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 _____ В.Г. Горячев
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва, 2020

Оглавление

Введение

Умножение матриц – одна из самых широко используемых операций в вычислениях, например, в таких областях, как физические расчёты и компьютерная графика, в которых произведение может требоваться с большой частотой. Одним из вариантов ускорения этой операции является улучшение самого алгоритма, таким является, например, оптимизированный алгоритм Винограда; другим - использование такой возможности современных компьютеров, как многопоточность, то есть разбивка последовательных действий на несколько параллельных. Это и будет рассматриваться в рамках настоящей лабораторной работы.

1 Аналитическая часть

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 линейной и параллельных реализаций по затраченному времени.

1.2 Описание и формулы

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы. Хотя исторически рассматривались, например, тре-

угольные матрицы, в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

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[3].

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:

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

Предыдущее 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 первой 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 расчета элементов результирующей матрицы сохраняется. После этого 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[2].

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.

Распространенный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью – создание новых параллель-

ных методов на основе обычных последовательных программ.

Для реализации алгоритма на параллельной основе 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.

1.3 Вывод

В этом разделе были сформулированы цели и задачи работы, даны математические описания матриц и алгоритмов умножения матриц, а также рассмотрены особенности применения параллельных потоков. В случае алгоритма умножения матриц Винограда основная идея заключается в том, чтобы распараллелить основной цикл тройной вложенности.

2 | Конструкторская часть

Входные данные - размеры умножаемых матриц и их содержимое - задаются из массива длин и формируются программно с помощью генератора случайных чисел соответственно. После вывода демонстрационного примера программа начинает проводить тестовые измерения скорости работы реализаций алгоритмов.

2.1 Требования к программному обеспечению

Программа должна выводить пример умножения матриц небольшого размера с целью контроля правильности работы алгоритмов, а также информацию об измерениях времени.

Для удобства пользователя могут выводиться проценты, по которым можно наглядно увидеть соотношение времени работы алгоритмов.

2.2 Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма умножения матриц Винограда, разделенная по частям.

