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 дисциплине «Анализ Алгоритмов»

Тема Алгоритмы умножения матриц

Студент Козырнов А.Д.

Группа ИУ7-52Б

Преподаватель Волкова Л. Л., Строганов Д.В.

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **3**

1 Аналитическая часть **4**

1.1 Описание алгоритмов 4

1.1.1 Классический алгоритм умножения матриц 4

1.1.2 Алгоритм Винограда умножения матриц 5

2 Конструкторская часть **6**

2.1 Представление алгоритмов 6

2.1.1 Трудоемкость алгоритмов 11

2.1.2 Модель вычислений 11

2.1.3 Классический алгоритм 11

2.1.4 Алгоритм Винограда 12

2.1.5 Оптимизированный алгоритм Винограда 13

3 Технологическая часть **15**

3.1 Требования к программному обеспечению 15

3.2 Средства реализации 15

3.3 Реализация алгоритмов 15

4 Исследовательская часть **21**

4.1 Технические характеристики 21

4.2 Время выполнения алгоритмов 21

4.3 Вывод 22

ЗАКЛЮЧЕНИЕ **24**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **25**

ВВЕДЕНИЕ

Матрицы представляют собой таблицы чисел, взаимосвязанных между собой [1].

Цель лабораторной работы — исследование алгоритмов умножения матриц следующими методами:

— классическим алгоритмом;

— алгоритмом Винограда;

— оптимизированного алгоритма Винограда.

Для достижения поставленной цели необходимо выполнить следующие задачи:

— реализовать указанные алгоритмы;

— сравнение требуемого времени выполнения алгоритмов в тиках процессора;

— описать и обосновать полученные результаты.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы умножения матриц.

1.1 Описание алгоритмов

Пусть даны матрицы A с размерами $N \times M$ и B с размерами $M \times K$. В результате умножения матрицы A на матрицу B получается матрица C с размером $N \times K$.

1.1.1 Классический алгоритм умножения матриц

Пусть даны матрицы A размерностью $n \times m$ и матрица B размерностью $m \times k$:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \dots & \dots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mk} \end{pmatrix}. \tag{1.1}$$

Тогда умножением матрицы A на матрицу B называется, где матрица C:

$$C = A \times B = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \dots & \dots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nk} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj} \quad (i = \overline{1,n},\ j = \overline{1,k}). \tag{1.3}$$

1.1.2 Алгоритм Винограда умножения матриц

Пусть даны матрицы A и B, имеющие размерность 4×4.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix}. \tag{1.4}$$

Для получение очередного элемента c_{ij} матрицы C в классическом алгоритме умножения матрицы выполняется по формуле:

$$c_{ij} = \begin{pmatrix} a_{n1} & a_{n2} & a_{n3} & a_{n4} \end{pmatrix} \times \begin{pmatrix} b_{j1} \\ b_{j2} \\ b_{j3} \\ b_{j4} \end{pmatrix}, \tag{1.5}$$

где $a_{ni},\ i = \overline{1,4}$ - элементы n-ой строки матрицы A; $b_{jk},\ k = \overline{1,4}$ - элементы j-ого столбца матрицы B.

В алгоритме Винограда для ускорения рассчетов снижается доля дорогих операций (умножения) и заменой их на сложение. Для достижения этой цели выполняется предварительная обработка. Запоминаются значения, что позволит заменить некоторые умножения сложением. Таким образом:

$$c_{ij} = (a_{n1}+b_{j2})(a_{n2}+b_{j1})+(a_{n3}+b_{j4})(a_{n4}+b_{j3})-a_{n1}a_{n2}-a_{n3}a_{n4}-b_{j1}b_{j2}-b_{j3}b_{j4}, \tag{1.6}$$

где элементы $a_{n1}a_{n2},\ a_{n3}a_{n4},\ b_{j1}b_{j2},\ b_{j3}b_{j4}$ - значения, которые получаются в предварительной обработке.

Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц. Основные различия между алгоритмами - наличие предварительной обработки и количество операций умножения.

2 Конструкторская часть

В данном резделе будут представлены схемы алгоритмов умножения матриц, приведены оценки трудоемкости алгоритмов.

2.1 Представление алгоритмов

На рисунках 2.2 — 2.3 представлен классический алгоритм умножения матриц. На рисунках 2.2 — 2.5 представлены два алгоритма умножения матриц Винограда — обычный и оптимизированный.

