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 № 4

Название: Параллельные вычисления в умножении матриц

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б _____ М. Р. Саркисян
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Оглавление

Введение

В ходе данной лабораторной работы необходимо изучить и реализовать алгоритмы умножения матриц, в частности, линейный алгоритм Винограда и его распараллеленные модификации.

1 Аналитическая часть

В данном разделе будут формализованы цель и задачи, а также рассмотрено описание параллельных вычислений применительно к задаче умножения матриц.

1.1 Цель и задачи

Целью данной лабораторной работы является изучение возможностей параллельных вычислений применительно к задаче умножения матриц, в частности, умножения матриц с применением алгоритма Винограда.

Задачи:

1. Изучить алгоритм умножения матриц Винограда.
2. Разработать схемы распараллеливания алгоритма умножения матриц Винограда.
3. Реализовать алгоритмы умножения матриц линейно и согласно разработанным схемам параллельных вычислений.
4. Провести сравнительный анализ временной эффективности линейной и параллельных реализаций.

1.2 Определения и формулы

1.2.1 Определение матрицы и операции умножения матриц

Матрицей A размера $[N \times M]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, которая представляет собой совокупность N строк и M столбцов, на пересечении которых находятся элементы[1].

Для двух матриц определена операция умножения, если число столбцов в первой матрице совпадает с числом строк во второй.

Пусть даны две прямоугольные матрицы и размеров $[N \times M]$ и $[M \times Q]$ соответственно. Результатом произведениея матриц A и B будет матрица C размера $[N \times Q]$, элементы c_{ij} которой могут быть вычислены по формуле 1.1.

$$\forall i \in \{1, \ldots, N\}, \forall j \in \{1, \ldots, Q\} : c_{i,j} = \sum_{k=1}^{M} a_{i,k} \cdot b_{k,j} \tag{1.1}$$

1.2.2 Алгоритм умножения матриц Винограда

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$.
Их скалярное произведение равно:

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.2}$$

Это равенство можно переписать в виде 1.3.

$$\begin{aligned} V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - \\ - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \end{aligned} \tag{1.3}$$

Для векторов, размер которых — нечетное число, равенство 1.3 принимает

вид 1.4.

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - \\ - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 + v_5 \cdot w_5 \tag{1.4}$$

Эти два равенства — 1.3 и 1.4 — могут быть обобщены на вектора произвольного размера.

Принцип алгоритма Винограда заключается в использовании равенств вида 1.3 и 1.4 в рамках умножения матриц — так, под векторами V и W понимаются строка певрой матрицы и столбец второй матрицы соответственно.

Выражение в правой части равенств 1.3 допускает предварительную обработку: его части $(v_1 \cdot v_2 + v_3 \cdot v_4)$ и $(w_1 \cdot w_2 + w_3 \cdot w_4)$ можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй соответсвенно.

Так, при вычислении $V \cdot W$ с использованием предварительно вычисленных значений нам необходимо выполнить лишь первые два умножения — $(v_1 + w_2) \cdot (v_2 + w_1)$ и $(v_3 + w_4) \cdot (v_4 + w_3)$ — и посчитать линейную комбинацию полученных значения согласно формуле 1.3.

В случае умножения матриц, строка и столбец которых преставляют собой вектора нечетного размера, схема рассчета элементов результирующей матрицы сохраняется. После чего, к каждому элементу c_{ij} результирующей матрицы прибавляется число $v_{im} \cdot u_{mj}$, где v_{im} - последний элемент i-той строки первой матрицы, u_{mj} - последний элемент j-того столбца второй матрицы.

1.2.3 Параллельные вычисления

При использовании многопроцессорных вычислительных систем с общей памятью обычно предполагается, что имеющиеся в составе системы процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти, и время доступа к памяти является одинаковым (при одновременном доступе нескольких процессоров к одному и тому же элементу памяти очередность и синхронизация доступа обеспечивается на аппаратном уровне).

Распространенный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью – создание новых параллельных методов на основе обычных последовательных программ[2].

Для реализации алгоритма на параллельной системе его следует представить в виде последовательности групп операций, причем операции в каждой группе могут быть выполнены одновременно[2] - то есть каждая операция любой группы зависит либо от входных данных, либо от результатов выполнения операций в предыдущих группах, но не зависит от результатов выполнения других операций в той же группе.

