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 Смирнов И.В.

Группа ИУ7-52Б

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

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Прямоугольной матрицей называется совокупность чисел, расположенных в виде прямоугольной таблицы, содержащей n строк и m столбцов.

Цель лабораторной работы — выполнить оценки ресурсной эффективности алгоритмов умножения матриц и их реализации. Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать математическую основу стандартного алгоритмов и алгоритма Винограда умножения матриц;

— описать модель вычислений;

— разобрать алгоритмы умножения матриц (стандартный, Винограда, оптимизированный алгоритм Винограда;

— выполнить оценку трудоемкости разрабатываемых алгоритмов;

— реализовать разработанные алгоритмы в программном обеспечении с 2 режимами работы (одиночного расчета и массированного замера процессорного времени выполнения реализации каждого алгоритма);

— выполнить замеры процессорного времени выполнения реализации разработанных алгоритмов в зависимости от варьируемого линейного размера матриц;

— выполнить сравнительный анализ рассчитанных трудоемкостей и результатов замера процессорного времени выполнения реализации трех алгоритмов с учетом лучшего и худшего случаев по трудоемкости

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы умножения матриц.

1.1 Описание алгоритмов

Классический (стандартный) алгоритм перемножения матриц — реализация математического определения умножения матриц. Имеет ассимптотическую сложность $O(n^3)$.

Алгоритм Винограда имеет ассимптотическую сложность $O(n^{2.3755})$, поэтому является одним из самых эффективных по времени алгоритмом умножения матриц [1].

Оптимизированный алгоритм Винограда улучшает классическую версию алгоритма Винограда для умножения матриц, устраняя некоторые избыточные операции и минимизируя накладные расходы, что приводит к повышению производительности [2].

ВЫВОД

В данном разделе были рассмотрены три основных алгоритма для умножения матриц: классический, алгоритм Винограда, оптимизированный алгоритм Винограда.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов умножения матриц, а также оценены трудоемкости каждого из алгоритмов.

2.1 Представление алгоритмов

На вход алгоритмов подаются две матрицы: M_1 размера $M \times N$, M_2 размера $N \times K$, где M, N, K — неотрицательные целые числа; на выходе - матрица M_3 размера $M \times K$.

На рисунках 2.1 — 2.3 приведены схемы трех алгоритмов умножения матриц: классического, Винограда, оптимизированного Винограда.