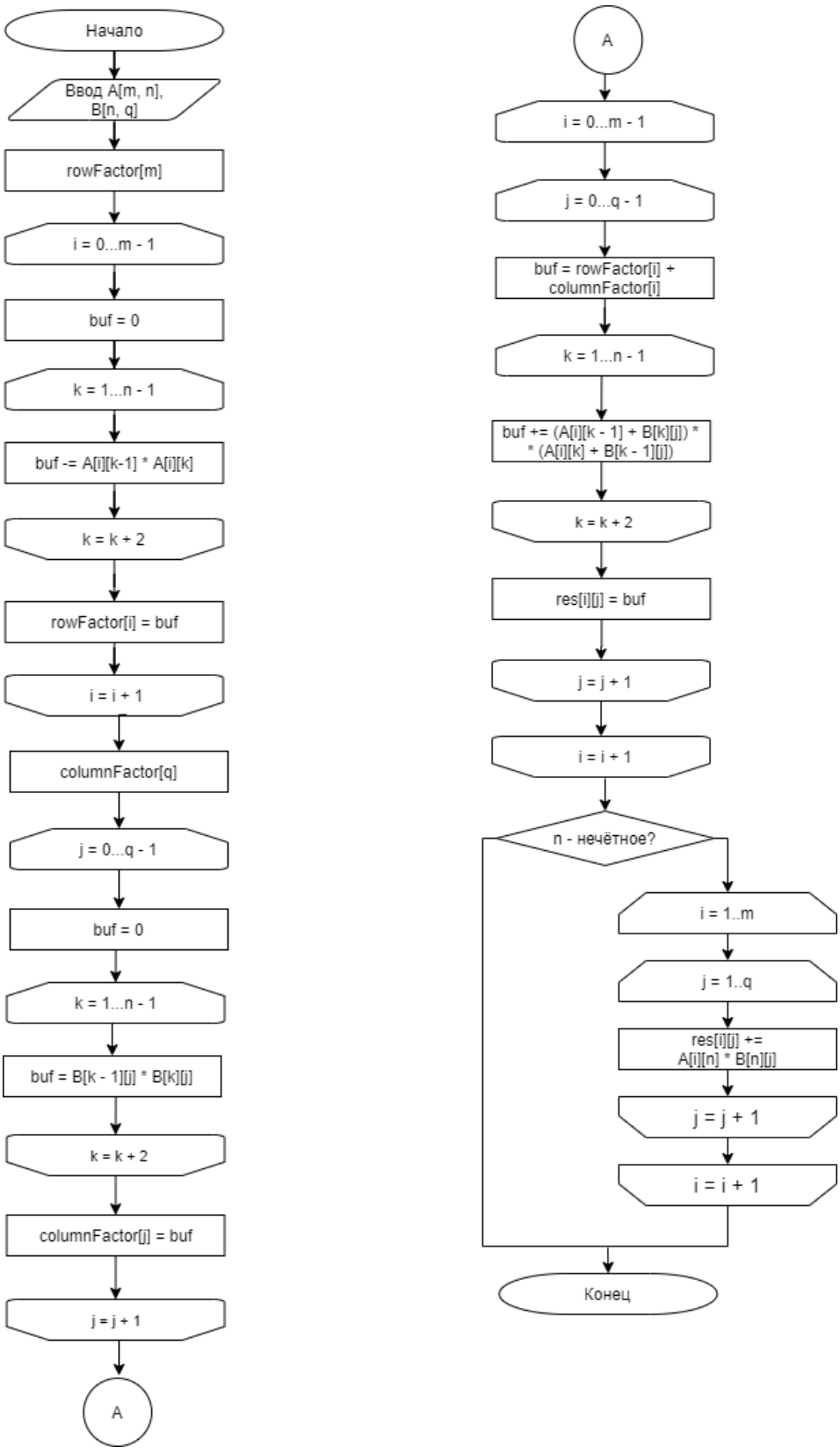


Рис. 2.1: Схема алгоритма умножения матриц Винограда

2.3 Схема параллельных вычислений

Оценивая возможность параллельных модификаций алгоритма Винограда, можно заметить, что он содержит два участка, где данные независимы друг от друга - при вычислениях векторов дополнительных слагаемых по строкам и столбцам и при вычислении элементов результирующей матрицы в итоговом тройном цикле. Из общего в обоих случаях только индексы матриц и массивов, что позволяет разделить операции в данных местах на потоки. Размеры векторов и матриц позволяют масштабировать разделение данных между потоками в зависимости от количества оных.

Наибольший практический смысл должно принести распараллеливание именно тройного цикла из-за количества повторений в нём.

2.4 Вывод

В ходе конструкторской части были сформулированы требования к программе, разработаны схемы алгоритмов, с помощью которых можно осуществить написание программы.

3 Технологическая часть

3.1 Выбор языка программирования

Для написания программной реализации алгоритма был выбран язык программирования C++ ввиду изучения оного на прошлых курсах, привычности в работе и быстродействия программ.

Среда разработки – Visual Studio 2019.

3.2 Реализация алгоритмов

В приводимых ниже листингах (3.1, 3.2 и 3.3) используется при подстановке в шаблоны тип Word – псевдоним для целочисленного типа short int, размер которого составляет 2 байта.

Листинг 3.1: Линейная реализация алгоритма умножения матриц Винограда

```
1  Matrix<Word> multiplicationVinogradOptimised(const Matrix<Word>& matA,
       const Matrix<Word>& matB) {
2      {   time_t _time = time(NULL);
3      if (!matA.isSuitForMult(matB))
4          throw MultiplicationException(__FILE__, typeid(matA).name(),
               __LINE__, ctime(&_time)); }
5
6      const size_t m = matA.rows(), q = matB.columns();
7      Matrix<Word> res(m, q);
8      size_t temp_size = (matA.columns() >> 1) << 1;
9      Matrix<Word>::MatrixLine rowFactor(m), columnFactor(q);
10     Word temp;
```

```
11
12      for (size_t i = 0; i < m; ++i) {
13          temp = 0;
14          for (size_t k = 0; k < temp_size; k += 2)
15              temp += (matA(i, k) * matA(i, k + 1));
16          rowFactor[i] = temp;
17      }
18      for (size_t i = 0; i < q; ++i) {
19          temp = 0;
20          for (size_t k = 0; k < temp_size; k += 2)
21              temp += matB(k, i) * matB(k + 1, i);
22          columnFactor[i] = temp;
23      }
24
25      for (size_t i = 0; i < m; ++i)
26          for (size_t j = 0; j < q; ++j) {
27              temp = -(rowFactor[i] + columnFactor[j]);
28              for (size_t k = 0; k < temp_size; k += 2) {
29                  temp += (matA(i, k) + matB(k + 1, j))
30                      * (matA(i, k + 1) + matB(k, j));
31              }
32              res(i, j) = temp;
33          }
34
35      if ((temp_size) != matA.columns())
36          for (size_t i = 0, t = temp_size; i < m; ++i)
37              for (size_t j = 0; j < q; ++j)
38                  res(i, j) += matA(i, t) * matB(t, j);
39      return res;
40  }
```

