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 №4 по дисциплине "Анализ алгоритмов"

Тема Параллельные вычисления применительно к задаче умножения матриц

Студент Андрич К.

Группа ИУ7И-56Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Термин «матрица» применяется во множестве разных областей: от программирования до кинематографии.

Матрица в математике – это таблица чисел, состоящая из определенного количества строк (m) и столбцов (n).

Мы встречаемся с матрицами каждый день, так как любая числовая информация, занесенная в таблицу, уже в какой-то степени считается матрицей.

Примером могут служить:

- список телефонных номеров;
- различные статистические данные;
- табель успеваемости ученика и многое другое.

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

В рамках достижения поставленной цели требуется решить следующие задачи:

- изучить алгоритм умножения матриц Винограда;
- разработать схемы распараллеливания алгоритма Винограда умножения матриц;
- реализовать алгоритмы умножения матриц линейно и согласно разработанным схемам параллельных вычислений;
- экспериментально подтвердить различия во временно́й эффективности линейной и параллельных реализаций на материале замеров времени выполнения реализации на квадратных матрицах переменных размерностей;
- описать и обосновать полученные результаты в отчете.

1 Аналитическая часть

В данном разделе будут изучены алгоритмы умножения матрицы и параллельные вычисления.

1.1 Определение матрицы и операции умножения матриц

Матрицей А 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 [4].

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

Пусть даны две прямоугольные матрицы А и В 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 [4] 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 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.

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

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.

1.3 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.

Для реализации алгоритма на параллельной системе его следует представить в виде последовательности групп операций, причем операции в каждой группе могут быть выполнены одновременно - 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.

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.

1.4 Вывод

В данном разделе были описаны цели и задачи лабораторной работы, изучены алгоритмы умножения матрицы и параллельные вычисления.

2 Конструкторская часть

2.1 Схемы алгоритмов

На рисунках 2.1, 2.2, 2.3 и 2.4 приведена схема алгоритма умножения матриц Винограда.