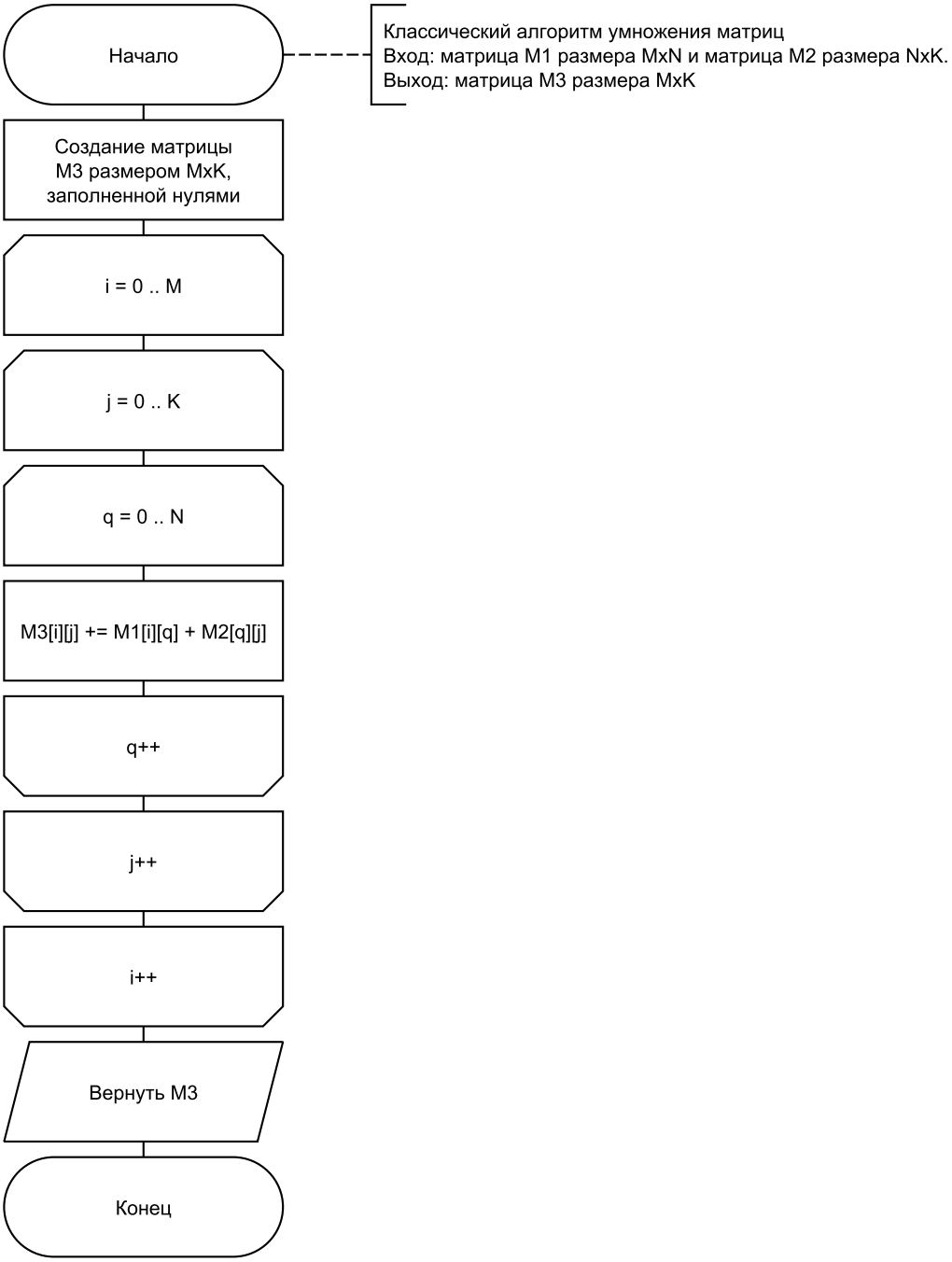
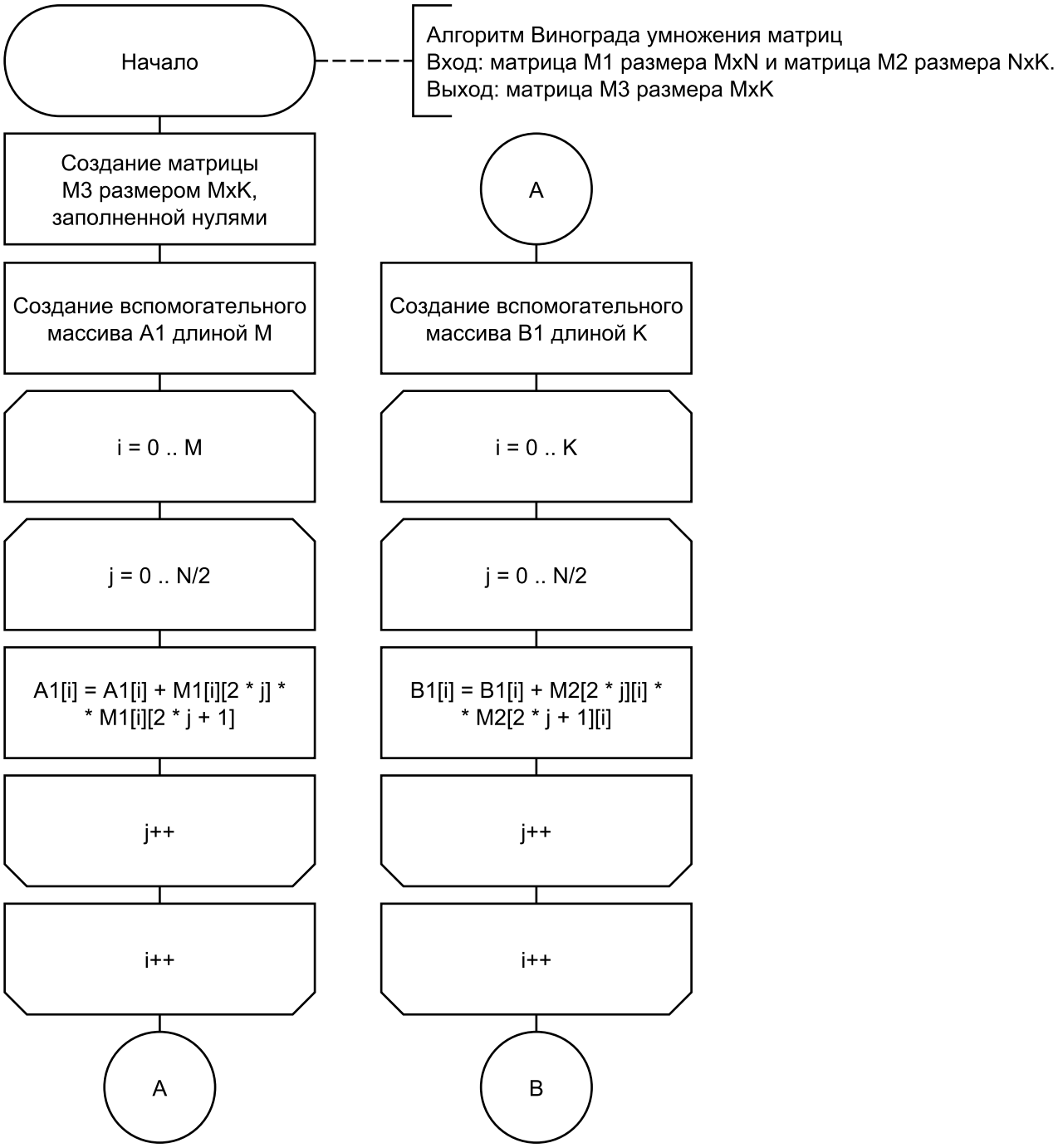