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

```
1  Matrix<Word> multiplicationVinogradParallel_1(const Matrix<Word>& matA,
       const Matrix<Word>& matB, size_t threadNum) {
2      {   time_t _time = time(NULL);
3      if (!matA.isSuitForMult(matB))
4          throw MultiplicationException(__FILE__, typeid(matA).name(),
```

```
        __LINE__, ctime(&_time)); }
5
6       const size_t m = matA.rows(), q = matB.columns();
7       Matrix<Word> res(m, q);
8       size_t temp_size = (matA.columns() >> 1) << 1;
9       Matrix<Word>::MatrixLine rowFactor(m), columnFactor(q);
10
11      shared_ptr<std::thread[]> threads = shared_ptr<std::thread[]>(new
           std::thread[threadNum]);
12
13      if (threadNum > 1) {
14          size_t proportion = threadNum * m / (m + q);
15          if (proportion == 0)
16              ++proportion;
17
18          size_t rows_per_t = m / proportion, cols_per_t = q / (threadNum
               - proportion);
19
20          size_t i = 0;
21          for (size_t start_row = 0, end_row; i < proportion; i++,
               start_row = end_row) {
22              end_row = (i == proportion - 1) ? m : start_row +
                   rows_per_t;
23              threads[i] = std::thread(&calcRow,
24                                       std::ref(rowFactor), std::ref(matA)
                                           , temp_size, start_row, end_row)
                                           ;
25          }
26
27          for (size_t start_col = 0, end_col; i < threadNum; ++i,
               start_col = end_col) {
28              end_col = (i == threadNum - 1) ? q : start_col + cols_per_t
                   ;
29              threads[i] = std::thread(&calcColumn,
30                                       std::ref(columnFactor), std::ref(
                                           matB), temp_size, start_col,
                                           end_col);
31          }
```

```
32
33          for (size_t i = 0; i < threadNum; ++i)
34              threads[i].join();
35
36      }
37      else {
38          calcRow(std::ref(rowFactor), std::ref(matA), temp_size, 0, m);
39          calcColumn(std::ref(columnFactor), std::ref(matB), temp_size,
                0, q);
40      }
41
42      for (size_t i = 0; i < m; ++i)
43          for (size_t j = 0; j < q; ++j) {
44              Word temp = -(rowFactor[i] + columnFactor[j]);
45              for (size_t k = 0; k < temp_size; k += 2) {
46                  temp += (matA(i, k) + matB(k + 1, j))
47                      * (matA(i, k + 1) + matB(k, j));
48              }
49              res(i, j) = temp;
50          }
51
52      if ((temp_size) != matA.columns())
53          for (size_t i = 0, t = temp_size; i < m; ++i)
54              for (size_t j = 0; j < q; ++j)
55                  res(i, j) += matA(i, t) * matB(t, j);
56
57      return res;
58  }
```