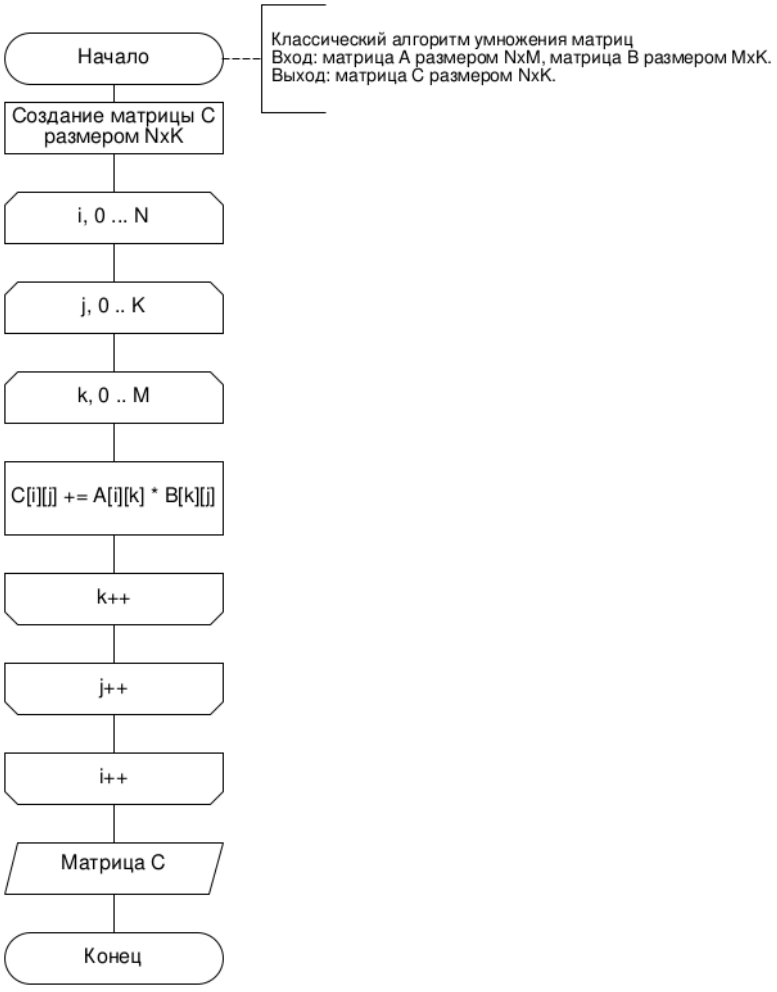

Рисунок 2.1 – Классический алгоритм умножения матриц

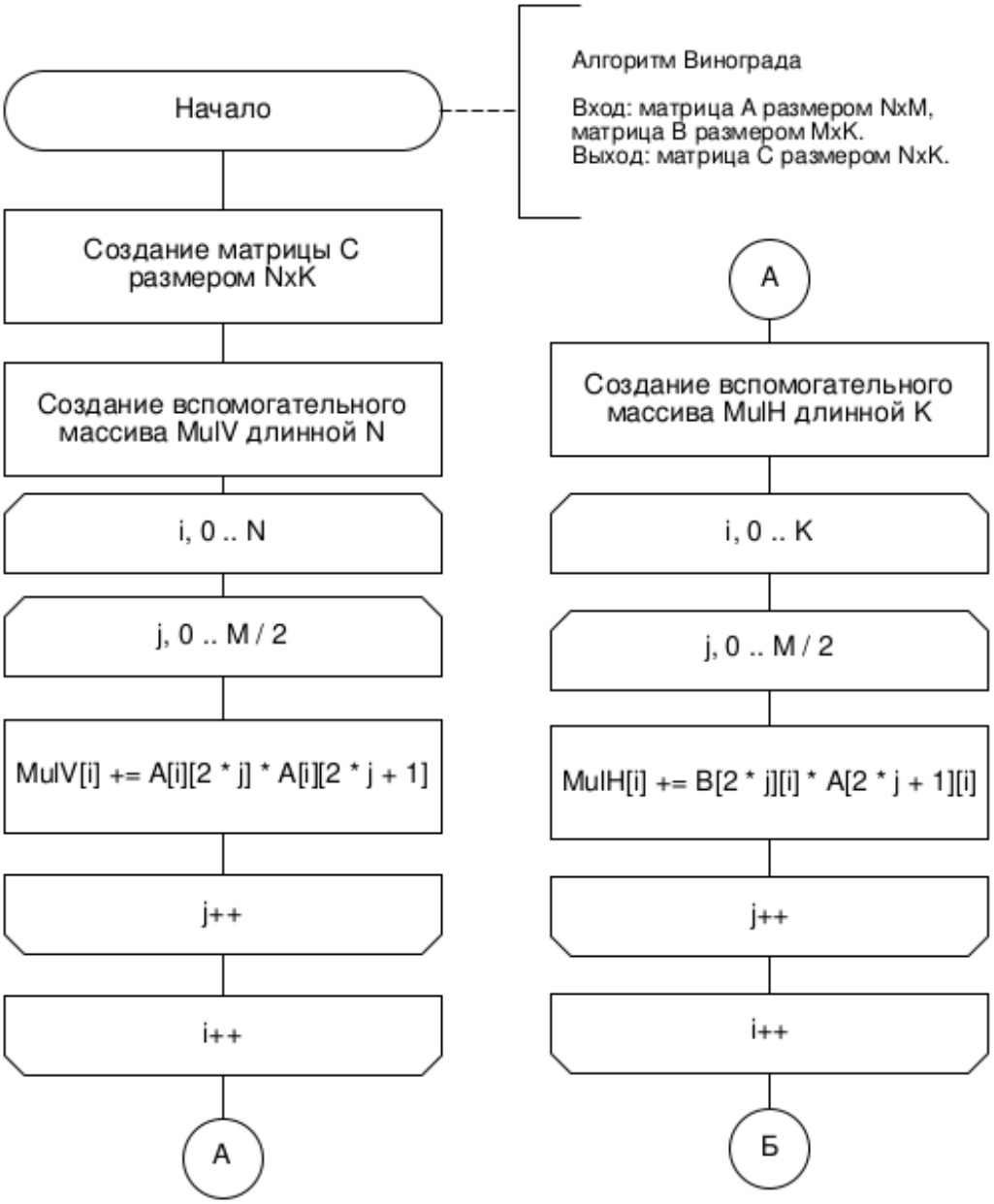

Рисунок 2.2 – Алгоритм Винограда. Часть 1

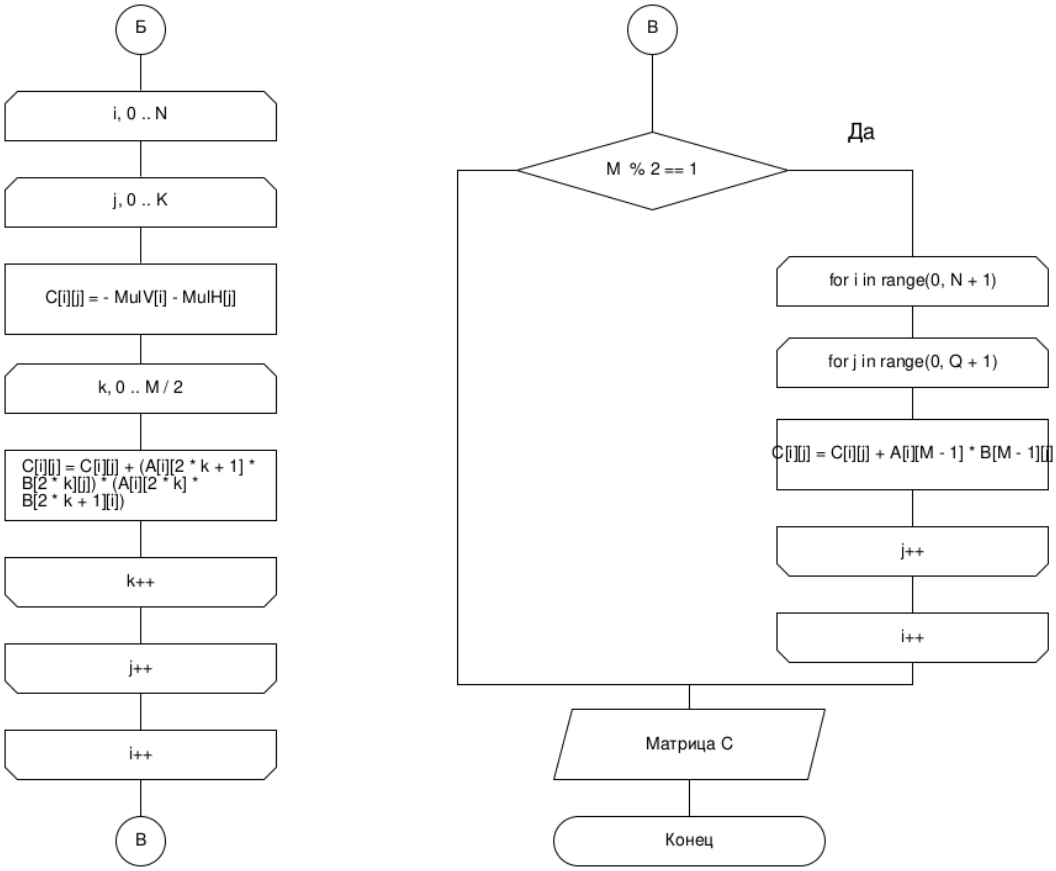

Рисунок 2.3 – Алгоритм Винограда. Часть 2

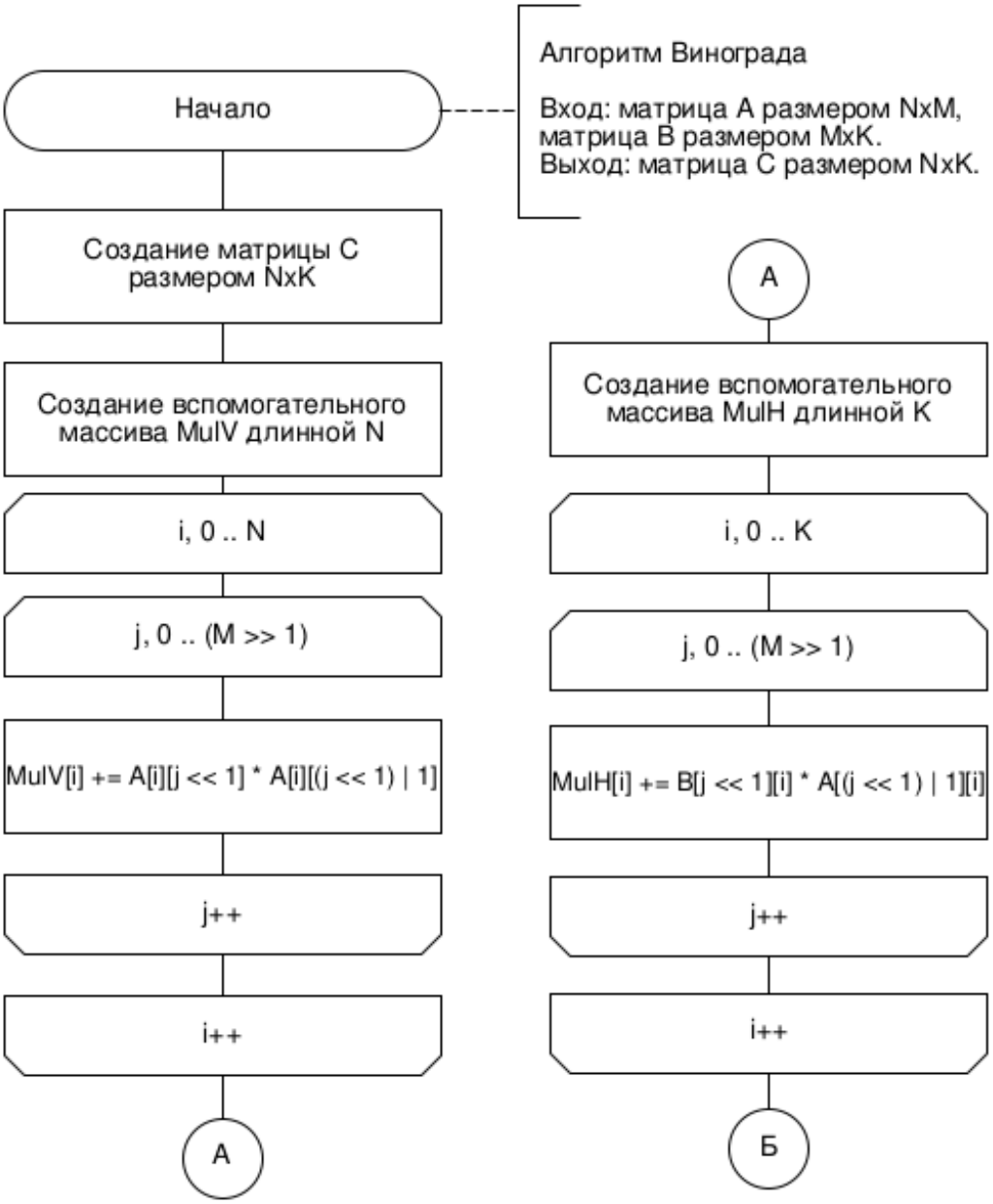

Рисунок 2.4 – Оптимизированный алгоритм Винограда. Часть 1

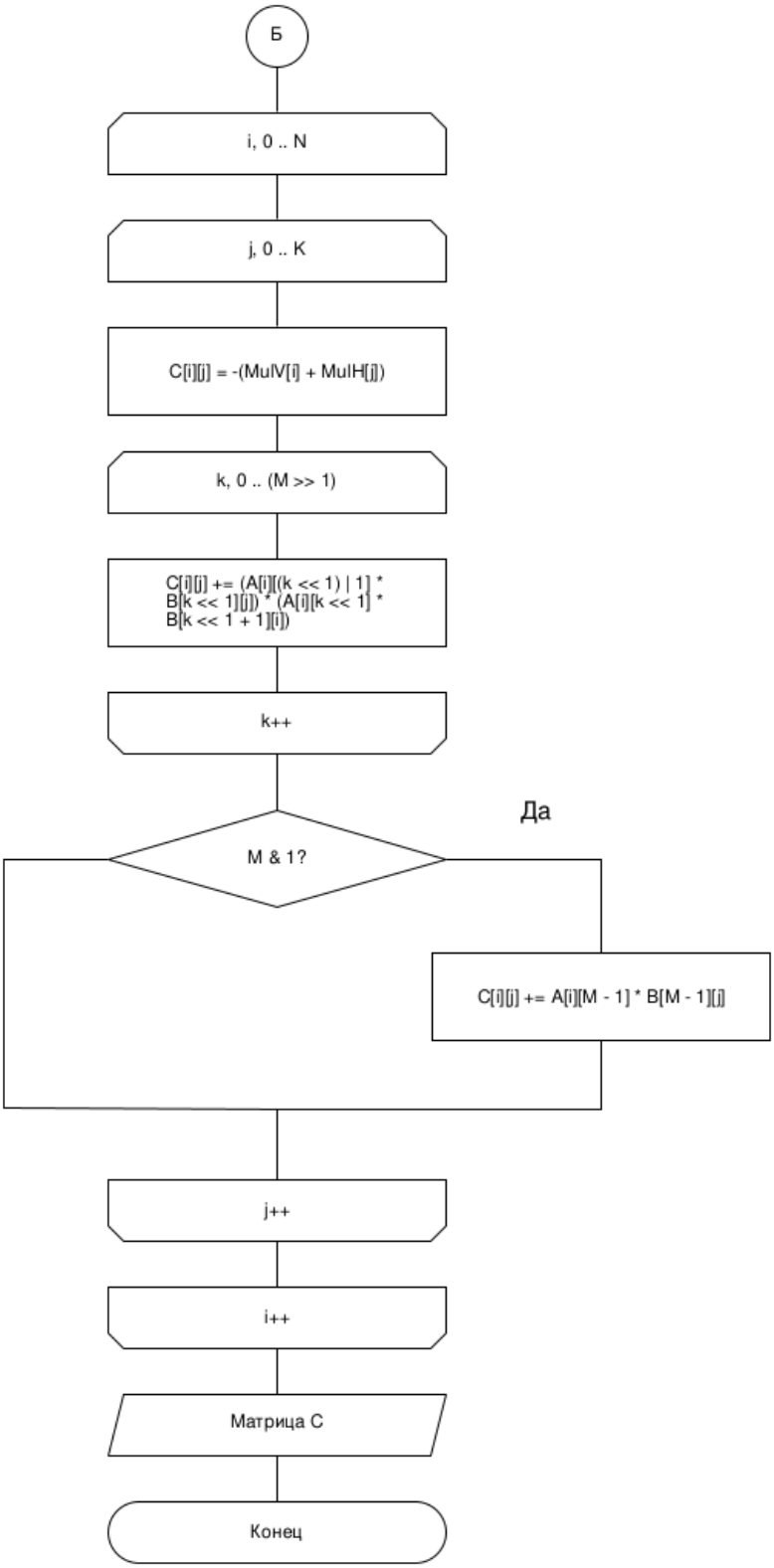

Рисунок 2.5 – Оптимизированный алгоритм Винограда. Часть 2

2.1.1 Трудоемкость алгоритмов