Рисунок 2.1 – Схема классического алгоритма умножения матриц


Начало
Алгоритм Винограда умножения матриц
Вход: матрица M1 размера MxN и матрица M2 размера NxK.
Выход: матрица M3 размера MxK
Создание матрицы M3 размером MxK, заполненной нулями
Создание вспомогательного массива A1 длиной M
i = 0 .. M
j = 0 .. N/2
A1[i] = A1[i] + M1[i][2 * j] * * M1[i][2 * j + 1]
j++
i++
A
A
Создание вспомогательного массива B1 длиной K
i = 0 .. K
j = 0 .. N/2
B1[i] = B1[i] + M2[2 * j][i] * * M2[2 * j + 1][i]
j++
i++
B

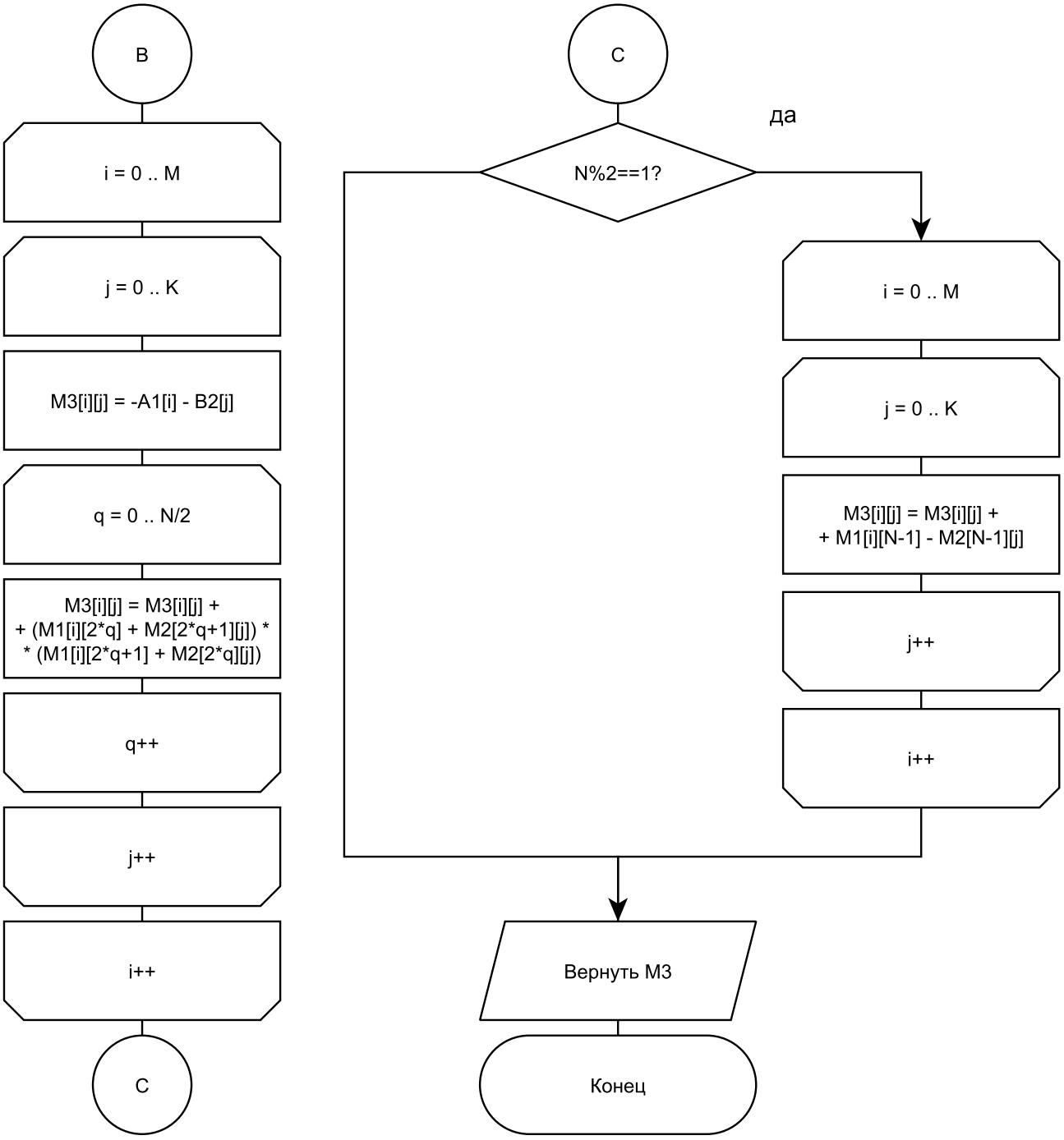
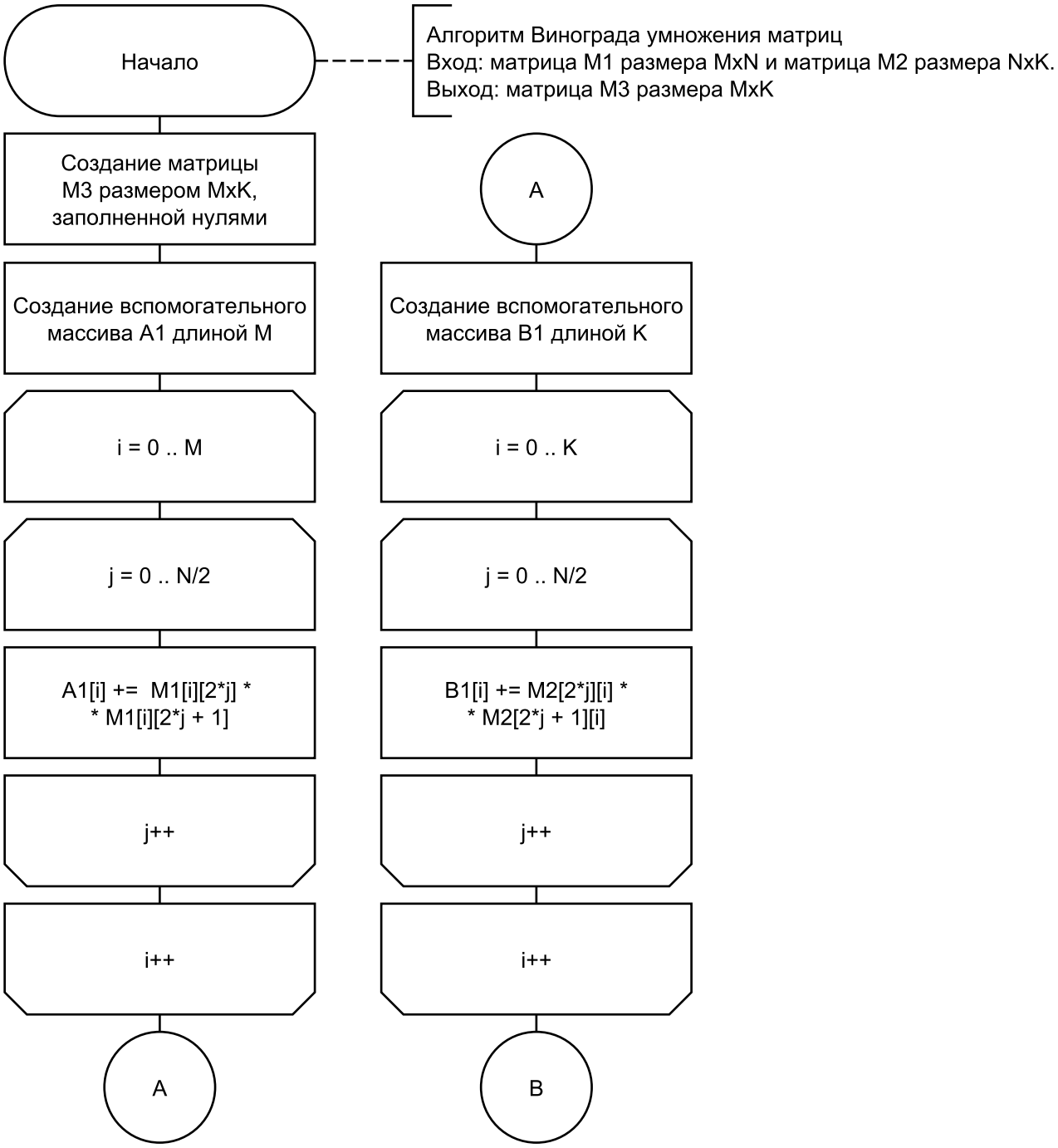