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

```
1  Matrix<Word> multiplicationVinogradParallel_2(const Matrix<Word>& matA,
       const Matrix<Word>& matB, size_t threadNum) {
2      {   time_t _time = time(NULL);
3      if (!matA.isSuitForMult(matB))
4          throw MultiplicationException(__FILE__, typeid(matA).name(),
               __LINE__, ctime(&_time)); }
5
```

```
6       const size_t m = matA.rows(), q = matB.columns();
7       Matrix<Word> res(m, q);
8       size_t temp_size = (matA.columns() >> 1) << 1;
9       Matrix<Word>::MatrixLine rowFactor(m), columnFactor(q);
10
11      calcRow(std::ref(rowFactor), std::ref(matA), temp_size, 0, m);
12      calcColumn(std::ref(columnFactor), std::ref(matB), temp_size, 0, q)
            ;
13
14      shared_ptr<std::thread[]> threads = shared_ptr<std::thread[]>(new
            std::thread[threadNum]);
15
16      size_t rows_per_t = m / threadNum;
17
18      for (size_t i = 0, start_row = 0, end_row; i < threadNum; ++i,
            start_row = end_row) {
19          end_row = (i == threadNum - 1) ? m : start_row + rows_per_t;
20          threads[i] = std::thread(&heartOfVinograd, std::ref(matA), std
                ::ref(matB), std::ref(res),
21                  std::ref(rowFactor), std::ref(columnFactor), temp_size,
                        start_row, end_row);
22      }
23
24      for (int i = 0; i < threadNum; ++i)
25          threads[i].join();
26
27      if ((temp_size) != matA.columns())
28          for (size_t i = 0, t = temp_size; i < m; ++i)
29              for (size_t j = 0; j < q; ++j)
30                  res(i, j) += matA(i, t) * matB(t, j);
31
32      return res;
33  }
```

3.3 Тестирование

Программа не подразумевает ручного ввода, что отбрасывает ситуации с заведомо ошибочными данными. Правильность расчётов можно отследить, сравнивая результаты между собой и с самостоятельно полученным расчётом по выведенным исходным данным. Пример такой проверки можно увидеть в таблице 3.1.

Таблица 3.1: Результаты тестирования программы

Исходн. матр.	Ожидается	Линейная реализация	Пар. схема 1	Пар. схема 2
Чётное N				
6 1 2 11 14 7 * 4 1 7 2 6 4 13 7	30 10 55 19 74 46 157 81 98 42 189 77	30 10 55 19 74 46 157 81 98 42 189 77	30 10 55 19 74 46 157 81 98 42 189 77	30 10 55 19 74 46 157 81 98 42 189 77
Нечётное N				
0 1 3 12 3 2 3 0 1 7 11 3 10 7 3 * 2 0 11 14 0 0 5 6 1 2	74 92 49 62 93 90	74 92 49 62 93 90	74 92 49 62 93 90	74 92 49 62 93 90

Все результаты, полученные в ходе тестирования, соответствуют ожидаемым.

3.4 Вывод

В ходе технологической части были успешно реализованы и протестированы требуемые вариации алгоритма умножения матриц Винограда, что позволит

произвести измерения времени работы и перейти к следующей части.

4 | Исследовательская часть

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Для удобства берутся две квадратные матрицы одинаковой размерности, заполненные произвольными данными. Было произведено две серии замеров:

- для матриц размером $N \times N = 360$ при переменном числе потоков для параллельных реализаций, число потоков $T \in \{1, 2, 6, 12, 24, 48\}$;
- для матриц размером $N \times N$, $N \in \{10, 100, 200, 300, 400, 500\}$ с постоянным числом потоков для параллельных реализаций, равным 12.

Технические характеристики устройства, на котором проводились замеры времени:

- операционная система Windows 10 64-bit;
- память 16 ГБ, DDR4, 2987 МГц;
- процессор AMD Ryzen 5 3600 6-Core Processor @ 3,6-4,2 Ггц;
- 12 логических процессоров.

Быстродействие алгоритмов измеряется в процессорном времени, которое более точно отражает время работы алгоритмов, с помощью функции _rdtsc()[1].

Каждый замер времени был произведен на 25 итерациях, в качестве результата взято время работы алгоритма, усредненное по всем итерациям.

Всё время в таблицах разделено на 10^5 - чтобы упростить и сократить запись.

Вследствие первой серии замеров были получены данные, представленные в табл. 4.1.

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

T	Линейно	Паралл. схема 1	Паралл. схема 2
1	67696.733	67942.876	67704.667
2	67732.056	67868.496	34293.450
6	67684.530	67828.526	12657.820
12	67792.655	68036.095	7706.102
24	67677.588	68176.124	8450.623
48	67635.709	68596.229	12435.956

Результаты второй серии замеров отражены в табл. 4.2.