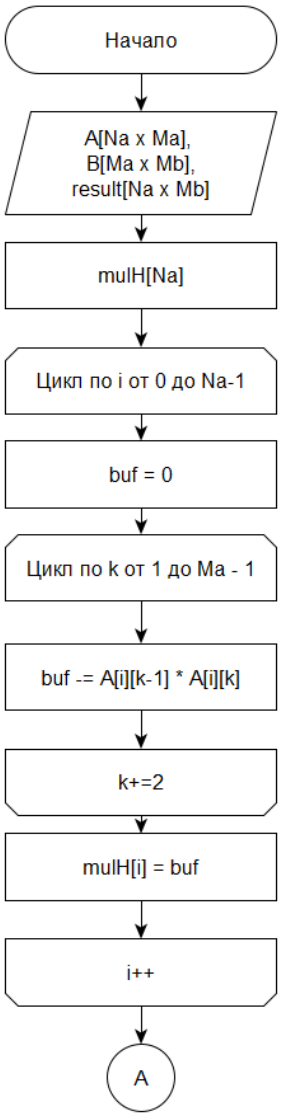

Рис. 2.1: Схема алгоритма Винограда умножения матриц, часть 1

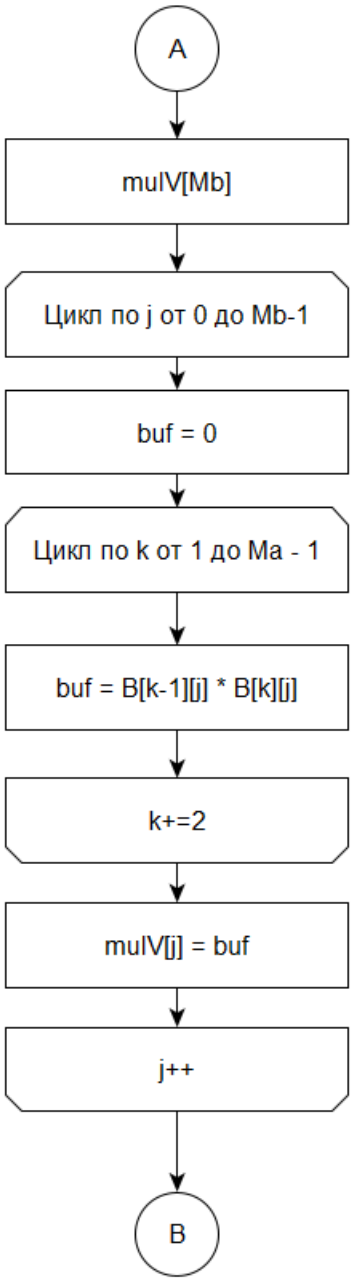

Рис. 2.2: Схема алгоритма Винограда умножения матриц, часть 2

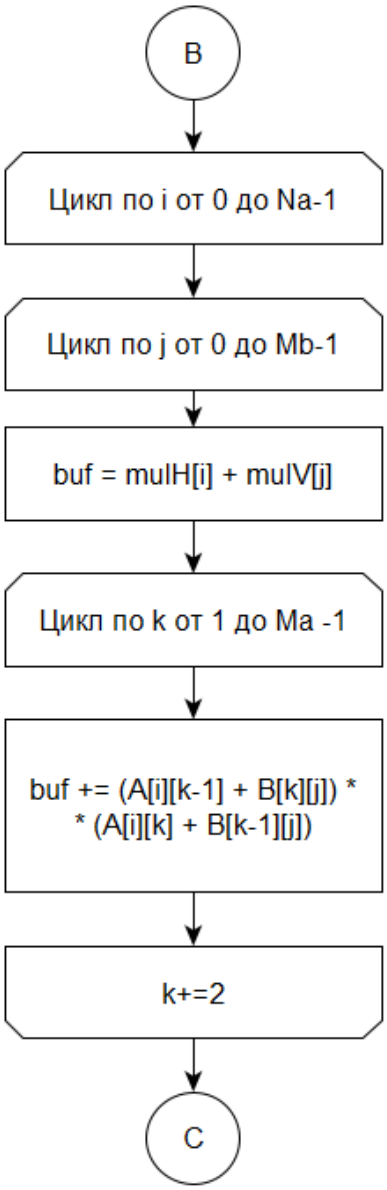

Рис. 2.3: Схема алгоритма Винограда умножения матриц, часть 3

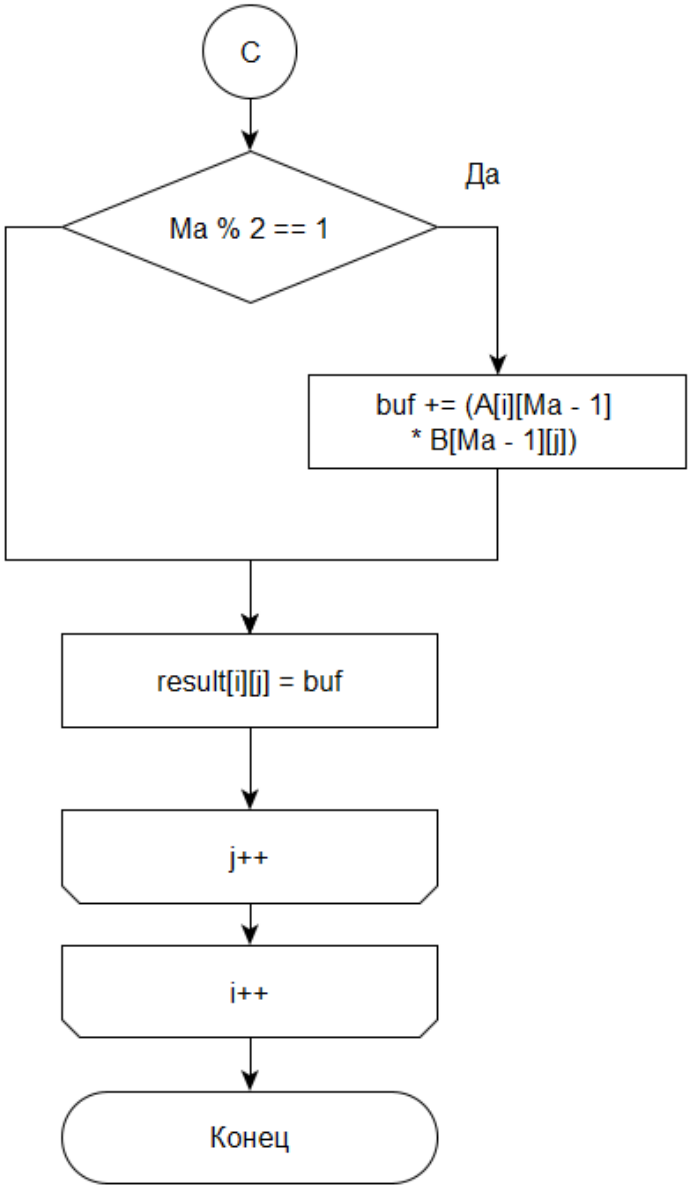

Рис. 2.4: Схема алгоритма Винограда умножения матриц, часть 4

2.2 Распараллеливание задачи

1. 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.

