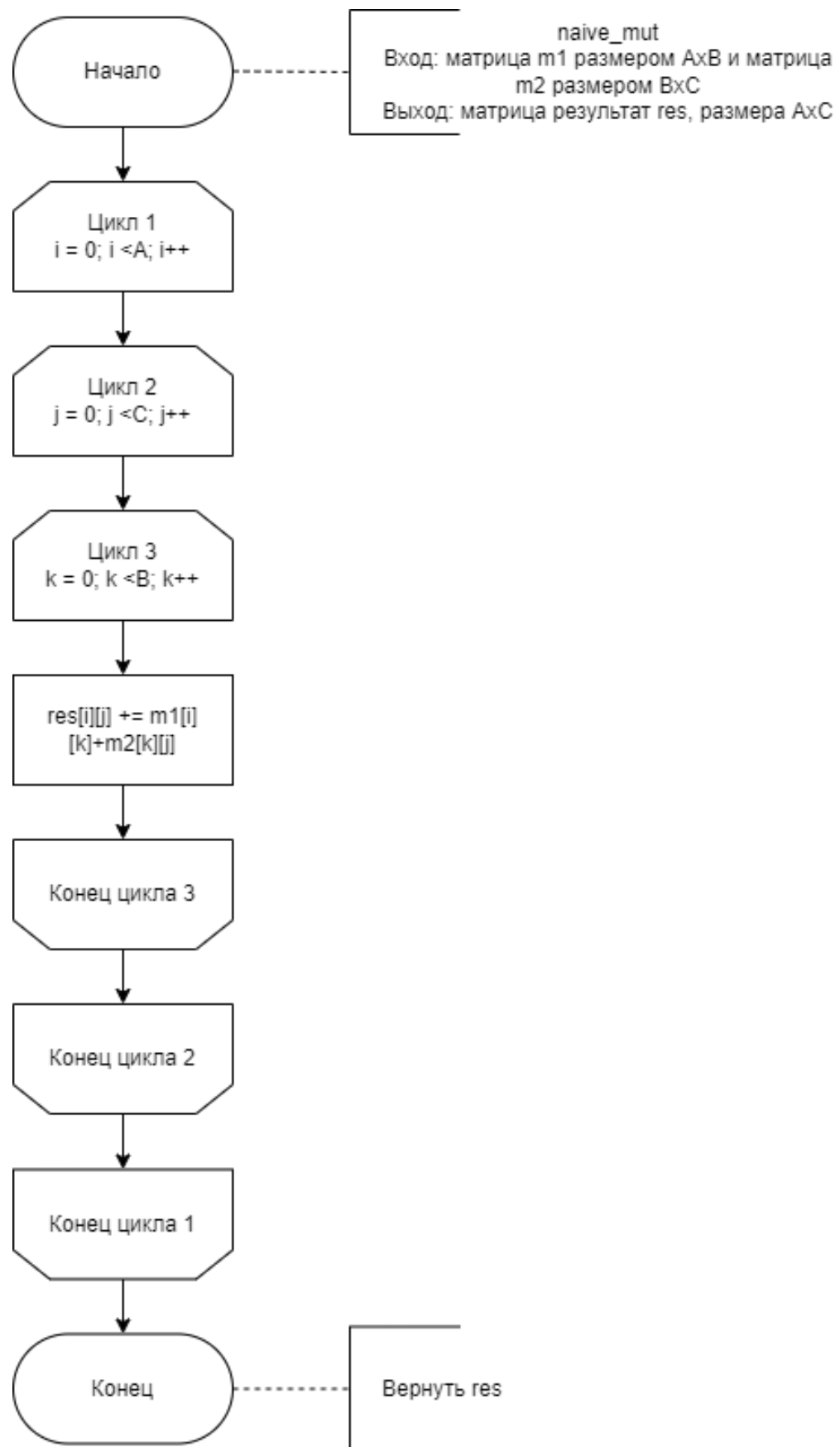


Рисунок 2.1 – Классическое перемножение матриц

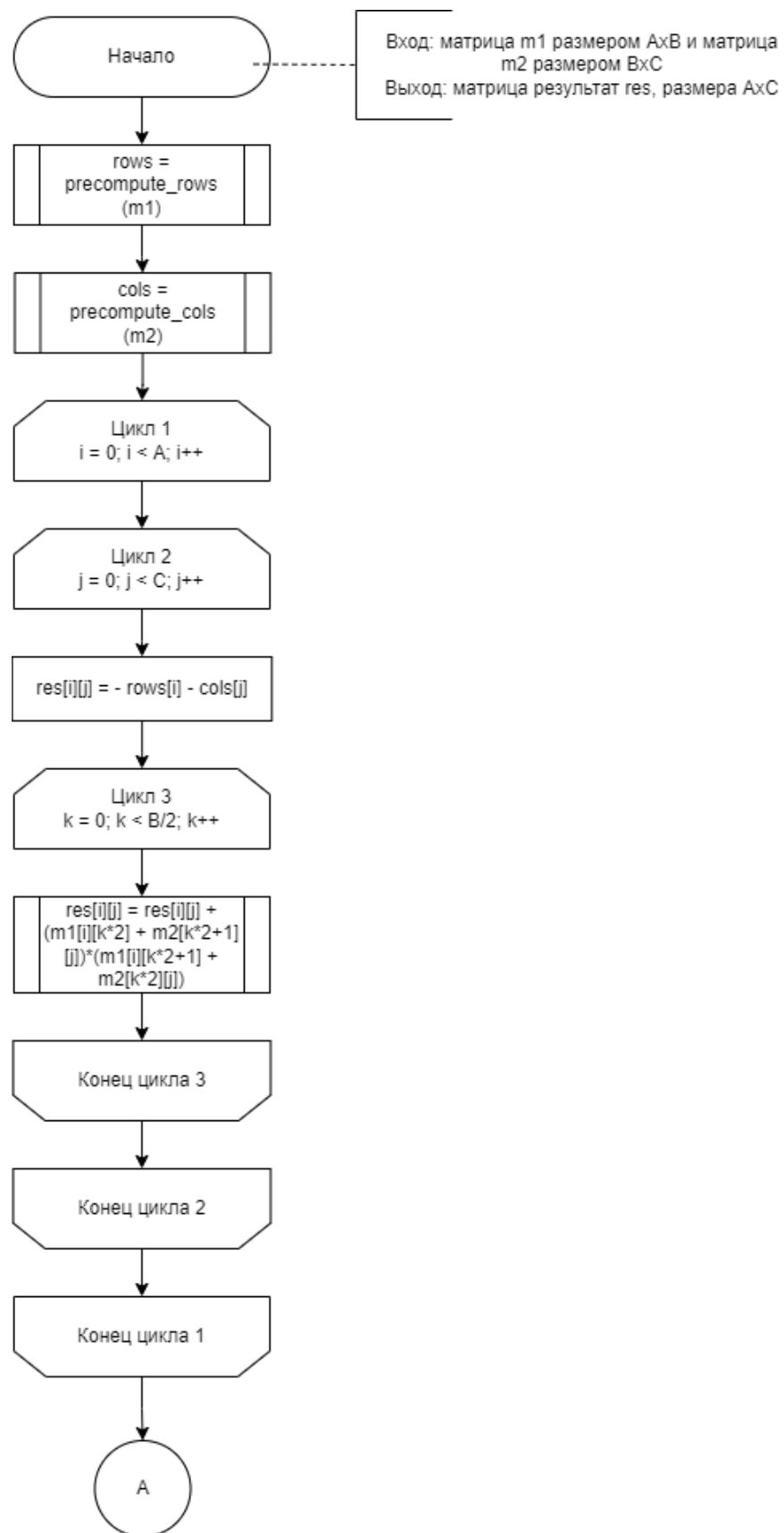


Рисунок 2.2 – Перемножение матриц с помощью