В следующих подразделах будут рассмотрены трудоемкость алгоритмов умножения матриц. Во всех последующих алгоритмах не будем учитывать инициализацию матрицы, в которую записывается результат, потому что данное действие есть во всех алгоритмах. Размер матрицы A $N \times M$, размер матрицы B $M \times K$.

2.1.2 Модель вычислений

Следующие операции будут иметь трудоемкость 1:

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

Следующие операции будут иметь трудоемкость 2:

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

Трудоемкость цикла будет рассчитываться как:

$$f_{cycly} = f_{init} + f_{check} + N(f_{body} + f_{check}), \tag{2.3}$$

где N - количество итераций цикла.

Трудоемкость условного оператора будет определяться как:

$$f_{if} = \begin{cases} f_{check}, & \text{трудоемкость в случае невыполнения условия,} \\ f_{check} + f_{body}, & \text{иначе.} \end{cases} \tag{2.4}$$

2.1.3 Классический алгоритм

Трудоемкость

— внешнего цикла $i = \overline{1, N}$:

$$f_i = 2 + N \cdot (f_{ibody} + 2); \tag{2.5}$$

— внутреннего цикла $j = \overline{1, K}$:

$$f_{ibody} = 2 + K \cdot (f_{jbody} + 2); \tag{2.6}$$

— внутреннего цикла $k = \overline{1, M}$:

$$f_{jbody} = 2 + M \cdot (12 + 2); \tag{2.7}$$

Тогда, если объединить (2.1) — (2.3):

$$f = 2 + N \cdot (4 + K \cdot (4 + 14M)); \tag{2.8}$$

После упрощения (2.8):

$$f = 14NKM + 4NK + 4N + 2 \approx 14NKM. \tag{2.9}$$

2.1.4 Алгоритм Винограда

Трудоемкость

— создания и инициализации массивов $MulH$ и $MulV$:

$$f_{init} = M + N; \tag{2.10}$$

— Заполнения массива $MulH$:

$$f_H = 2 + N(2 + \frac{M}{2} \cdot 17); \tag{2.11}$$

— Заполнения массива $MulV$:

$$f_V = 2 + K(2 + \frac{M}{2} \cdot 17); \tag{2.12}$$

— основного цикла умножения для четных размеров матриц:

$$f_{cycle} = 2 + N(2 + K(11 + 14M)); \tag{2.13}$$

— цикла умножения последней нечетной строки и последнего нечетного столбца:

$$f_{odd} = \begin{cases} 2, & \text{четный размер} \\ 2 + N(2 + 14K), & \text{иначе} \end{cases} \tag{2.14}$$

.

Для нечетной размерности матрицы трудоемкость:

$$f_1 = f_{init} + f_H + f_V + f_{cycle} + f_{odd} \approx 14NMK. \tag{2.15}$$

2.1.5 Оптимизированный алгоритм Винограда

Трудоемкость

— создания и инициализации массивов $MulH$ и $MulV$:

$$f_{init} = M + N; \tag{2.16}$$

— Заполнения массива $MulH$:

$$f_H = 2 + N(2 + \frac{M}{2} \cdot 11); \tag{2.17}$$

— Заполнения массива $MulV$:

$$f_V = 2 + K(2 + \frac{M}{2} \cdot 11); \tag{2.18}$$

— цикл умножения для четных размеров матриц:

$$f_{cycleeven} = 2 + N(2 + K(11 + 8.5 \cdot M)); \tag{2.19}$$

— цикл умножения для нечетных размеров матриц:

$$f_{cycleodd} = 2 + N(2 + K(22 + 8.5 \cdot M)); \tag{2.20}$$

Для нечетной размерности матриц трудоемкость:

$$f_1 = f_{init} + f_H + f_V + f_{cycleodd} \approx 8.5 \cdot NMK. \tag{2.21}$$

ВЫВОД

В данном разделе были рассмотрены представления алгоритмов и трудоемкость их выполнения. Классический алгоритм и обычный алгоритм Винограда имеют одинаковую трудоемкость. Оптимизированный алгоритм Винограда имеет наименьшую трудоемкость из представленных алгоритмов.

3 Технологическая часть

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

3.1 Требования к программному обеспечению

Входные данные: матрица A и B, где количество столбцов первой матрицы равно количеству строк второй матрицы;

Выходные данные: матрица C.

3.2 Средства реализации

Для реализации алгоритмов был выбран язык C, потому что можно контролировать количество выделяемой и используемой памяти в ЭВМ. Это необходимо на ЭВМ с ограниченным количеством оперативной памяти.

3.3 Реализация алгоритмов

В листингах 3.1 — 3.3 предоставлены листинги реализации алгоритмов умножения матриц.

Листинг 3.1 – Классический алгоритм умножения матриц

```
1  matrix_t *matrix_mult(const matrix_t *A, const matrix_t *B)
2  {
3      if (!A || !B || A->cols != B->rows)
4      {
5          return NULL;
6      }
7      matrix_t *C = matrix_alloc(A->rows, B->cols);
8      if (!C)
9      {
10         return NULL;
11     }
```

```
12      float **data_C = C->data;
13      float **data_A = A->data;
14      float **data_B = B->data;
15      for (size_t i = 0; i < A->rows; i++)
16      {
17          for (size_t j = 0; j < B->cols; j++)
18          {
19              // data_C[i][j] = 0.0f;
20              for (size_t k = 0; k < A->cols; k++)
21              {
22                  data_C[i][j] = data_C[i][j] + data_A[i][k] *
                        data_B[k][j];
23              }
24          }
25      }
26      return C;
27  }
```