Одновременное выполнение операций группы параллельной системой поддерживается на уровне операционной системы с помощью механизма потоков[3].

Потоки исполняются в общем адресном пространстве параллельной программы. Как результат, взаимодействие параллельных потоков можно организовать через использование общих данных, являющихся доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения. В случае же, когда общие данные могут изменяться несколькими потоками, необходимы специальные усилия для организации правильного взаимодействия — очередного обращения к данным для записи[3].

1.3 Вывод

Итак, в этом разделе были поставлены цель и задачи, рассмотрено описание параллельных вычислений применительно к задаче умножения матриц и кратко охарактеризованы особенности их применения.

2 Конструкторская часть

В данном разделе будут рассмотрены требования к ПО и схемы алгоритмов.

2.1 Требования к программному обеспечению

Входные данные: две матрицы и их размерности. Выходные данные: результат умножения введенных матриц.

На рисунке 2.1 представлена IDEF0-диаграмма, описывающая алгоритм нахождения произведения двух матриц.

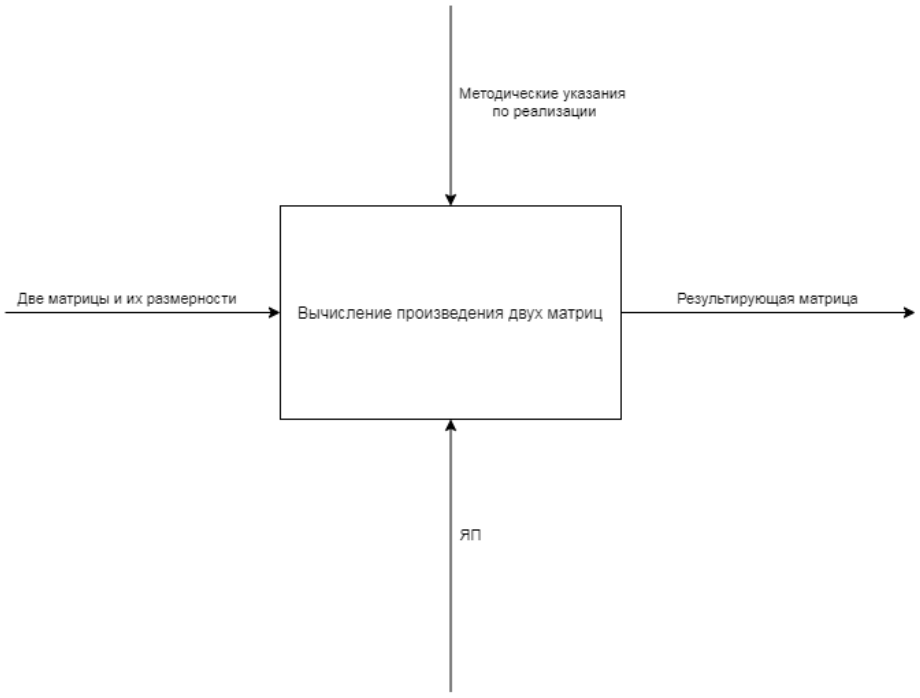


Рис. 2.1: IDEF0-диаграмма алгоритма нахождения произведения двух матриц

Тестирование будет производиться по стратегии чёрного ящика, что означает отсутствие знания о внутреннем коде тестируемого объекта.

2.2 Схемы алгоритмов

На рисунках 2.2, 2.3, 2.4 и 2.5 представлена схема алгоритма умножения матриц Винограда.