2. 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, 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.

2.3 Вывод

Была описана структура программного комплекса и схемы алгоритмов, так же проведены функциональные тесты работы программы, сделано предположение о том, какая из схем параллельных вычислений окажется эффективнее.

3 | Технологическая часть

В этом разделе будет обоснован выбор языка програмирования, описаны технические характеристики,приведены листинги кода реализованных алгоритмов.

3.1 Выбор ЯП

В качестве языка программирования мной был выбран C++ так как этот язык удобен для работы с потоками. Для замера времени выполнения использовалась функция $high\_resolution\_clock :: now()$.

3.2 Требование к ПО

- корректное умножение двух матриц;
- при матрицах неправилыных размеров программа не должна аварийно завершаться.

3.3 Реализация алгоритмов

В следующем разделе представлены листинги реализаций алгоритма Винограда. В листинге 3.1 представлена линейная реализация алгоритма умножения матриц Винограда. В листингах 3.2 и 3.3 представлены реализации алгоритма согласно разработанным схемам параллельных вычислений.

Листинг 3.1: Линейная реализация алгоритма Винограда умножения матриц

```
int vng_mult(int** A, int** B, int** result, int N_a, int M_a, int M_b)
{
  int* mulH = new int[N_a];
  int* mulV = new int[M_b];

  // PART 1
  int buf;
  for (int i = 0; i < N_a; i++)
  {
    buf = 0;
    for (int k = 1; k < M_a; k += 2)
    buf -= A[i][k - 1] * A[i][k];
    mulH[i] = buf;
  }

  // PART 2
  for (int j = 0; j < M_b; j++)
  {
    buf = 0;
    for (int k = 1; k < M_a; k += 2)
    buf -= B[k - 1][j] * B[k][j];
    mulV[j] = buf;
  }

  // PART 3 + 4
  for (int i = 0; i < N_a; i++)
  for (int j = 0; j < M_b; j++)
  {
    buf = mulH[i] + mulV[j];
    for (int k = 1; k < M_a; k += 2)
    buf += (A[i][k - 1] + B[k][j]) * (A[i][k] + B[k - 1][j]);

    if (M_a % 2)
    buf += A[i][M_a - 1] * B[M_a - 1][j];

    result[i][j] = buf;
  }

  return SUCCESS;
}
```

Листинг 3.2: caption

```
int vng_mult_par1(int** A, int** B, int** result, int N_a, int M_a, int
    M_b)
{
  int* mulH = new int[N_a];
  int* mulV = new int[M_b];
  std::thread* threads = new std::thread[t_count];

  if (t_count > 1)
  {
    int proportion = t_count * N_a / (N_a + M_b);
    int rows_t = (proportion) ? proportion : 1;
    int cols_t = t_count - rows_t;

    int rows_per_t = N_a / rows_t;
    int cols_per_t = M_b / cols_t;

    int start_row = 0;
    for (int i = 0; i < rows_t; i++)
    {
      int end_row = (i == rows_t - 1) ? N_a : start_row + rows_per_t;
      threads[i] = std::thread(scheme1_part1, A, N_a, M_a, mulH, start_row
          , end_row);
      start_row = end_row;
    }

    int start_col = 0;
    for (int i = rows_t; i < t_count; i++)
    {
      int end_col = (i == t_count - 1) ? M_b : start_col + cols_per_t;
      threads[i] = std::thread(scheme1_part2, B, M_a, M_b, mulV, start_col
          , end_col);
      start_col = end_col;
    }

    for (int i = 0; i < t_count; i++)
    {
      threads[i].join();
    }
  }
```

```
else
{
  int buf;
  for (int i = 0; i < N_a; i++)
  {
    buf = 0;
    for (int k = 1; k < M_a; k += 2)
    buf -= A[i][k - 1] * A[i][k];
    mulH[i] = buf;
  }

  for (int j = 0; j < M_b; j++)
  {
    buf = 0;
    for (int k = 1; k < M_a; k += 2)
    buf -= B[k - 1][j] * B[k][j];
    mulV[j] = buf;
  }
}

for (int i = 0; i < N_a; i++)
for (int j = 0; j < M_b; j++)
{
  buf = mulH[i] + mulV[j];
  for (int k = 1; k < M_a; k += 2)
  buf += (A[i][k - 1] + B[k][j]) * (A[i][k] + B[k - 1][j]);

  if (M_a % 2)
  buf += A[i][M_a - 1] * B[M_a - 1][j];

  result[i][j] = buf;
}

return SUCCESS;
}
}
```

```
void scheme1_part1(int** A, int N_a, int M_a, int* mulH, int start, int end)
{
  int buf;
  for (int i = start; i < end; i++)
  {
    buf = 0;
    for (int k = 1; k < M_a; k += 2)
    buf -= A[i][k - 1] * A[i][k];
    mulH[i] = buf;
  }
}

void scheme1_part2(int** B, int M_a, int M_b, int* mulV, int start, int end)
{
  int buf;
  for (int j = start; j < end; j++)
  {
    buf = 0;
    for (int k = 1; k < M_a; k += 2)
    buf -= B[k - 1][j] * B[k][j];
    mulV[j] = buf;
  }
```