алгоритма Копперсмита–Винограда, часть 1

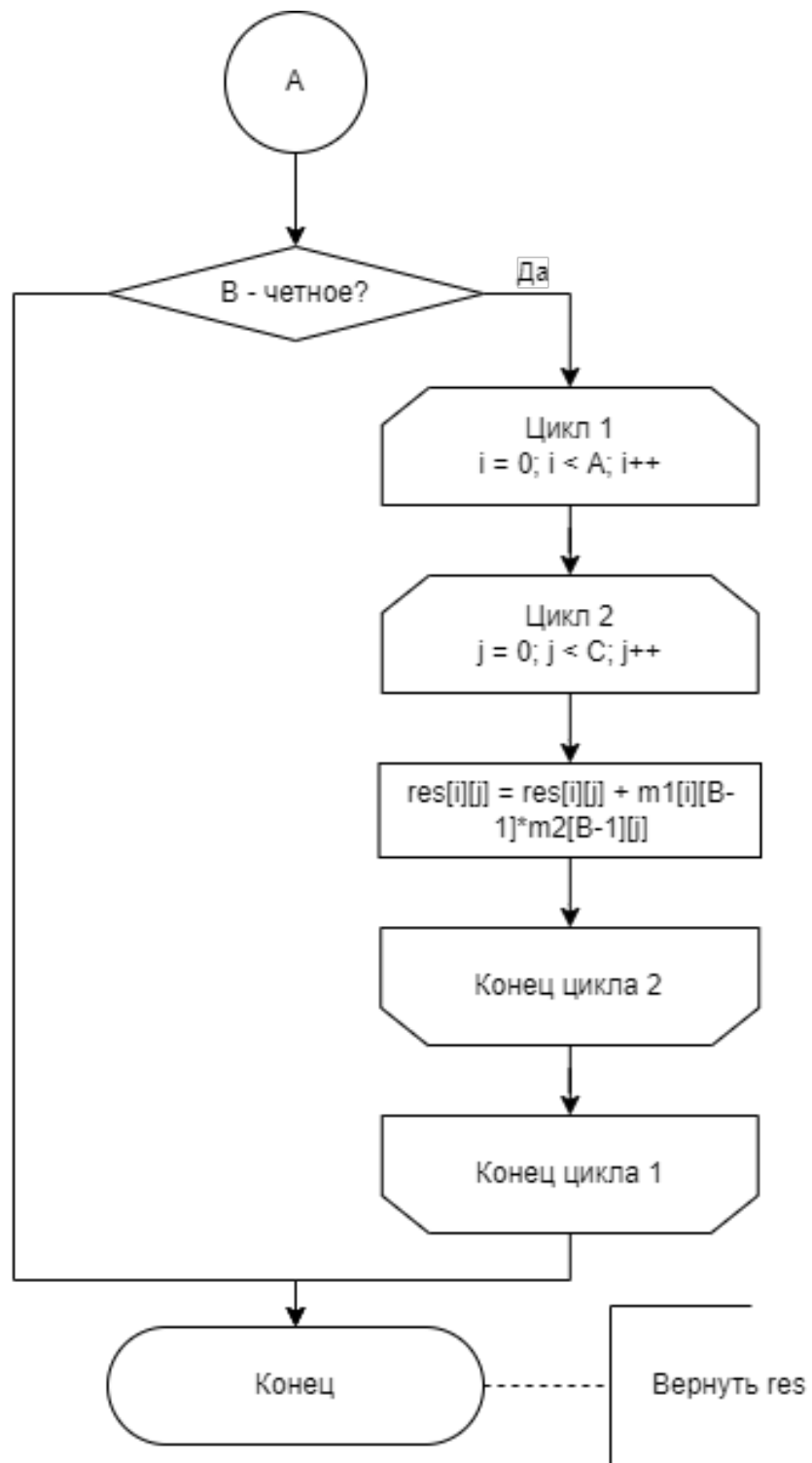


Рисунок 2.3 – Перемножение матриц с помощью алгоритма Копперсмита–Винограда, часть 2