Листинг 3.2 – Алгоритм умножения матриц Винограда

```
1  matrix_t *matrix_mult_vinograd(const matrix_t *A, const matrix_t *B)
2  {
3      if (!A || !B || A->cols != B->rows)
4      {
5          return NULL;
6      }
7      size_t N = A->rows, M = A->cols, K = B->cols;
8      float *MulH = (float *)calloc(N, sizeof(float));
9      if (!MulH)
10     {
11         return NULL;
12     }
13     float *MulV = (float *)calloc(K, sizeof(float));
14     if (!MulV)
15     {
16         free(MulH);
17         return NULL;
18     }
19     matrix_t *C = matrix_alloc(A->rows, B->cols);
20     if (!C)
21     {
22         free(MulH);
```

```
23          free(MulV);
24          return NULL;
25      }
26      float **data_C = C->data;
27      float **data_A = A->data;
28      float **data_B = B->data;
29      for (size_t i = 0; i < N; i++)
30      {
31          for (size_t k = 0; k < M / 2; k++)
32          {
33              MulH[i] = MulH[i] + data_A[i][2 * k] * data_A[i][2 * k + 1];
34          }
35      }
36      for (size_t i = 0; i < K; i++)
37      {
38          for (size_t k = 0; k < M / 2; k++)
39          {
40              MulV[i] = MulV[i] + data_B[2 * k][i] * data_B[2 * k + 1][i];
41          }
42      }
43      for (size_t i = 0; i < N; i++)
44      {
45          for (size_t j = 0; j < K; j++)
46          {
47              data_C[i][j] = -MulH[i] - MulV[j];
48              for (size_t k = 0; k < M / 2; k++)
49              {
50                  data_C[i][j] = data_C[i][j] +
51                                                 (data_A[i][2 * k + 1] +
                                                    data_B[2 * k][j]) //*
52                                               * (data_A[i][2 * k] +
                                                    data_B[2 * k + 1][j]);
53              }
54          }
55      }
56      if (M % 2) //if M is odd
57      {
58          for (size_t i = 0; i < N; i++)
59          {
60              for (size_t j = 0; j < K; j++)
61              {
```

```
62                  data_C[i][j] = data_C[i][j] + data_A[i][M - 1] *
                        data_B[M - 1][j];
63              }
64          }
65      }
66      free(MulH);
67      free(MulV);
68      return C;
69  }
```

Листинг 3.3 – Оптимизированный алгоритм умножения матриц Винограда

```
1  matrix_t *matrix_mult_vinograd_opt(const matrix_t *A, const matrix_t *B)
2  {
3      if (!A || !B || A->cols != B->rows)
4      {
5          return NULL;
6      }
7      size_t N = A->rows, M = A->cols, K = B->cols;
8      size_t M2 = (M >> 1);
9      float *MulH = (float *)calloc(N, sizeof(float));
10     if (!MulH)
11     {
12         return NULL;
13     }
14     float *MulV = (float *)calloc(K, sizeof(float));
15     if (!MulV)
16     {
17         free(MulH);
18         return NULL;
19     }
20     matrix_t *C = matrix_alloc(A->rows, B->cols);
21     if (!C)
22     {
23         free(MulH);
24         free(MulV);
25         return NULL;
26     }
27     float **data_C = C->data;
28     float **data_A = A->data;
29     float **data_B = B->data;
30     for (size_t i = 0; i < N; i++)
```

```
31      {
32          for (size_t k = 0; k < M2; k++)
33          {
34              MulH[i] += data_A[i][k << 1] * data_A[i][(k << 1) | 0x1];
35          }
36      }
37      for (size_t i = 0; i < K; i++)
38      {
39          for (size_t k = 0; k < M2; k++)
40          {
41              MulV[i] += data_B[(k << 1)][i] * data_B[(k << 1) | 0x1][i];
42          }
43      }
44      for (size_t i = 0; i < N; i++)
45      {
46          for (size_t j = 0; j < K; j++)
47          {
48              data_C[i][j] -= (MulH[i] + MulV[j]);
49              for (size_t k = 0; k < M2; k++)
50              {
51                  data_C[i][j] +=
52                                                  (data_A[i][(k << 1) |
                                                     0x1] + data_B[(k <<
                                                     1)][j]) //*
53                                              * (data_A[i][(k << 1)] +
                                                   data_B[(k << 1) |
                                                   0x1][j]);
54              }
55              if (M & 1) //if M is odd
56              {
57                  data_C[i][j] += data_A[i][M - 1] * data_B[M - 1][j];
58              }
59          }
60      }
61      free(MulH);
62      free(MulV);
63      return C;
64  }
```

ВЫВОД

В данном разделе были представлены средства реализации, требования к программному обеспечению, технические характеристики и реализация алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Характеристики используемого оборудования:

— Микроконтроллер STM32F303 с ОЗУ 48Кб, процессор до 72 Мгц [2]

4.2 Время выполнения алгоритмов

В таблице 4.1 приведено время выполнения алгоритмов для разных квадратных матриц. На рисунке 4.1 показаны графики выполнения алгоритмов умножения матриц в тиках процессора.