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

```
int vng_mult_par2(int** A, int** B, int** result, int N_a, int M_a, int
    M_b, int t_count)
{
  int* mulH = new int[N_a];
  int* mulV = new int[M_b];
  std::thread* threads = new std::thread[t_count];

  int buf;
  for (int i = 0; i < N_a; i++)
  {
    buf = 0;
    for (int k = 1; k < M_a; k += 2)
    buf -= A[i][k - 1] * A[i][k];
    mulH[i] = buf;
  }

  for (int j = 0; j < M_b; j++)
  {
    buf = 0;
    for (int k = 1; k < M_a; k += 2)
    buf -= B[k - 1][j] * B[k][j];
    mulV[j] = buf;
  }

  int rows_per_t = N_a / t_count;
  int start_row = 0;
  for (int i = 0; i < t_count; i++)
  {
    int end_row = (i == t_count - 1) ? N_a : start_row + rows_per_t;
    threads[i] = std::thread(scheme_2, A, B, result, N_a, M_a, M_b, mulH,
        mulV, start_row, end_row);
    start_row = end_row;
  }

  for (int i = 0; i < t_count; i++)
  {
    threads[i].join();
  }

  return SUCCESS;
}
```

```
void scheme_2(int** A, int** B, int** result, int N_a, int M_a, int M_b, int
    * mulH, int* mulV, int start_row, int end_row)
{
  int buf;
  for (int i = 0; i < N_a; i++)
  for (int j = 0; j < M_b; j++)
  {
    buf = mulH[i] + mulV[j];
    for (int k = 1; k < M_a; k += 2)
    buf += (A[i][k - 1] + B[k][j]) * (A[i][k] + B[k - 1][j]);

    if (M_a % 2)
    buf += A[i][M_a - 1] * B[M_a - 1][j];

    result[i][j] = buf;
  }
}
```

3.4 Тестовые данные

Результаты тестирования программы приведены в таблице 3.1. В случае, если умножение матриц не допускается при проверке размеров, выводится соответствующее сообщение.

Таблица 3.1: Тесты для проверки корректности программы.

Матрица 1	**Матрица 2**	**Ожидаемый рез.**	**Фактический рез.**
$\begin{bmatrix}1&2&3\\2&3&4\\3&4&5\end{bmatrix}$	$\begin{bmatrix}2&3&4\\3&4&5\\4&5&6\end{bmatrix}$	$\begin{bmatrix}20&26&32\\29&38&47\\38&50&67\end{bmatrix}$	$\begin{bmatrix}20&26&32\\29&38&47\\38&50&67\end{bmatrix}$
$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$	$\begin{bmatrix}1&1&1\\1&1&1\end{bmatrix}$	Can't multiply matricies with such sizes	Can't multiply matricies with such sizes
$\begin{bmatrix}1&2\\2&4\\4&8\end{bmatrix}$	$\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}$	$\begin{bmatrix}9&12&15\\18&24&30\\39&48&60\end{bmatrix}$	$\begin{bmatrix}9&12&15\\18&24&30\\39&48&60\end{bmatrix}$

3.5 Вывод

В данном разделе был обоснован выбор языка програмирования и приведены листинги кода реализованных алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены демонстрация работы программы и исследование процессорного времени реализаций алгоритмов.