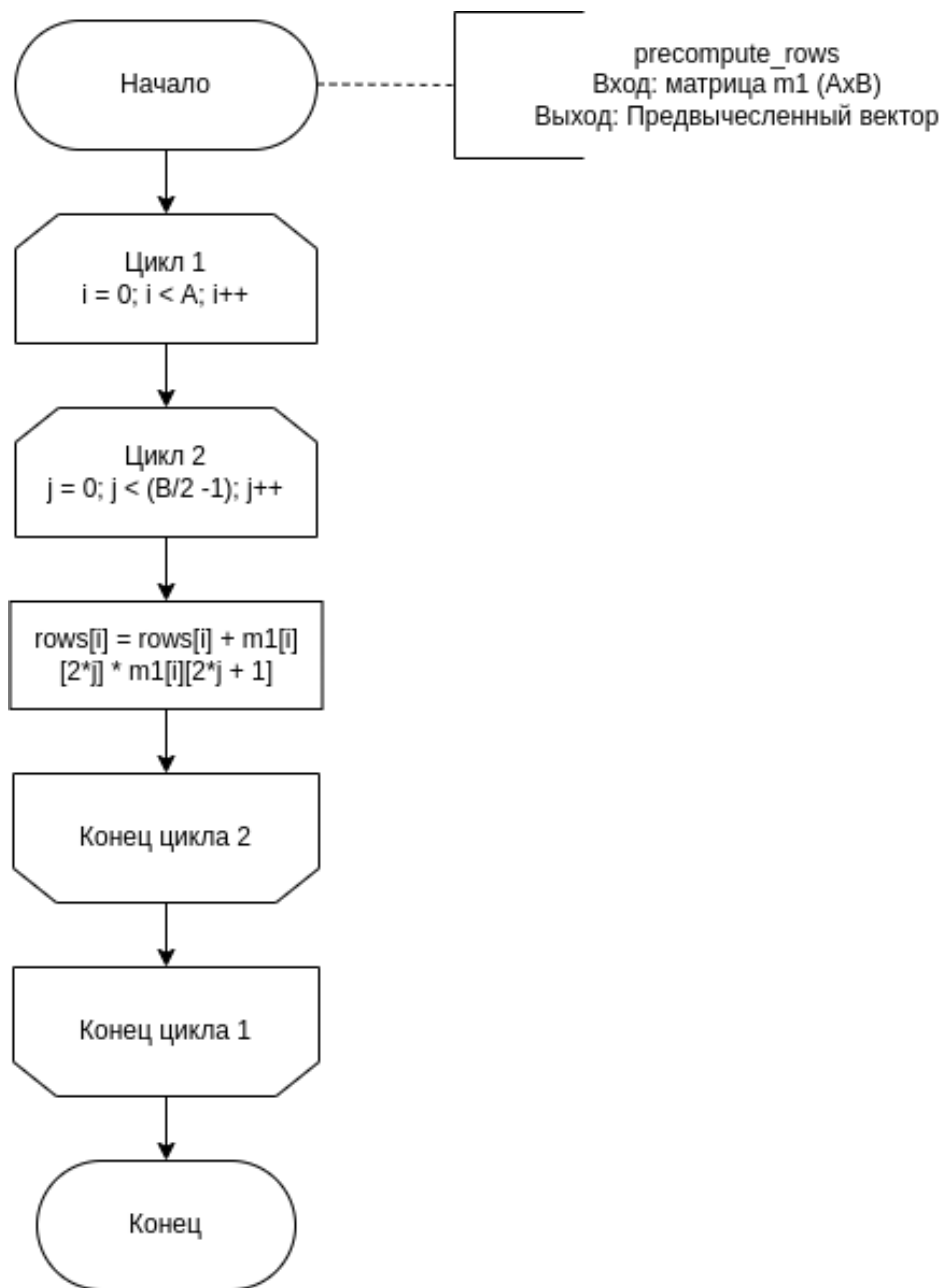


Рисунок 2.4 – Перемножение матриц с помощью алгоритма Копперсмита–Винограда, вычисление векторов, часть 1

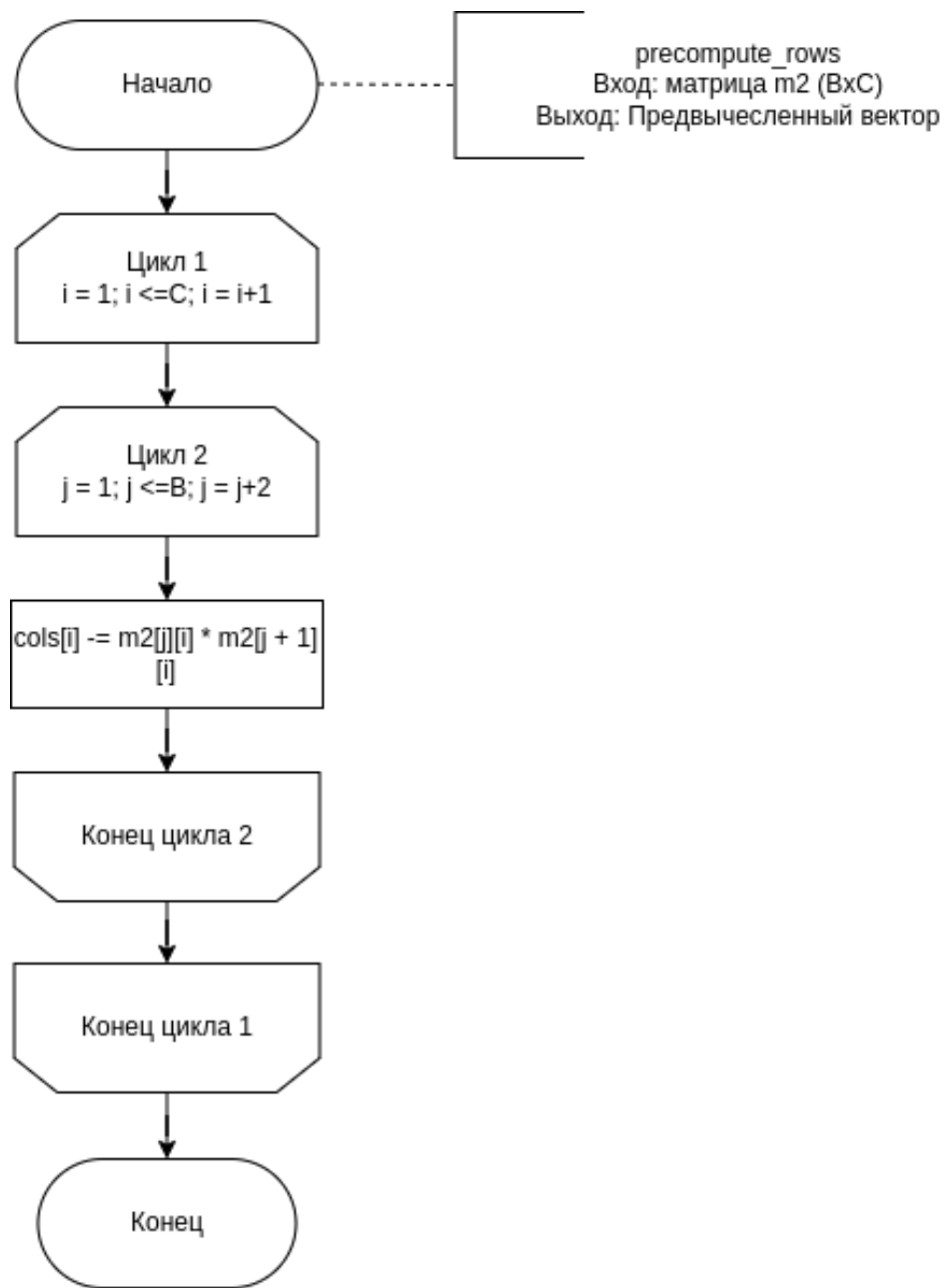


Рисунок 2.5 – Перемножение матриц с помощью алгоритма Копперсмита–Винограда, вычисление векторов, часть 2