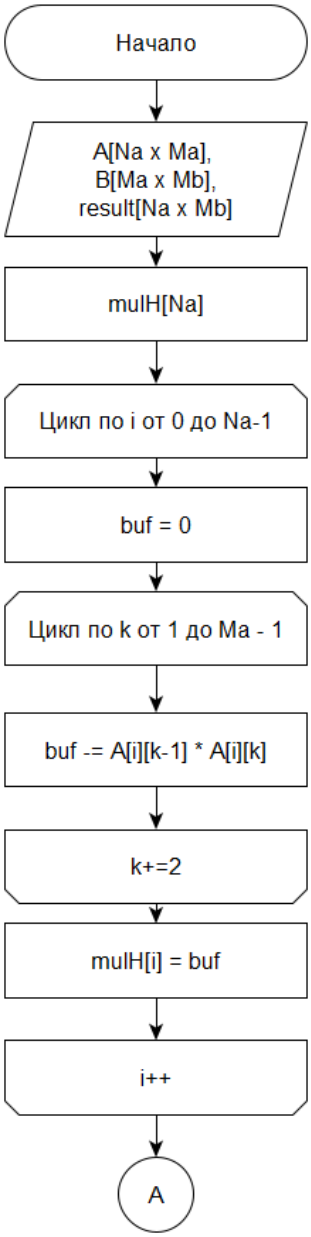


Рис. 2.2: Схема алгоритма Винограда умножения матриц. Часть 1

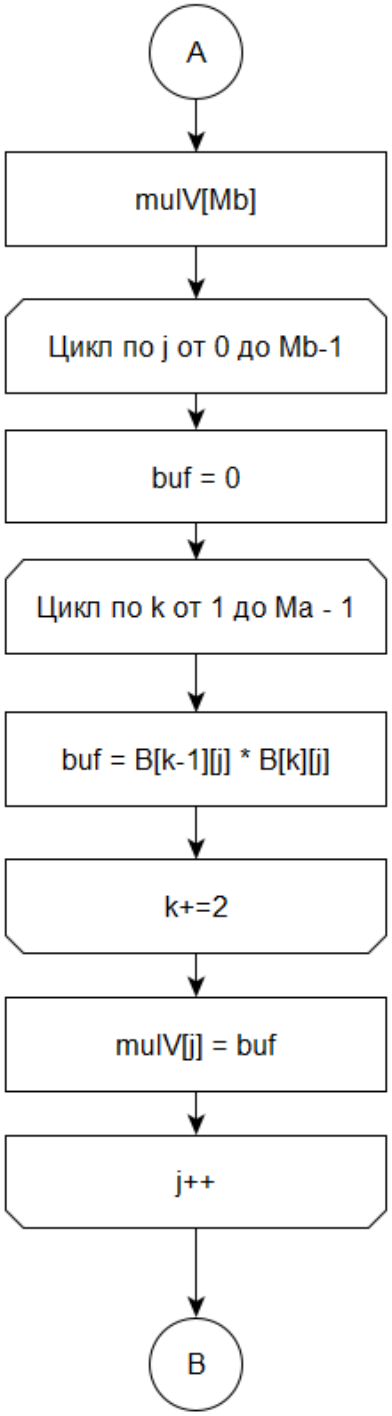


Рис. 2.3: Схема алгоритма Винограда умножения матриц. Часть 2

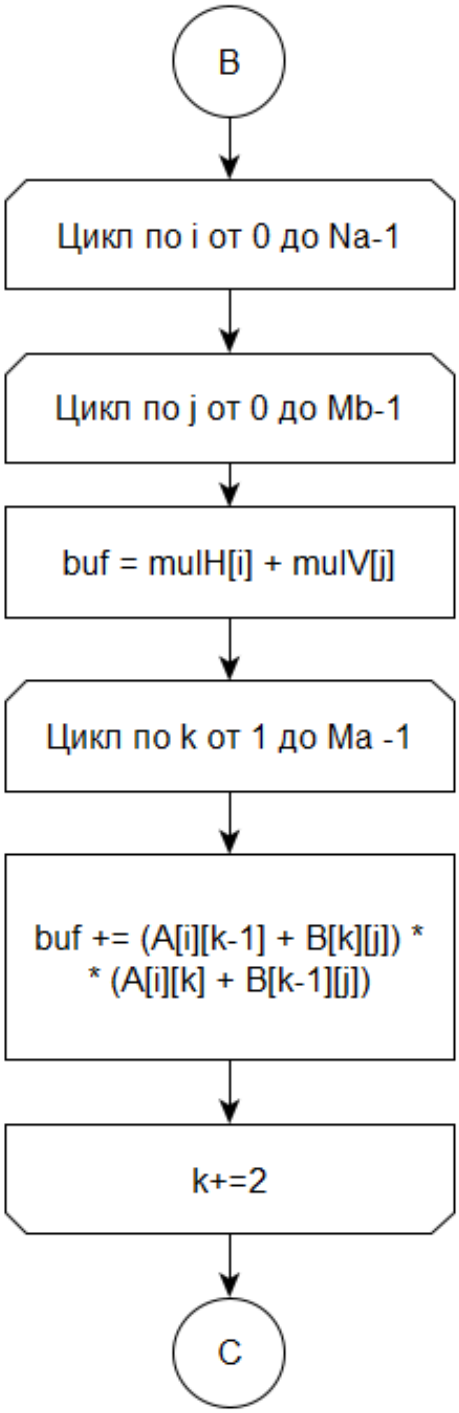


Рис. 2.4: Схема алгоритма Винограда умножения матриц. Часть 3

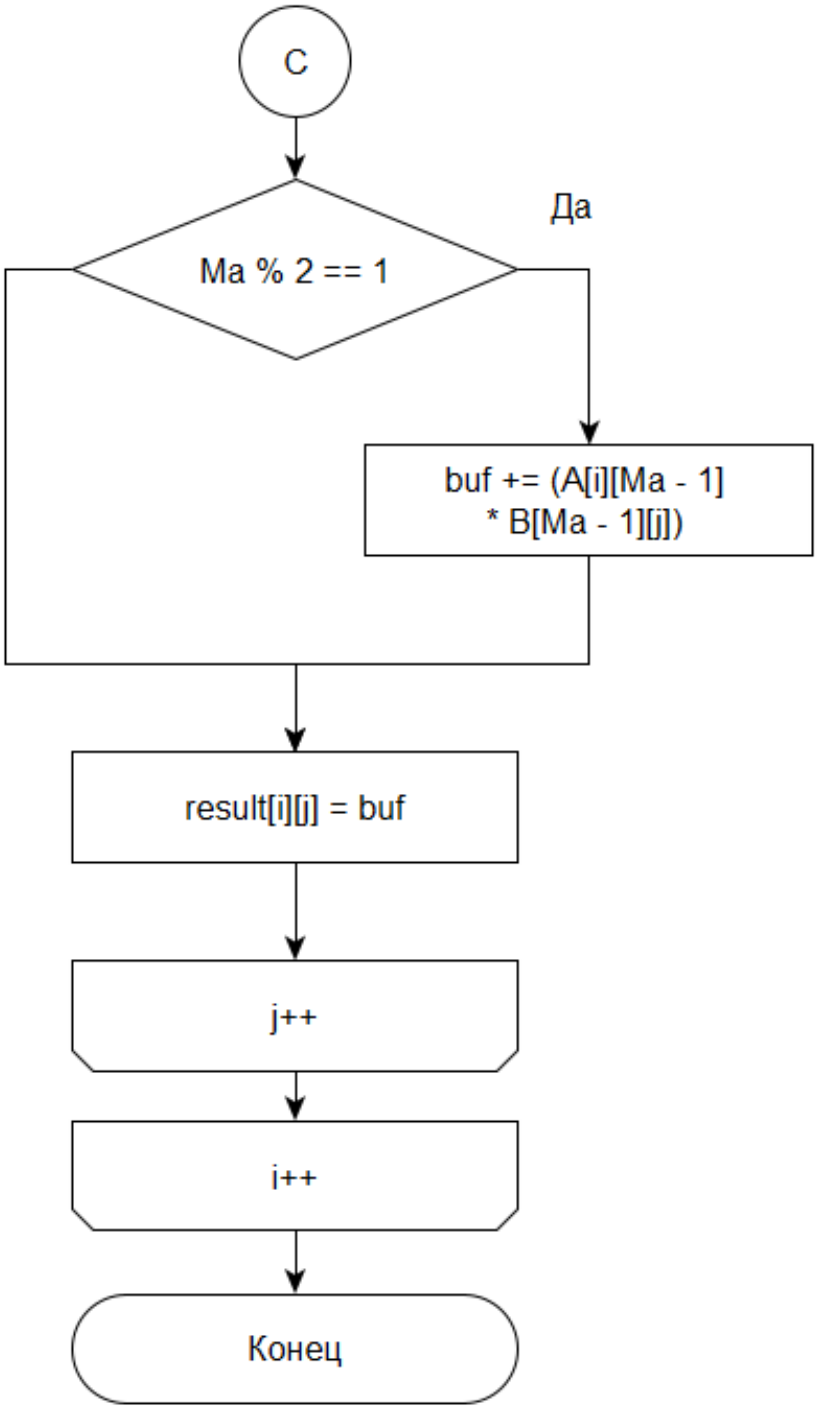


Рис. 2.5: Схема алгоритма Винограда умножения матриц. Часть 4

2.2.1 Схемы параллельных вычислений

Рассмотрим первые два цикла алгоритма, вычисление произведений пар последовательных элементов каждой строки первой матрицы и каждого столбца второй. Вычисления произведений для каждой из строк первой матрицы не зависят от результатов вычислений для других строк. Вычисления произведений для каждого столбца второй матрицы не зависят от результатов вычислений для других столбцов. Вместе с тем, вычисления для строк первой матрицы не зависят от результатов вычислений для столбцов второй.