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

```
Menu:
1 - multiply
2 - measure time
0 - exit
Choice: 1
Input sizes of first matrix, space separated: 3 3
Input sizes of second matrix, space separated: 3 3

Input first matrix:
1 2 3
4 5 6
7 8 9

Input second matrix:
1 1 1
1 1 1
1 1 1

Result of multiplication with Linear Vinograd algorithm :
6 6 6
15 15 15
24 24 24

Result of multiplication with Parallel Vinograd algorithm - scheme 1 :
6 6 6
15 15 15
24 24 24

Result of multiplication with Parallel Vinograd algorithm - scheme 2 :
6 6 6
15 15 15
24 24 24
```

Рис. 4.1: Демонстрация работы программы

4.2 Время выполнения алгоритмов

В следующем разделе приведены результаты замеров времени выполнения реализаций на произвольно заполненных квадратных матрицах. Замеры времени произведены с помощью функции $high\_resolution\_clock :: now()$.

Было произведено две серии замеров:

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

Каждый замер времени был произведен на 100 итерациях, в качетсве результата взято время работы алгоритма, усредненное по всем итерациям.

Результаты замеров времени приведены в таблицах 4.1 и 4.2.

Таблица 4.1: Результаты первой серии замеров процессорного времени в мсек. для реализаций алгоритмов умножения матриц

N	Линейно	Паралл. схема 1	Паралл. схема 2
1	429	436	436
2	435	331	256
4	432	291	211
8	422	269	206
16	439	157	82
32	433	228	102

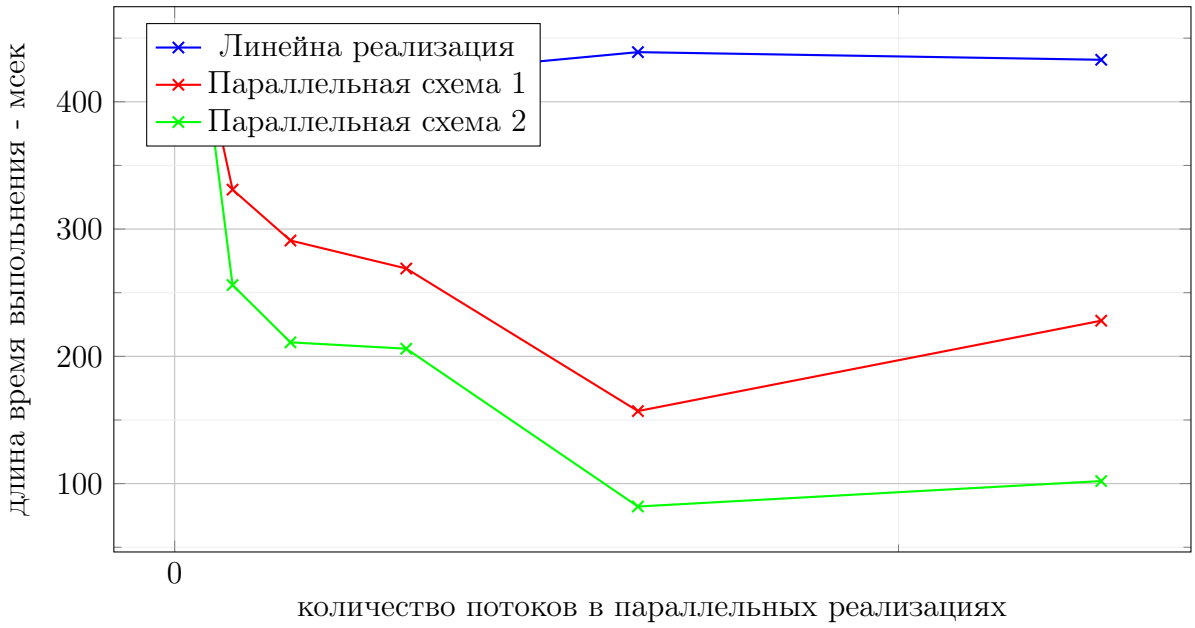

Рис. 4.2: Результаты первой серии замеров процессорного времени в мсек. для реализаций алгоритмов умножения матриц

Таблица 4.2: Результаты второй серии замеров процессорного времени в мсек. для реализаций алгоритмов умножения матриц

N	Линейно	Паралл. схема 1	Паралл. схема 2
100	5	26	32
200	45	132	122
300	152	166	158
400	323	232	178
500	437	269	207