Таблица 4.1 – Время работы алгоритмов (в тиках процессора)

Размер матриц	Классический	Виноград	Виноград оптимизированный
2	0.98	0.75	0.73
3	0.83	0.79	0.74
4	1.01	0.90	0.90
5	1.18	1.15	1.23
6	1.68	1.56	1.50
7	2.10	2.00	2.05
8	2.80	2.46	2.49
9	3.72	3.21	3.36
10	5.60	4.18	5.36
10	5.04	4.50	4.50
21	41.45	30.15	31.04
32	143.91	91.93	90.18
43	316.11	213.24	218.61
54	592.50	367.35	403.39
65	937.01	608.54	620.80
76	1531.09	960.95	967.34
87	2216.74	1423.29	1448.46
98	3178.86	2044.33	2059.81
109	4462.48	2832.23	2927.09

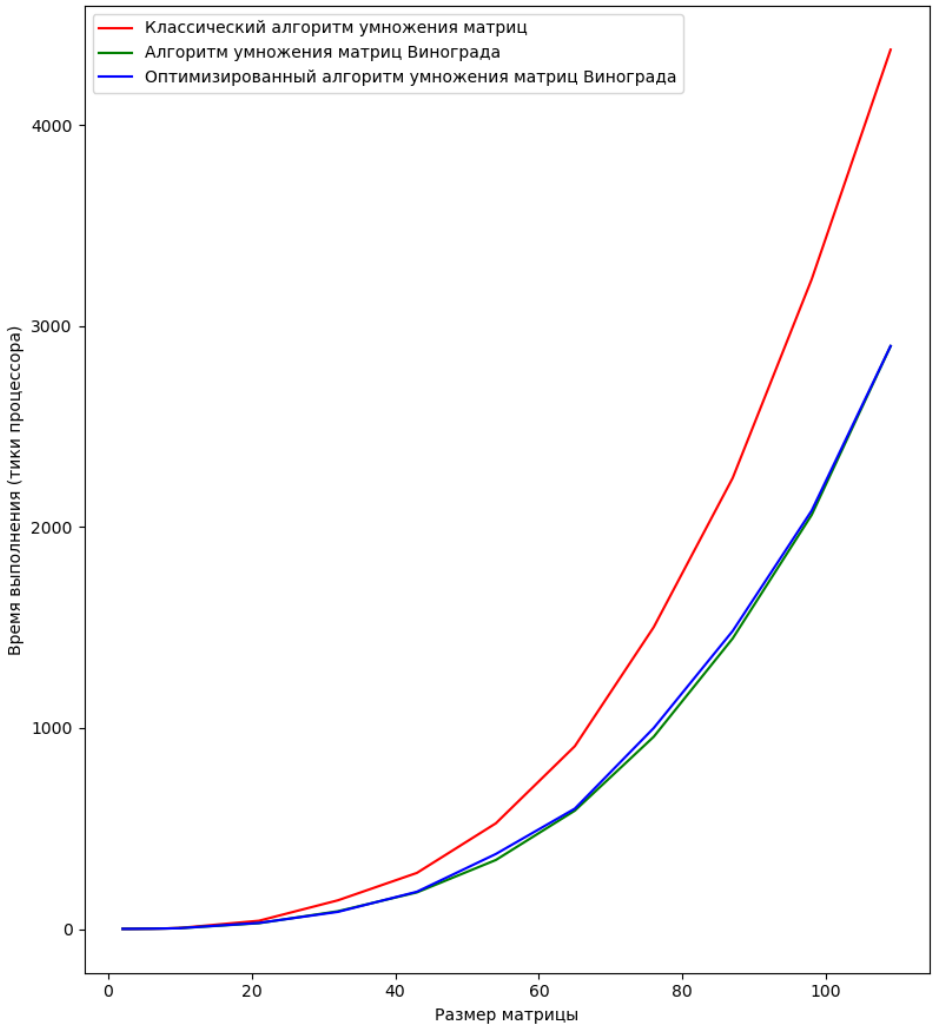

Рисунок 4.1 – Графики в тиках процессора

4.3 Вывод

Сравнения проводились на квадратных матрицах четного и нечетного размеров. Во всех случаях классический алгоритм проигрывает алгоритму Винограда по количеству необходимых тиков процессора на выполнения задачи. Оптимизированная и обычная версии алгоритма Винограда используют довольно одинаковое количество тиков процессора, отчего можно сказать, что их производительность примерно одинаковая. Возможно, это произошло по причие оптимизаций

компилятора.

ЗАКЛЮЧЕНИЕ

Экспериментально показано, что оба алгоритма Винограда требуют меньше процессорного времени, чем алгоритм классического умножения матриц. Оптимизированный алгоритм Винограда имеет наименьшую трудоемкость, когда алгоритм классического умножения матриц имеет наивысшую трудоемкость.

В ходе выполнения данной лабораторной работы были решены следующие задачи:

— реализованы указанные алгоритмы;

— сравнено требуемое время выполнения алгоритмов в тиках процессора;

— описаны и обоснованы полученные результаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Матрицы: примеры [Электронный ресурс] URL: https://spravochnick.ru/matematika/matricy_primery_s_resheniem_i_obyasneniem/ (дата обращения: 27.09.24)

[2] Микроконтроллер STM32F303 Discovery [Электронный ресурс] URL: https://www.st.com/en/evaluation-tools/stm32f3discovery.html#st-also-like (дата обращения: 27.09.24)