Рисунок 2.2 – Схема алгоритма Винограда умножения матриц


Начало
Алгоритм Винограда умножения матриц
Вход: матрица M1 размера MxN и матрица M2 размера NxK.
Выход: матрица M3 размера MxK
Создание матрицы M3 размером MxK, заполненной нулями
Создание вспомогательного массива A1 длиной M
i = 0 .. M
j = 0 .. N/2
A1[i] += M1[i][2*j] * * M1[i][2*j + 1]
j++
i++
A
A
Создание вспомогательного массива B1 длиной K
i = 0 .. K
j = 0 .. N/2
B1[i] += M2[2*j][i] * * M2[2*j + 1][i]
j++
i++
B

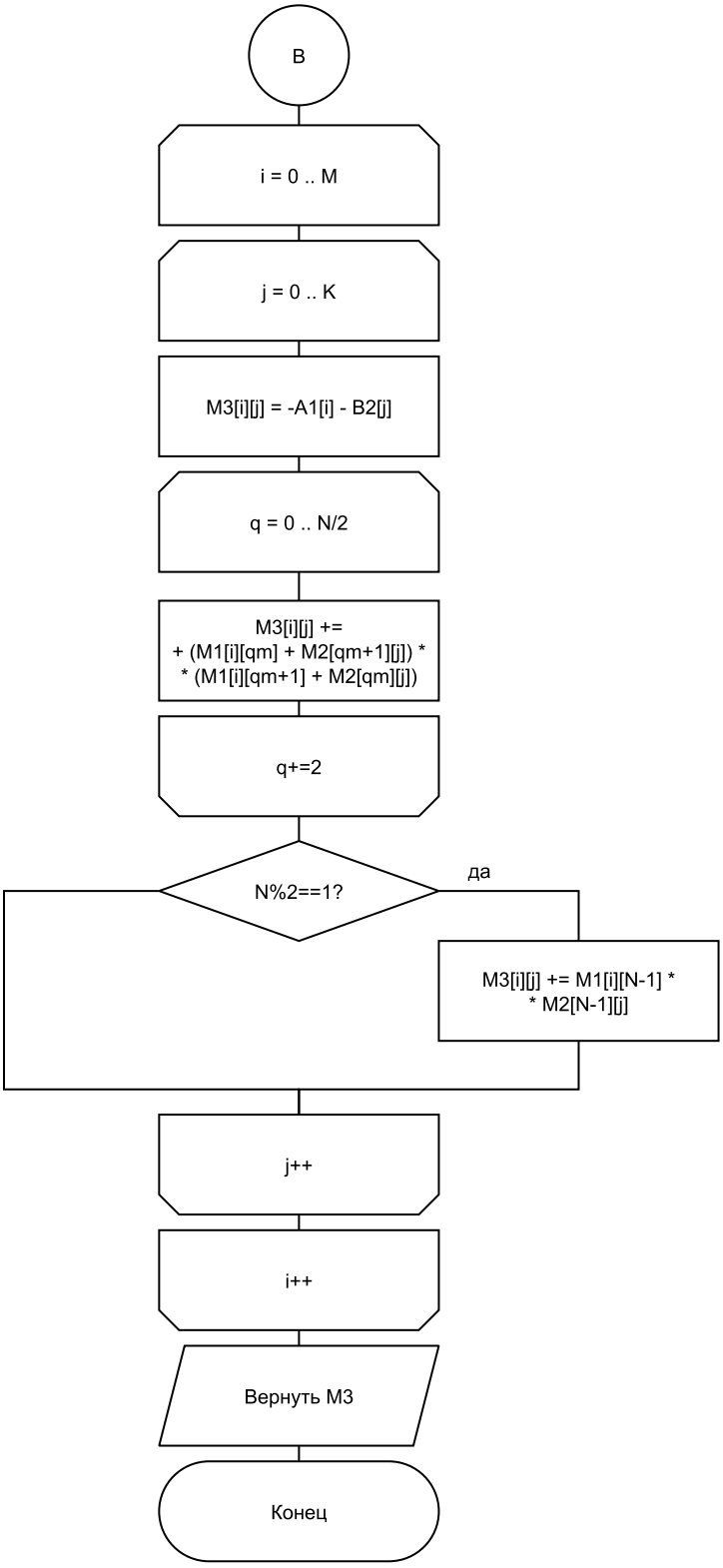


Рисунок 2.3 – Схема оптимизированного алгоритма Винограда

Оптимизация алгоритма Винограда заключается:

— в инкременте наиболее вложенного счетчика цикла на 2;

— в замене операций x = x + k на x += k;

— в объединении III и IV части алгоритма .

2.2 Модель вычислений

Для последующего вычисления трудоемкости была введена модель вычислений:

1. операции из списка (2.1) имеют трудоемкость 1;

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

2. трудоемкость условного оператора `if условие then A else B` рассчитывается как (2.2);

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

3. трудоемкость цикла рассчитывается как (2.3);

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

4. трудоемкость вызова функции/возврата результата равна 0.

2.3 Трудоемкость алгоритмов

В следующих частях будут рассчитаны трудоемкости представленных ранее классического алгоритма, алгоритма Винограда, оптимизированного алгоритма Винограда. Трудоемкость инициализации результирующей матрицы учитываться

не будет, поскольку данное действие есть во всех алгоритмах и не является самым трудоемким.

Введем обозначения:

— M — кол-во строк первой матрицы;

— N — кол-во столбцов первой матрицы и кол-во строк второй матрицы;

— K — кол-во столбцов второй матрицы.

2.3.1 Классический алгоритм перемножения матриц

Трудоемкость стандартного алгоритма умножения матриц состоит из:

— внешнего цикла по $i \in [1..M]$, трудоемкость которого: $f = 2 + M \cdot (2 + f_{body})$;

— цикла по $j \in [1..K]$, трудоемкость которого: $f = 2 + K \cdot (2 + f_{body})$;

— цикла по $q \in [1..N]$, трудоемкость которого: $f = 2 + 10 \cdot N$.

Трудоемкость классического алгоритма равна трудоемкости внешнего цикла. Ее можно вычислить, подставив циклы тела (2.4):

$$f_{classic} = 2 + M \cdot (4 + K \cdot (4 + 10N)) = 2 + 4M + 4MK + 10MNK \approx 10MNK \quad (2.4)$$

2.3.2 Алгоритм Винограда

Трудоемкость алгоритма Винограда состоит из:

— создания и инициализации массивов A1 и B1, трудоемкость которого (2.11):

$$f_{init} = M + K; \quad (2.5)$$

— заполнения массива A1, трудоемкость которого (2.12):

$$f_{A1} = 2 + M \cdot (4 + \frac{N}{2} \cdot 15); \quad (2.6)$$

— заполнения массива B1, трудоемкость которого (2.13):

$$f_{B1} = 2 + K \cdot (4 + \frac{N}{2} \cdot 15); \tag{2.7}$$

— цикла заполнения для четных размеров, трудоемкость которого (2.14):

$$f_{cycle} = 2 + M \cdot (2 + K \cdot (2 + 7 + 4 + \frac{N}{2} \cdot (4 + 20))); \tag{2.8}$$