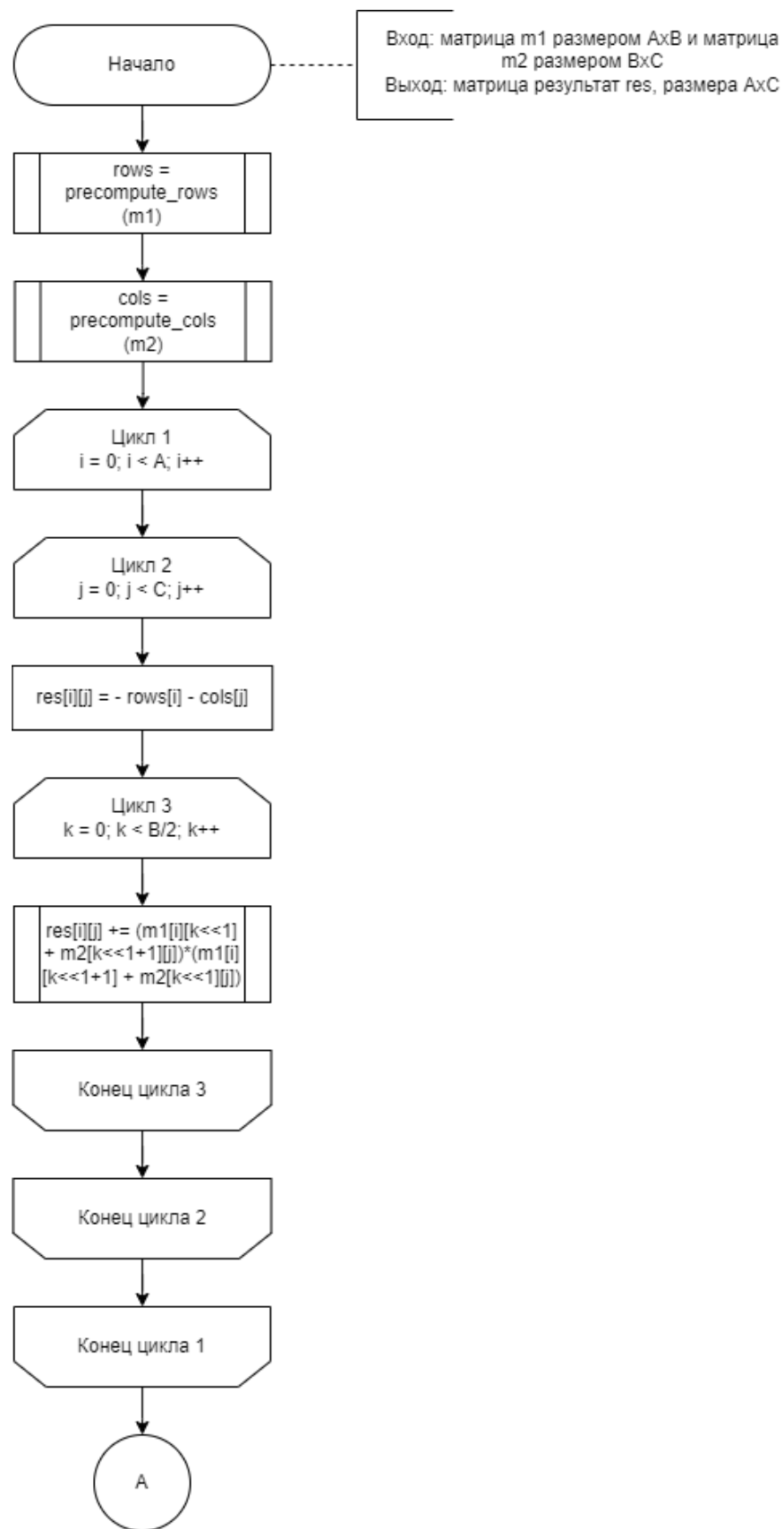


Рисунок 2.6 – Оптимизированное перемножение матриц с помощью алгоритма Копперсмита–Винограда, часть 1