Участок алгоритма, содержащий два первых цикла, может быть распараллелен по подмножествам строк первой матрицы и по подмножествам столбцов второй: каждый поток будет выполнять предварительные вычисления для определенного подмножества строк первой матрицы или для определенного подмножества столбцов второй.

Рассмотрим этот подход в качестве первой схемы параллельных вычислений.

Рассмотрим участок алгоритма с заполнением результирущей матрицы: вычисление элемента результирующей матрицы для каждой строки первой матрицы не зависит от результатов вычислений для других строк. Можно распараллелить участок алгоритма с заполнением результирующей матрицы по подмножествам строк: каждый поток будет выполнять вычисления для определенного подмножетсва строк результирующей матрицы. В этом участке находятся три вложенных цикла — именно этот участок алгоритма имеет наибольшую сложность.

Рассмотрим этот подход в качестве второй схемы параллельных вычислений.

Таким образом, для изучения выбрано две схемы параллельных вычислений:

- распараллеливание участка алгоритма, содержащего первые два цикла с предварительными вычислениями произведений пар элементов для каждой строки первой матрицы и каждого столбца второй по подмножествам строк первой матрицы и подмножествам столбцов второй;

- распараллеливание участка алгоритма, содержащего три вложенных цикла с вычислением элементов результирующей матрицы по подмножествам

строк результирующей матрицы.

Предположительно[4], из двух описанных схем наибольшую выгоду принесет вторая — распараллеливание участка алгоритма, который содержит три вложенных цикла. Это предположение будет в дальнейшем проверено в исследовательском разделе.

3 | Технологическая часть

В данном разделе будут рассмотрены средства реализации и представлен листинг кода, а также проведено тестирование по стратегии чёрного ящика.

3.1 Средства реализации

Для реализации программы мной был выбран C++[5], так как это язык программирования, содержащий средства создания эффективных программ практически любого назначения. Он также был достаточно подробно изучен мной на предыдущем курсе.

3.2 Листинг кода

В листингах 3.1, 3.2, 3.3 представлены линейная и две модифицированные (распараллеленные) реализации алгоритма умножения матриц Винограда.

Листинг 3.1: Линейная реализация алгоритма умножения матриц Винограда

```
1 int vinogradMulti(int** A, int** B, int** result, int N_a, int M_a, int
      M_b, int dummy)
2 {
3   int* mulH = new int[N_a];
4   int* mulV = new int[M_b];
5
6   int buf;
7   for (int i = 0; i < N_a; i++)
8   {
```

```
9       buf = 0;
10      for (int k = 1; k < M_a; k += 2)
11        buf -= A[i][k - 1] * A[i][k];
12      mulH[i] = buf;
13    }
14
15    for (int j = 0; j < M_b; j++)
16    {
17      buf = 0;
18      for (int k = 1; k < M_a; k += 2)
19        buf -= B[k - 1][j] * B[k][j];
20      mulV[j] = buf;
21    }
22
23    for (int i = 0; i < N_a; i++)
24      for (int j = 0; j < M_b; j++)
25      {
26        buf = mulH[i] + mulV[j];
27        for (int k = 1; k < M_a; k += 2)
28          buf += (A[i][k - 1] + B[k][j]) * (A[i][k] + B[k - 1][j]);
29
30        if (M_a % 2)
31          buf += A[i][M_a - 1] * B[M_a - 1][j];
32
33        result[i][j] = buf;
34      }
35
36    return OK;
37  }
```