— цикла, для дополнения результирующего массива суммой последних нечетных строки и столбца, если общий размер нечетный, трудоемкость которого (2.15):

$$f_{last} = \begin{cases} 2, & \text{размер четный,} \\ 2 + M \cdot (2 + 14K), & \text{иначе.} \end{cases} \tag{2.9}$$

Итого, результирующая трудоемкость алгоритма Винограда равна (2.16)

$$f_{final} = f_{init} + f_{A1} + f_{B1} + f_{cycle} + f_{last} \approx 12MNK \tag{2.10}$$

Алгоритм Винограда (неоптимизированный) имеет большую трудоемкость, чем классический алгоритм.

2.3.3 Оптимизированный алгоритм Винограда

Трудоемкость оптимизированного алгоритма Винограда состоит из:

— создания и инициализации массивов A1 и B1 а также доп. переменной, хранящей N/2, трудоемкость которого (2.11):

$$f_{init} = M + K + 3; \tag{2.11}$$

— заполнения массива A1, трудоемкость которого (2.12):

$$f_{A1} = 2 + M \cdot (4 + \frac{N}{2} \cdot 11); \tag{2.12}$$

— заполнения массива B1, трудоемкость которого (2.13):

$$f_{B1} = 2 + K \cdot (4 + \frac{N}{2} \cdot 11); \tag{2.13}$$

— цикла заполнения для четных размеров, трудоемкость которого (2.14):

$$f_{cycle} = 2 + M \cdot (2 + K \cdot (2 + 7 + 2 + \frac{N}{2} \cdot (17)) + f_{last}); \tag{2.14}$$

где f_{last} — IV часть алгоритма, трудоемкость которой равна (2.15):

$$f_{last} = \begin{cases} 2, & \text{размер четный,} \\ 2 + 11, & \text{иначе.} \end{cases} \tag{2.15}$$

Итого, результирующая трудоемкость оптимизированного алгоритма Винограда равна (2.16)

$$f_{final} = f_{init} + f_{A1} + f_{B1} + f_{cycle} + f_{last} \approx 8.5MNK \tag{2.16}$$

Оптимизированный алгоритм Винограда имеет меньшую трудоемкость, по сравнению с классическим алгоритмом.

ВЫВОД

В данном разделе были представлены схемы алгоритмов умножения матриц, а также приведены трудоемкости каждого из трех 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: две матрицы, где количество столбцов первой матрицы равна количетсву строк второй матрицы;

Выходные данные: матрица, где количество строк равно количеству строк первой матрицы, а количество столбцов равно количеству столбцов второй матрицы.

3.2 Средства реализации

В данной работе для реализации был выбран язык программирования $C++$ [3]. Выбор обсуловлен наличием функции вычисления процессорного времени в библиотеке *std::chrono* [4]. Время было замерено с помощью функции *std::clock* [5].

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 представлены реализации алгоритмов нахождения расстояния Левенштейна и Дамерау–Левенштейна.

Листинг 3.1 – Классический алгоритм

```
1  Matrix Simple(const Matrix& m1, const Matrix& m2)
2  {
3      size_t rows1 = m1.rows();
4      size_t cols1 = m1.columns();
5      size_t cols2 = m2.columns();
6
7      Matrix res(rows1, cols2, 0);
8
9      for (size_t i = 0; i < rows1; ++i)
10         for (size_t j = 0; j < cols2; ++j)
11             for (size_t k = 0; k < cols1; ++k)
12                 res[i][j] = res[i][j] + m1[i][k] * m2[k][j];
13
14     return res;
15 }
```

Листинг 3.2 – Алгоритм Винограда

```
1  Matrix Winograd(const Matrix& m1, const Matrix& m2)
2  {
3      size_t rows1 = m1.rows();
4      size_t cols1 = m1.columns();
5      size_t cols2 = m2.columns();
6
7      Matrix res(rows1, cols2);
8
9      std::vector<int> row_factors(rows1, 0);
10     std::vector<int> col_factors(cols2, 0);
11
12     for (size_t i = 0; i < rows1; ++i)
13         for (size_t j = 0; j < cols1 / 2; ++j)
14             row_factors[i] = row_factors[i] + m1[i][2 * j] * m1[i][2 *
                   j + 1];
15
16     for (size_t i = 0; i < cols2; ++i)
17         for (size_t j = 0; j < cols1 / 2; ++j)
18             col_factors[i] = col_factors[i] + m2[2 * j][i] * m2[2 * j +
                   1][i];
19
20     for (size_t i = 0; i < rows1; ++i)
```

```
21      {
22          for (size_t j = 0; j < cols2; ++j)
23          {
24              res[i][j] = -row_factors[i] - col_factors[j];
25              for (size_t k = 0; k < cols1 -1; k += 2)
26              {
27                  res[i][j] = res[i][j] + (m1[i][k] + m2[k + 1][j]) *
28                      (m1[i][k + 1] + m2[k][j]);
29              }
30          }
31      }
32
33      if (cols1 % 2)
34      {
35          for (size_t i = 0; i < rows1; ++i)
36              for (size_t j = 0; j < cols2; ++j)
37                  res[i][j] = res[i][j] + m1[i][cols1 - 1] *
38                  m2[cols1 - 1][j];
39      }
40
41      return res;
42  }
```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```
1  Matrix WinogradOpt(const Matrix& m1, const Matrix& m2)
2  {
3      size_t rows1 = m1.rows();
4      size_t cols1 = m1.columns();
5      size_t cols2 = m2.columns();
6
7      Matrix res(rows1, cols2);
8
9      std::vector<int> row_factors(rows1, 0);
10     std::vector<int> col_factors(cols2, 0);
11
12     for (size_t i = 0; i < rows1; ++i)
13         for (size_t j = 0; j < cols1 / 2; ++j)
14             row_factors[i] += m1[i][2 * j] * m1[i][2 * j + 1];
15
16     for (size_t i = 0; i < cols2; ++i)
17         for (size_t j = 0; j < cols1 / 2; ++j)
```