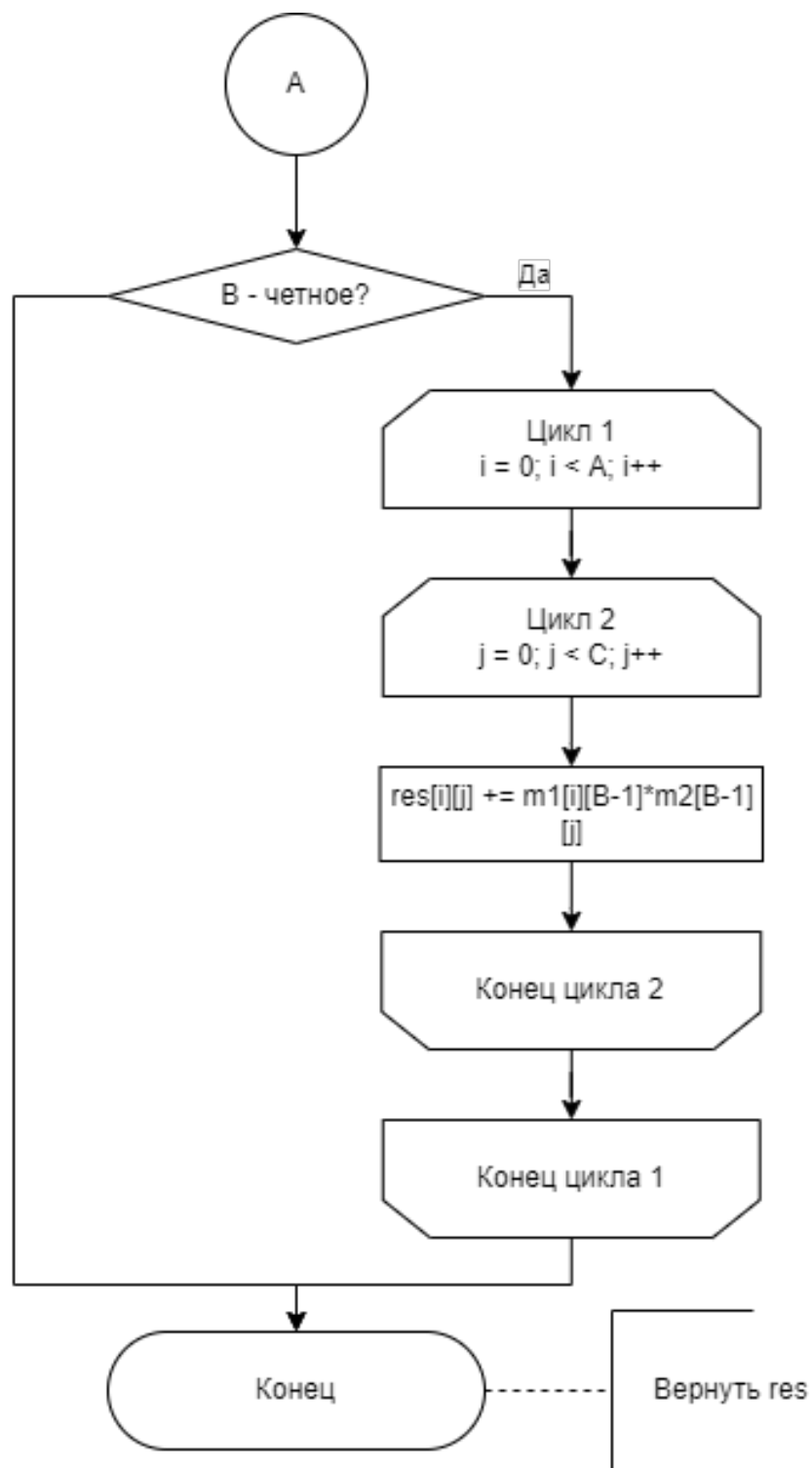


Рисунок 2.7 – Оптимизированное перемножение матриц с помощью алгоритма Копперсмита–Винограда, часть 2

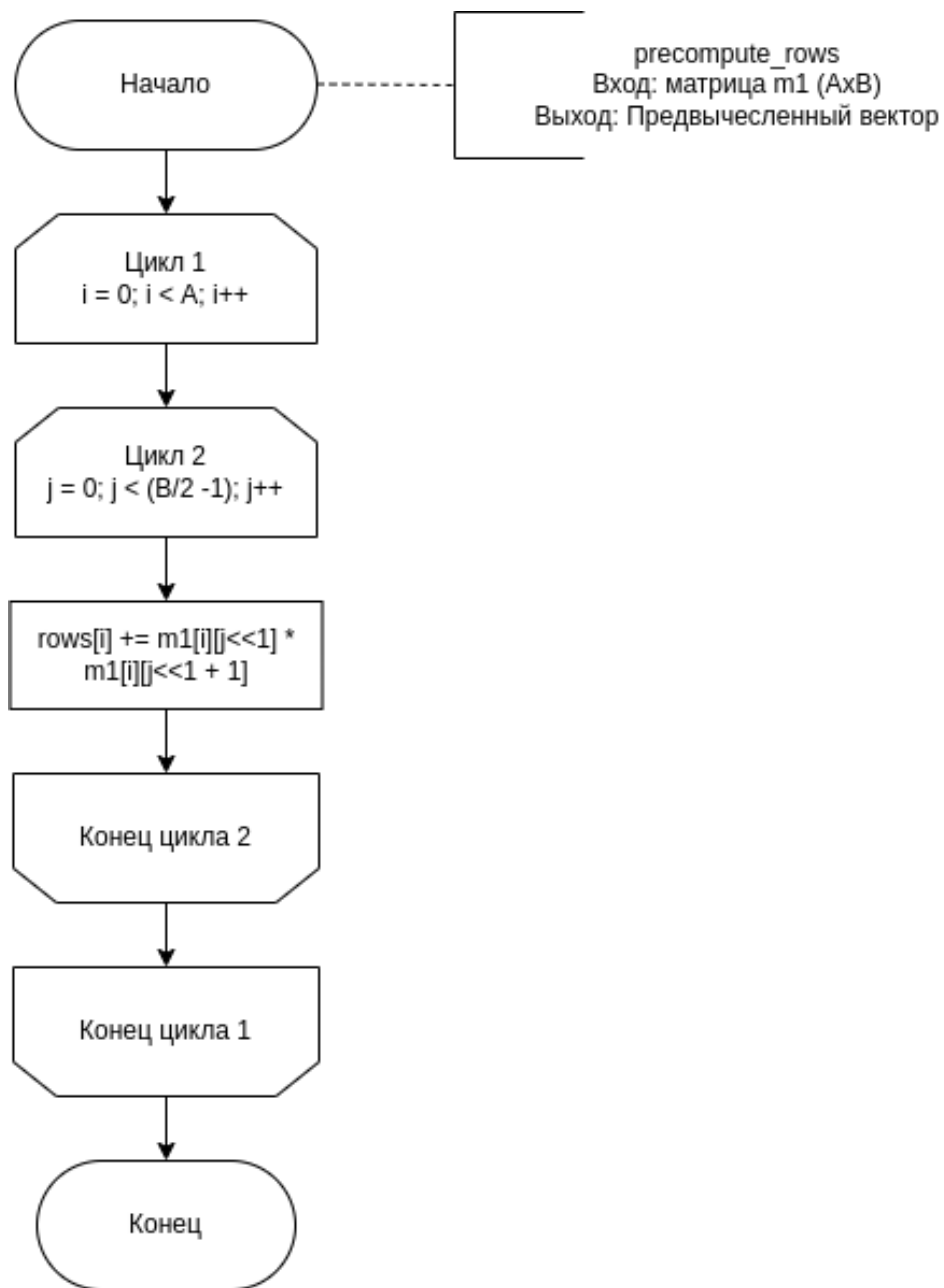


Рисунок 2.8 – Оптимизированное перемножение матриц с помощью алгоритма Копперсмита–Винограда, вычисление векторов, часть 1