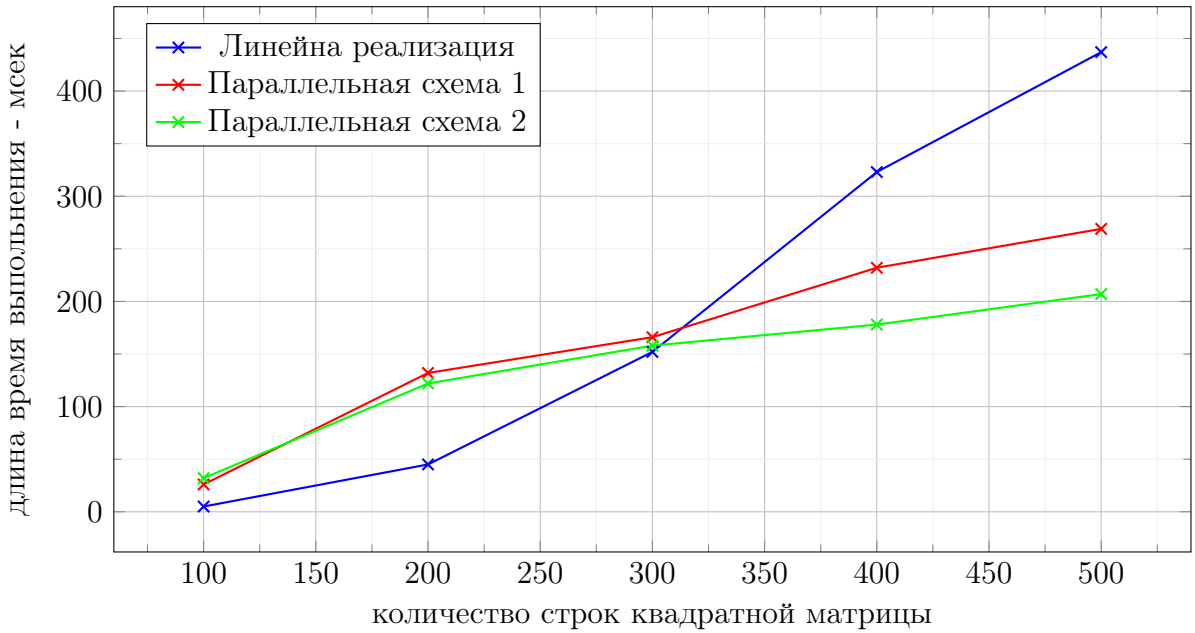

Рис. 4.3: Результаты второй серии замеров процессорного времени в мсек. для реализаций алгоритмов умножения матриц

Результаты проведенного эксперимента отвечают ожиданиям.

На входных матрицах небольших размерностей линейная реализация алгоритма работает быстрее, так как имеются накладные расходы на параллельные реализации:

- создание потоков;
- дополнительные вычисления;

Они превышают выгоду во времени на, непосредственно, умножение матриц. При увеличении размерностей входных матриц, выгода во времени на умножение параллельных реализаций превышает накладные расходы, и параллельные реализации выигрывают по времени над линейной. Это продемонстрировано на рисунке 4.3.

Причем, при увеличении размерностей входных данных, выгода по времени второй схемы параллельных вычислений превышает выгоду первой. 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. Это продемонстрировано на рисунке 4.3.

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 4.2.

4.3 Вывод

В данном разделе были приведены демонстрация работы программы и исследование процессорного времени реализаций алгоритмов, исходя из исследований сделаны выводы об эффективности выбранных схем распарраллеливания данной задачи.

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 материале замеров времени выполнения реализаций на квадратных матрицах;
- описаны и обоснованы полученные результаты в отчете.

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

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.

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 с увеличением числа потоков время исполнения параллельных реализаций уменьшается, однако по достижению 16 снова начинает увеличиваться.

Литература

[1] Демьянович Юрий Казимирович О параллельных вычислениях // КИО. 2007. №3. [Электронный ресурс.] URL: https://cyberleninka.ru/article/n/o-parallelnyh-vychisleniyah (дата обращения: 20.10.2021).

[2] Гладышев Е.И., Мурыгин А.В. Многопоточность в приложениях // Актуальные проблемы авиации и космонавтики. 2012. №8. [Электронный ресурс.] URL: https://cyberleninka.ru/article/n/mnogopotochnost-v-prilozheniyah (дата обращения: 20.10.2021).

[3] Антон Владимирович Желудков, Дмитрий Васильевич Макаров, Павел Владимирович Фадеев 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: 20.10.2021).

[4] Умножение матриц.[Электронный ресурс].Режим доступа: https://life-prog.ru/2_90314_umnozhenie-matrits.html (дата обращения: 20.10.2021)