Листинг 3.2: Реализация алгоритма Винограда умножения матриц с использованием первой схемы параллельных вычислений

```
1  int vinogradMultiParallel1(int** A, int** B, int** result, int N_a, int
       M_a, int M_b, int t_count)
2  {
3    int* mulH = new int[N_a];
4    int* mulV = new int[M_b];
5    std::thread* threads = new std::thread[t_count];
```

```
6
7   if (t_count > 1)
8   {
9     int proportion = t_count * N_a / (N_a + M_b);
10    int rows_t = (proportion) ? proportion : 1;
11    int cols_t = t_count - rows_t;
12
13    int rows_per_t = N_a / rows_t;
14    int cols_per_t = M_b / cols_t;
15
16    int start_row = 0;
17    for (int i = 0; i < rows_t; i++)
18    {
19      int end_row = (i == rows_t - 1) ? N_a : start_row + rows_per_t;
20      threads[i] = std::thread(vinogradMultiParallel11, A, N_a, M_a,
          mulH, start_row, end_row);
21      start_row = end_row;
22    }
23
24    int start_col = 0;
25    for (int i = rows_t; i < t_count; i++)
26    {
27      int end_col = (i == t_count - 1) ? M_b : start_col + cols_per_t;
28      threads[i] = std::thread(vinogradMultiParallel12, B, M_a, M_b,
          mulV, start_col, end_col);
29      start_col = end_col;
30    }
31
32    for (int i = 0; i < t_count; i++)
33    {
34      threads[i].join();
35    }
36  }
37  else
38  {
39    int buf;
40    for (int i = 0; i < N_a; i++)
41    {
```

```
42        buf = 0;
43        for (int k = 1; k < M_a; k += 2)
44          buf -= A[i][k - 1] * A[i][k];
45        mulH[i] = buf;
46      }
47
48      for (int j = 0; j < M_b; j++)
49      {
50        buf = 0;
51        for (int k = 1; k < M_a; k += 2)
52          buf -= B[k - 1][j] * B[k][j];
53        mulV[j] = buf;
54      }
55    }
56
57    int buf;
58    for (int i = 0; i < N_a; i++)
59      for (int j = 0; j < M_b; j++)
60      {
61        buf = mulH[i] + mulV[j];
62        for (int k = 1; k < M_a; k += 2)
63          buf += (A[i][k - 1] + B[k][j]) * (A[i][k] + B[k - 1][j]);
64
65        if (M_a % 2)
66          buf += A[i][M_a - 1] * B[M_a - 1][j];
67
68        result[i][j] = buf;
69      }
70
71    return OK;
72  }
```

Листинг 3.3: Реализация алгоритма Винограда умножения матриц с использованием второй схемы параллельных вычислений

```
1  int vinogradMultiParallel2(int** A, int** B, int** result, int N_a, int
       M_a, int M_b, int t_count)
2  {
3    int* mulH = new int[N_a];
```

```
4   int* mulV = new int[M_b];
5   std::thread* threads = new std::thread[t_count];
6
7   int buf;
8   for (int i = 0; i < N_a; i++)
9   {
10    buf = 0;
11    for (int k = 1; k < M_a; k += 2)
12      buf -= A[i][k - 1] * A[i][k];
13    mulH[i] = buf;
14  }
15
16  for (int j = 0; j < M_b; j++)
17  {
18    buf = 0;
19    for (int k = 1; k < M_a; k += 2)
20      buf -= B[k - 1][j] * B[k][j];
21    mulV[j] = buf;
22  }
23
24  int rows_per_t = N_a / t_count;
25  int start_row = 0;
26  for (int i = 0; i < t_count; i++)
27  {
28    int end_row = (i == t_count - 1) ? N_a : start_row + rows_per_t;
29    threads[i] = std::thread(vinogradMultiParallel21, A, B, result, N_a
          , M_a, M_b, mulH, mulV, start_row, end_row);
30
31    start_row = end_row;
32  }
33  for (int i = 0; i < t_count; i++)
34  {
35    threads[i].join();
36  }
37
38  return OK;
39 }
```

3.3 Тестирование

Тестирование производилось по стратегии чёрного ящика, что означает отсутствие знания о внутреннем коде тестируемого объекта.

Результаты тестирования приведены в таблице 3.1.

Таблица 3.1: Результаты тестирования

№	n1, m1	n2, m2	matrix1	matrix2	expected result	got result
1	0 0	0 0				
2	1 1	0 0	4			
3	0 0	1 1		1		
4	3 2	1 1	1 3 2 3 2 3	1		
5	1 1	1 2	5	2 3	10 15	10 15
6	2 2	2 2	1 2 0 1	-1 2 1 3	1 8 1 3	1 8 1 3

3.4 Вывод

В данном разделе рассматривались средства реализации и представлены листинги кода вышеописанных алгоритмов с последующим их тестированием.