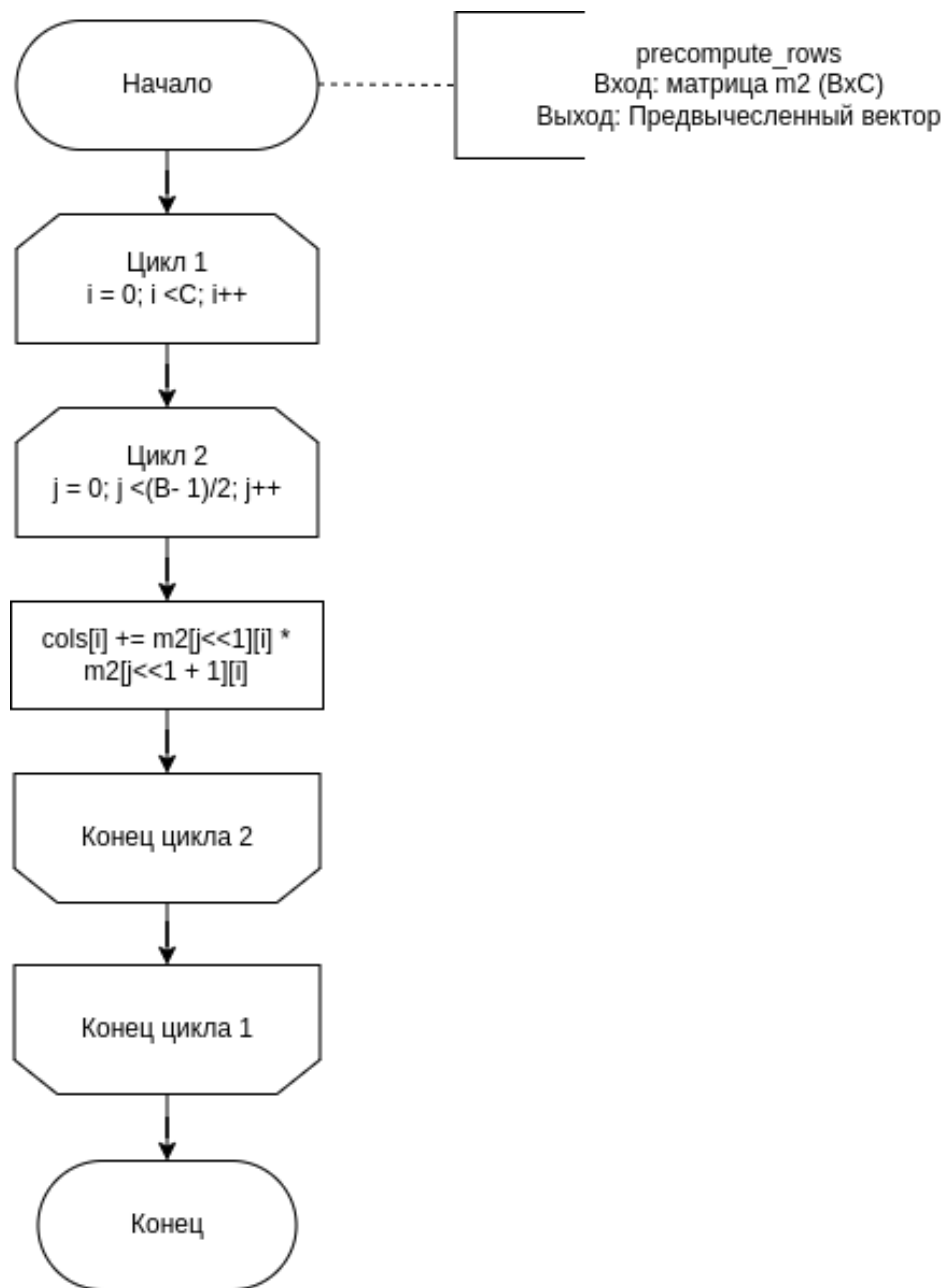


Рисунок 2.9 – Оптимизированное перемножение матриц с помощью алгоритма Копперсмита–Винограда, вычисление векторов, часть 2

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):

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

Вывод

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

3 Технологическая часть

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

3.1 Требования к программному обеспечению

К программе предъявляется ряд условий:

- На вход подается 3 числа (A, B, C), определяющие размеры матриц, а также сами матрицы размерами $A \times B$ и $B \times C$;
- На выход ПО должно выводить результат перемножения 2 матриц;

3.2 Средства реализации

Для реализации данной лабораторной работы необходимо установить следующее программное обеспечение:

- Rust Programming Language v1.64.0 - язык программирования
- Criterion.rs v0.4.0 - Средство визуализации данных
- LaTeX - система документооборота

3.3 Листинги кода

В следующих листингах представлены следующие алгоритмы

1. В листинге 3.1 представлен классический алгоритм перемножения матриц
2. В листингах 3.2 и 3.3 представлены алгоритм перемножения матриц с использованием алгоритма Копперсмита–Винограда
3. В листингах 3.4 и 3.5 представлены алгоритм перемножения матриц с использованием оптимизированного алгоритма Копперсмита–Винограда

3.3.1 Классический алгоритм перемножения матриц

Листинг 3.1 – Классический алгоритм перемножения матриц

```
1  fn naive_mut(&self, m2: &Vec<Vec<i32>>) -> Vec<Vec<i32>> {  
2      let mut result = vec![vec![0; m2[0].len()]; self.len()];  
3      for i in 0..self.len() {  
4          for j in 0..m2[0].len() {  
5              for k in 0..m2.len() {  
6                  result[i][j] += self[i][k] * m2[k][j];  
7              }  
8          }  
9      }
```

Замечание: здесь и далее под ключевым словом `self` понимается ссылка на экземпляр типа матрицы ($Vec < Vec < i32 > >$).

3.3.2 Алгоритм перемножения матриц Копперсмита–Винограда

Листинг 3.2 – Алгоритм перемножения матриц Копперсмита–Винограда

```
1  fn winograd_mut(&self, m2: &Vec<Vec<i32>>) -> Vec<Vec<i32>> {
2      let mut result = vec![vec![0; m2[0].len()]; self.len()];
3
4      let row_factor = self.precompute_rows();
5      let col_factor = m2.precompute_cols();
6
7      for i in 0..result.len()
8      {
9          for j in 0..result[0].len()
10         {
11             result[i][j] = -row_factor[i] - col_factor[j];
12             for k in 0..(m2.len() / 2)
13             {
14                 result[i][j] = result[i][j] + (self[i][k*2] + m2[k*2+1][j]) *
15                     (self[i][k*2+1] + m2[k*2][j]);
16             }
17         }
18     }
19
20     if m2.len()%2 == 1
21     {
22         for i in 0..result.len()
23         {
24             for j in 0..result[0].len()
25             {
26                 result[i][j] = result[i][j] + self[i][m2.len()-1] * m2[m2.len()-1][j]
27             }
28         }
29     }
30     result
31 }
```

Листинг 3.3 – Алгоритм перемножения матриц Копперсмита–Винограда, вычисление векторов

```
1  fn precompute_rows(&self) -> Vec<i32>
2  {
3      let mut row_factor = vec![0; self.len()];
4      for i in 0..self.len()
5      {
```



```

6         for j in 0..((self[0].len())/2)
7         {
8             row_factor[i] = row_factor[i] + self[i][j*2] * self[i][j*2 + 1];
9         }
10    }
11
12    row_factor
13 }
14
15 fn precompute_cols(&self) -> Vec<i32> {
16     let mut col_factor = vec![0; self[0].len()];
17     for i in 0..self[0].len()
18     {
19         for j in 0..((self.len())/2)
20         {
21             col_factor[i] = col_factor[i] + self[j*2][i] * self[j*2 + 1][i];
22         }
23     }
24
25     col_factor
26 }