```
18              col_factors[i] += m2[2 * j][i] * m2[2 * j + 1][i];
19
20      for (size_t i = 0; i < rows1; ++i)
21      {
22          for (size_t j = 0; j < cols2; ++j)
23          {
24              res[i][j] = -row_factors[i] - col_factors[j];
25              for (size_t k = 0; k < cols1 - 1; k += 2)
26              {
27                  res[i][j] += (m1[i][k] + m2[k + 1][j]) *
28                      (m1[i][k + 1] + m2[k][j]);
29              }
30
31              if (cols1 % 2)
32              {
33                  res[i][j] += m1[i][cols1 - 1] * m2[cols1 - 1][j];
34              }
35          }
36      }
37
38      return res;
39  }
```

ВЫВОД

В данном разделе были рассмотрены требования к программному обеспечению, используемые средства реализации, а также приведены листинги кода для умножения матриц с помощью классического алгоритма, алгоритма Винограда, оптимизированного алгоритма Винограда.

4 Исследовательская часть

4.1 Технические характеристики

Характеристики используемого оборудования:

— Операционная система — Windows 11 Home [6]

— Память — 16 Гб.

— Процессор — Intel(R) Core(TM) i5-10300H CPU @ 2.50ГГц [7]

— Микроконтроллер — STM32F303 [8]

4.2 Описание используемых типов данных

Используемые типы данных:
размер матрицы — целое число типа $size\_t$;
матрица — $std :: vector< std :: vector<int> >$.

4.3 Время выполнения алгоритмов

Результаты замеров времени работы трех алгоритмов умножения матриц приведены в таблице 4.1. Время работы алгоритмов замерялось на микроконтроллере STM32F303 с тактовой частотой до 72 МГц. Замеры времени проводились на матрицах одинаковой длины и усреднялись для каждого набора одинаковых экспериментов. Каждое значение получено путем взятия среднего из 100 измерений. Зависимости времени умножения от размера матрицы для трех алгоритмов представлены на рисунке 4.1.

Таблица 4.1 – Время работы алгоритмов (в мс)

Размер матрицы	Классический	Виноград	Виноград (оптимизированный)
1	0.12	0.34	0.30
2	0.55	0.52	0.50
4	1.05	0.62	0.52
8	1.56	1.56	1.50
10	4.68	3.36	2.08
20	28.60	4.16	19.76
21	96.72	25.48	21.84
32	299.00	92.04	76.96
43	498.16	197.08	172.64
54	890.24	417.56	359.32
65	1628.64	777.40	670.28
76	2272.40	1161.16	991.12
87	3104.16	1944.80	1682.20
98	3307.20	2643.16	2276.04

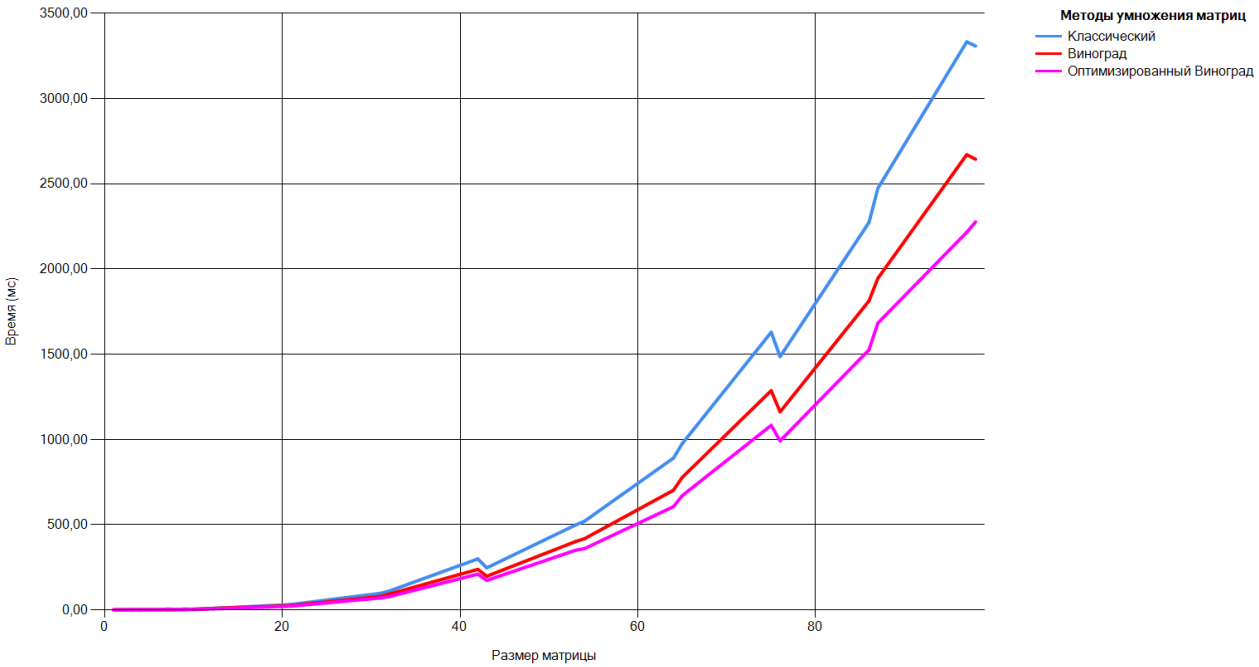


Рисунок 4.1 – Сравнение алгоритмов по времени

4.4 Вывод

В результате исследования было получено, что при больших размерах матриц (свыше 10), алгоритм Винограда работает быстрее стандартного алгоритма более, чем 1.2 раза, а оптимизированный алгоритм Винограда быстрее стандартного алгоритма почти в 1.5 раза. В итоге, можно сказать, что при таких данных следует использовать оптимизированный алгоритм Винограда.

Также было выявлено, что на четных размерах реализация алгоритма Винограда работает быстрее, чем на нечетных размерах матриц, что обусловлено необходимостью проводить дополнительные вычисления для крайних строк и столбцов. Следовательно, стоит использовать алгоритм Винограда для матриц, которые имеют четные размеры.