4 | Исследовательская часть

В данном разделе будет представлено сравнение реализаций алгоритма по времени.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Результаты замеров времени выполнения реализаций приведены на произвольно заполненных квадратных матрицах. Замеры времени произведены с помощью функции $std :: chrono :: high\_resolution\_clock :: now()$.

Было произведено две серии замеров:

- для матриц размером $N \times N = 500$ при переменном числе потоков для параллельных реализаций, число потоков $T \in \{1, 2, 4, 8, 16, 32\}$;
- для матриц размером $N \times N$, $N \in \{100, 200, 300, 400, 500\}$ с постоянным числом потоков для параллельных реализаций, равным 4.

Каждый замер времени был произведен на 100 итерациях, в качестве результата взято усредненное время работы алгоритма.

Технические характеристики устройства, на котором проводились замеры времени:

- операционная система Windows 10 64-bit;
- память 8 ГБ;

- процессор Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz;
- 4 логических процессоров.

Результаты замеров времени приведены в таблицах 4.1 и 4.2, графически представлены на рисунках 4.1 и 4.2.

Таблица 4.1: Результаты первой серии замеров процессорного времени в мсек. для реализаций алгоритмов умножения матриц, T — число потоков

T	Линейно	Паралл. схема 1	Паралл. схема 2
1	334	337	341
2	338	355	232
4	334	339	163
8	331	339	171
16	330	345	169
32	338	355	186

Таблица 4.2: Результаты второй серии замеров процессорного времени в мсек. для реализаций алгоритмов умножения матриц, N — количество строк входных квадратных матриц

N	Линейно	Паралл. схема 1	Паралл. схема 2
100	2	4	3
200	19	22	12
300	69	70	36
400	167	170	84
500	332	336	162

Рис. 4.1: Результаты первой серии замеров процессорного времени

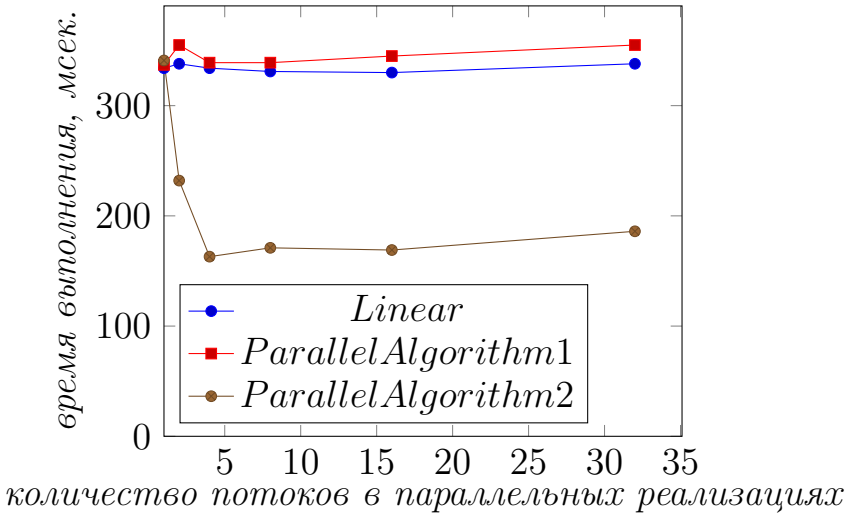
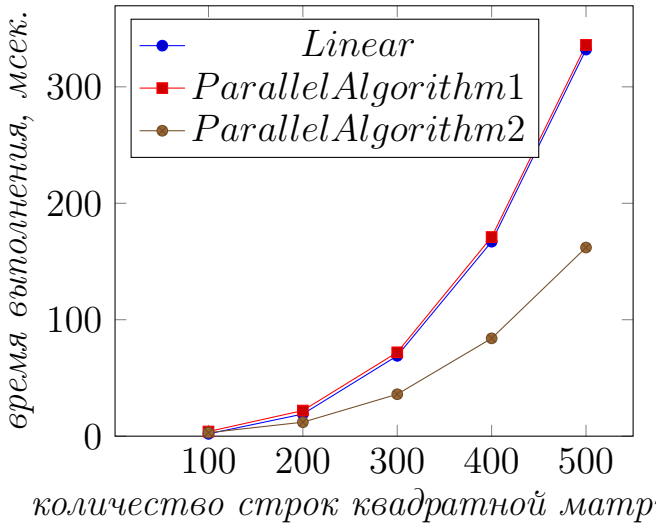


Рис. 4.2: Результаты второй серии замеров процессорного времени



4.2 Вывод

На входных матрицах небольших размерностей линейная реализация алгоритма работает быстрее, так как накладные расходы на параллельные реализации — создание потоков, дополнительные вычисления — превышают выгоду во времени на умножение матриц. При увеличении размерностей входных матриц, выгода во времени на умножение параллельной реализации превышает накладные расходы, и параллельная реализация выигрывает по времени над линейной.