```

3.3.3 Оптимизированный алгоритм перемножения матриц Копперсмита–Винограда

Листинг 3.4 – Оптимизированный алгоритм перемножения матриц Копперсмита–Винограда

```

1  fn winograd_mut_improved(&self, m2: &Vec<Vec<i32>>) -> Vec<Vec<i32>> {
2      let (a, b, c) = (self.len(), m2.len(), m2[0].len());
3      let mut result = vec![vec![0; c]; a];
4
5      let row_factor = self.precompute_rows_imp();
6      let col_factor = m2.precompute_cols_imp();
7
8      for i in 0..a
9      {
10         for j in 0..c
11         {
12             result[i][j] = -row_factor[i] - col_factor[j];
13             for k in 0..(m2.len() / 2)
14             {
15                 result[i][j] += (self[i][k<<1] + m2[(k<<1)+1][j]) * (self[i][(k<<1)+1] +
16                                     m2[k<<1][j]);
17             }
18         }
19     }
20
21     result
22 }

```

```

16         }
17     }
18 }
19
20 if m2.len()%2 == 1
21 {
22     for i in 0..a
23     {
24         for j in 0..c
25         {
26             result[i][j] += self[i][b-1] * m2[b-1][j]
27         }
28     }
29 }
30
31 result
32 }

```

Листинг 3.5 – Оптимизированный алгоритм перемножения матриц Копперсмита–Винограда, вычисление векторов

```

1  fn precompute_rows_imp(&self) -> Vec<i32> {
2      let mut row_factor = vec![0; self.len()];
3      for i in 0..self.len()
4      {
5          for j in 0..((self[0].len())/2)
6          {
7              row_factor[i] += self[i][j<<1] * self[i][(j<<1) + 1];
8          }
9      }
10
11     row_factor
12 }
13
14 fn precompute_cols_imp(&self) -> Vec<i32> {
15     let mut col_factor = vec![0; self[0].len()];
16     for i in 0..self[0].len()
17     {
18         for j in 0..((self.len())/2)
19         {
20             col_factor[i] += self[j<<1][i] * self[(j<<1) + 1][i];
21         }
22     }
23
24     col_factor
25 }

```

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.

Таблица 3.1 – Тестирование функций

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)
$\begin{pmatrix} 1 & -2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 6 \\ 4 & 12 & 18 \\ 4 & 12 & 18 \end{pmatrix}$

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

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

- Операционная система: Arch Linux [2] 64-bit.
- Оперативная память: 16 Гб.
- Процессор: 11th Gen Intel® Core™ i5-11320H @ 3.20 ГГц [3].

4.1 Пример выполнения

```
Введите a, b, c через пробел (матрицы - a x b, b x c): 3 2 3
Input matrix A : [3 x 2]
row 1: 1 2
row 2: 3 4
row 3: 5 6
Input matrixB : [2 x 3]
row 1: 1 2 3
row 2: 4 5 6
Multiplication results.
Naive multiplication:
9 12 15
19 26 33
29 40 51
Winograd multiplication:
9 12 15
19 26 33
29 40 51
Winograd multiplication improved:
9 12 15
19 26 33
29 40 51
```

Рисунок 4.1 – Пример выполнения программы

4.2 Время выполнения алгоритмов

Алгоритмы тестировались при помощи инструментов замера времени предоставляемых библиотекой Criterion.rs [4]. Пример функции по замеру времени приведен в листинге 4.1. Количество повторов регулируется тестирующей системой самостоятельно, однако ввиду трудоемкости вычислений, количество повторов было ограничено до 12.

Листинг 4.1 – Пример бенчмарка

```
1 fn matmut_even(c: &mut Criterion) {
2     let plot_config = PlotConfiguration::default().summary_scale(AxisScale::Linear);
3
4     let mut group = c.benchmark_group("Matrix multiplication (of even size)");
5     group.plot_config(plot_config);
6     group.sample_size(12);
7     for size in (50..1150).step_by(100) {
8         let (m1, m2) = (generate_matrix(size, size), generate_matrix(size, size));
9         // group.throughput(Throughput::Bytes(size_of_val(&*m1) as u64 + size_of_val(&*m2) as
10             u64));
11         group.bench_with_input(BenchmarkId::new("Naive", size), &size, |b, size| {
12             b.iter(|| black_box(m1.naive_mut(black_box(&m2))))
13         });
14         group.bench_with_input(BenchmarkId::new("Winograd", size), &size, |b, size| {
15             b.iter(|| black_box(m1.winograd_mut(black_box(&m2))))
16         });
17         group.bench_with_input(BenchmarkId::new("Winograd Imp.", size), &size, |b, size| {
18             b.iter(|| black_box(m1.winograd_mut_improved(black_box(&m2))))
19         });
20     }
21     group.finish();
22 }
23 fn matmut_odd(c: &mut Criterion) {
24     let plot_config = PlotConfiguration::default().summary_scale(AxisScale::Linear);
```

График, показывающий время перемножения матриц 3 методами (четный размер матриц)