ЗАКЛЮЧЕНИЕ

В результате исследования было определено, что классический алгоритм умножения матриц проигрывает по времени алгоритму Винограда примерно в 1.2 раза из-за того, что в алгоритме Винограда часть вычислений происходит заранее, а также сокращается часть сложных операций - операций умножения, поэтому предпочтение следует отдавать алгоритму Винограда. Но лучшие показатели по времени выдает оптимизированный алгоритм Винограда – он примерно в 1.2 раза быстрее алгоритма Винограда на размерах матриц свыше 10 из-за замены операций равно и плюс на операцию плюс-равно, за счет замены операции умножения операцией сдвига, а также за счет предвычислений некоторых слагаемых, что дает проводить часть вычислений быстрее. Поэтому при выборе самого быстрого алгоритма предпочтение стоит отдавать оптимизированному алгоритму Винограда.

В ходе выполнения данной лабораторной работы были решены следующие задачи:

— описана математическая основа стандартного алгоритма и алгоритма Винограда умножения матриц;

— представлена модель вычислений для анализа трудоемкости алгоритмов;

— разобраны алгоритмы умножения матриц: стандартный, Винограда и оптимизированный алгоритм Винограда;

— выполнена оценка трудоемкости разработанных алгоритмов;

— реализованы алгоритмы в программном обеспечении с двумя режимами работы: одиночный расчет и массированный замер процессорного времени выполнения каждого алгоритма;

— выполнены замеры процессорного времени выполнения реализации разработанных алгоритмов в зависимости от варьируемого линейного размера матриц;

— проведен сравнительный анализ рассчитанных трудоемкостей и результатов замеров процессорного времени выполнения трех алгоритмов, с учетом лучшего и худшего случаев по трудоемкости.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Understanding 'Winograd Fast Convolution' [Электронный ресурс]. URL: https://medium.com/@dmangla3/understanding-winograd-fast-convolution-a75458744ff (дата обращения: 20.09.2024).

[2] Efficient Winograd Convolution [Электронный ресурс]. URL: https://arxiv.org/pdf/1901.01965 (дата обращения: 20.09.2024).

[3] C++ Programming Language [Электронный ресурс]. URL: https://devdocs.io/cpp/ (дата обращения: 20.09.2024).

[4] C++ Date and time utilities [Электронный ресурс]. URL: https://en.cppreference.com/w/cpp/chrono (дата обращения: 20.09.2024).

[5] std::clock documentation [Электронный ресурс]. URL: https://en.cppreference.com/w/cpp/chrono/c/clock (дата обращения: 20.09.2024).

[6] STM32F303 PDF Documentation [Электронный ресурс]. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f303/documentation.html (дата обращения: 20.09.2024).

[7] Intel® Core™ i5-10300H Processor [Электронный ресурс]. URL: https://ark.intel.com/content/www/us/en/ark/products/201839/intel-core-i5-10300h-processor-8m-cache-up-to-4-50-ghz.html (дата обращения: 20.09.2024).