Это продемонстрировано на рисунке 4.2.

Первый же вариант распараллеливания оказался неуспешным, так как незначительно уступает по времени в сравнении с линейной реализацией алгоритма при любых числе потоков и размере матриц.

На моем устройстве вторая параллельная реализация работает в среднем в 2 раза быстрее, чем линейная и первая параллельная реализация. Это продемонстрировано на рисунке 4.2.

До определенного момента при увеличении числа потоков время исполнения параллельных реализаций уменьшается на таких же входных данных, но по достижению количества потоков, равного 16 — количество логических процессоров устройства, умноженное на 4 — снова начинает расти. Это продемонстрировано на рисунке 4.1.

Заключение

В ходе выполнения работы решены следующие задачи:

- изучен алгоритм умножения матриц Винограда;
- разработаны схемы распараллеливания алгоритма Винограда умножения матриц;
- реализованы алгоритмы умножения матриц — линейно и согласно разработанным схемам параллельных вычислений;
- экспериментально подтверждены различия во временно́й эффективности линейной и параллельных реализаций на квадратных матрицах;

Различия во временной эффективности, подтвержденные экспериментально на материале замеров времени исполнения реализаций, отвечают ожиданиям.

Вследствие исследования было установлено, что на входных матрицах небольших размерностей линейная реализация алгоритма работает быстрее, так как накладные расходы на параллельные реализации превышают выгоду во времени. При увеличении размерностей входных матриц, выгода во времени на умножение параллельных реализаций превышает накладные расходы, и параллельные реализации выигрывают по времени над линейной.

При увеличении размерностей входных данных, выгода по времени второй схемы параллельных вычислений превышает выгоду первой.

Первый же вариант распараллеливания оказался неуспешным, так как незначительно уступает по времени в сравнении с линейной реализацией алгоритма

при любых числе потоков и размере матриц. Это объясняется тем, что информацию, полученную от потоков, необходимо синхронизировать, а создание новых потоков требует времени и памяти из-за копирования данных. Всё это ограничивает применение многопоточности.

На моем устройстве вторая схема параллельных вычислений работает в среднем в 1.3 раза быстрее, чем первая на таких же вхожных данных, и в 2 раза быстрее, чем линейная реализация.

Для числа потоков, меньшего 16 — количество логических процессоров устройства, умноженное на 4 — с увеличением числа потоков время исполнения параллельных реализаций уменьшается, однако по достижению — снова начинает увеличиваться.

Список использованных источников

1. Белоусов И. В. Матрицы и определители, учебное пособие по линейной алгебре. // М.: Изд-во МГТУ им. Н.Э. Баумана. 2006.

2. Демьянович Ю. К. О параллельных вычислениях // КИО. 2007. [Электронный ресурс.]

   URL: https://cyberleninka.ru/article/n/o-parallelnyh-vychisleniyah

   Дата обращения: 22.10.2020.

3. Гладышев Е. И., Мурыгин А. В. Многопоточность в приложениях // Актуальные проблемы авиации и космонавтики. 2012. [Электронный ресурс.]

   URL: https://cyberleninka.ru/article/n/mnogopotochnost-v-prilozheniyah

   Дата обращения: 22.10.2020.

4. Желудков А. В., Макаров Д. В., Фадеев П. В. Исследование программных потоков на примере задачи умножения матриц // Символ науки. 2016. №11-3. [Электронный ресурс.]

   URL: https://cyberleninka.ru/article/n/issledovanie-programmnyh-potokov-na-primere-zadachi-umnozheniya-matrits

   Дата обращения: 23.10.2020.

5. Руководство по языку C++ [Электронный ресурс.]

URL: https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=vs-2019

Дата обращения: 23.10.2020.