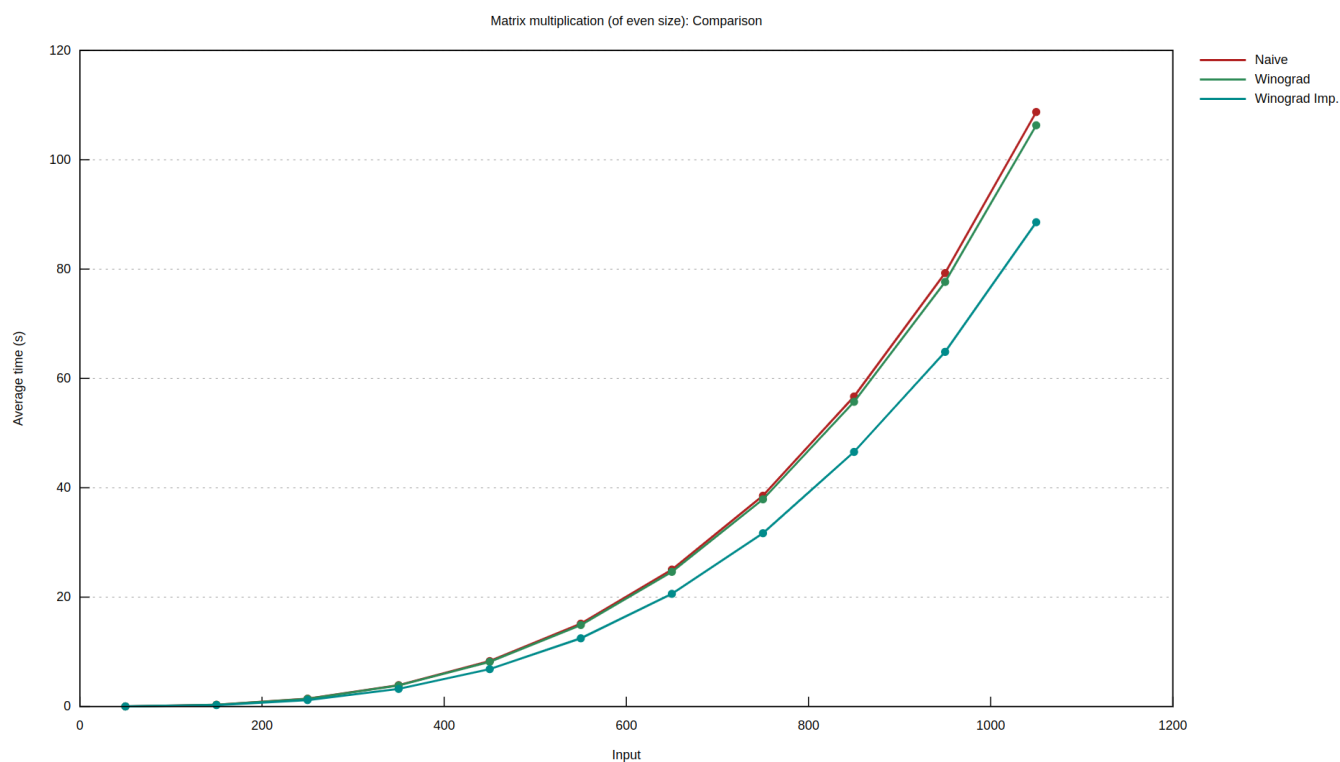


Рисунок 4.2 – График зависимости времени выполнения от размера выполнения, четный размер

График, показывающий время перемножения матриц 3 методами (нечетный размер матриц)

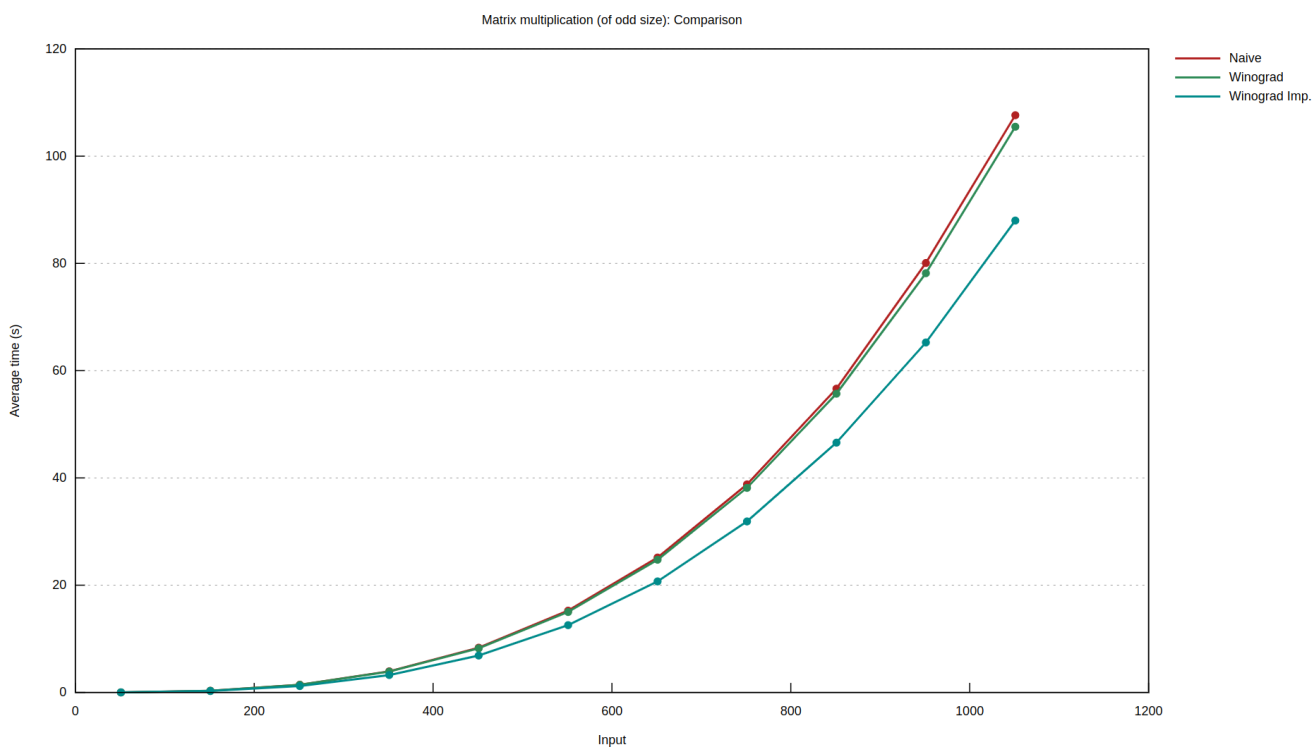


Рисунок 4.3 – График зависимости времени выполнения от размера выполнения, нечетный размер

Вывод

Время работы алгоритма Винограда незначительно меньше стандартного алгоритма умножения, однако при должной оптимизации есть возможность сильно лучше время выполнения.

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.
4. Был подготовлен отчёт по лабораторной работе, содержащий: актуальность исследования; характеристики предложенной реализации (по времени и памяти); краткие рекомендации об особенностях применения оптимизаций; результаты тестирования;

Время работы алгоритма Винограда незначительно меньше стандартного алгоритма умножения, однако при должной оптимизации есть возможность сильно лучше время выполнения.

Цели лабораторной работы по изучению и исследованию особенностей оптимизации сложных вычислений были достигнуты

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Copeersmith D., Winograd S.* Matrix Multiplication via Arithmetic Progressions. — 1990. — Дата обращения: 19.10.2022. Режим доступа: <http://www.cs.umd.edu/~gasarch/TOPICS/ramsey/matrixmult.pdf>.
2. Arch Linux [Электронный ресурс]. — Дата обращения: 19.10.2022. Режим доступа: <https://archlinux.org/>.
3. Процессор Intel® Core™ i5-11320H [Электронный ресурс]. — Дата обращения: 19.10.2022. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/217183/intel-core-i511320h-processor-8m-cache-up-to-4-50-ghz-with-ipu.html>.
4. Criterion [Электронный ресурс]. — Дата обращения: 19.10.2022. Режим доступа: <https://github.com/bheisler/criterion.rs>.