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

N	Линейно	Паралл. схема 1	Паралл. схема 2
10	1.932	229.493	228.652
100	1471.993	1716.114	508.790
200	11591.301	11805.451	1983.097
300	39135.849	39343.261	4640.958
400	92982.661	93307.789	10854.679
500	181393.747	182054.944	21347.212

Полученные данные удобно воспринимать в виде графиков, данных на рисунках 4.1 и 4.2.

Рис. 4.1: Результаты первой серии замеров процессорного времени для реализаций алгоритмов умножения матриц

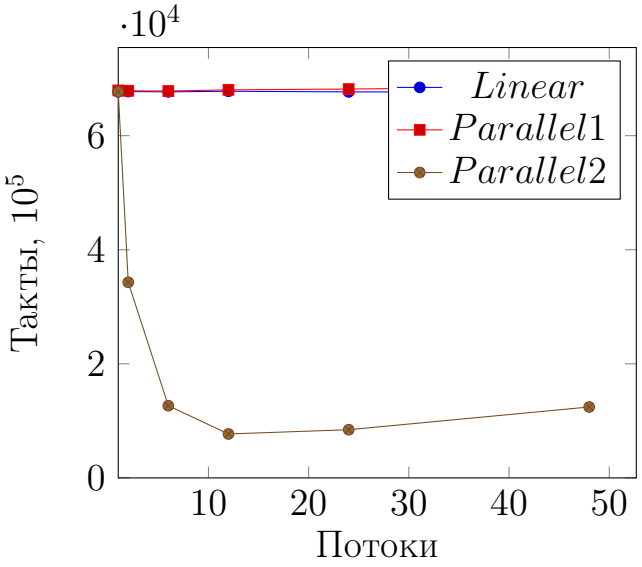
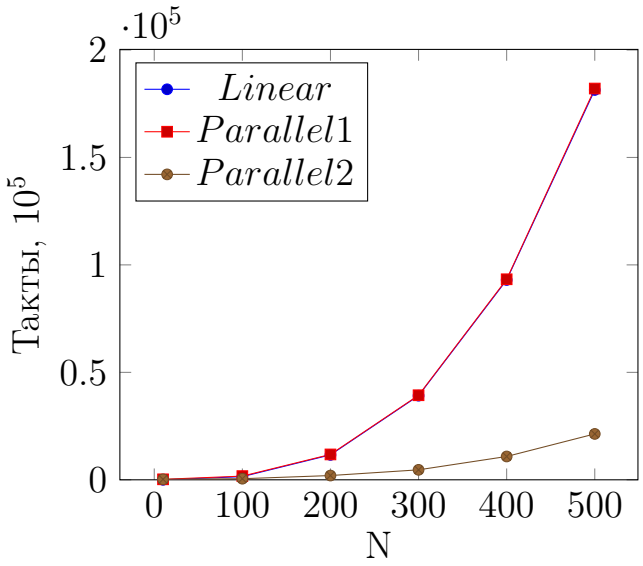


Рис. 4.2: Результаты второй серии замеров процессорного времени для реализаций алгоритмов умножения матриц, N — количество строк входных квадратных матриц



4.2 Вывод

Результаты тестирования показывают, что уже при 6 и, достоверно, вплоть до 48 потоках вторая параллельная реализация алгоритма Винограда значительно опережает по скорости выполнения остальные функции, однако наи-

большая эффективность приходится на число 12 - число потоков процессора, на котором проводились измерения. В таком случае время на параллельное вычисление решения будет составлять около 11-16% - чем больше число элементов, тем сильнее заметно преимущество алгоритма. В то же время первый вариант распараллеливания оказался неудачным, поскольку не даёт никакого выигрыша по времени (и даже незначительно, но уступает) в сравнении с линейной реализацией алгоритма при любых числе потоков и размере матриц.

Заключение

В ходе лабораторной работы были реализованы и изучены различные варианты распараллеливания алгоритма умножения матриц, проведено экспериментальное сравнение по затраченному времени. Проведённое исследование позволяет сделать вывод, что вторая параллельная реализация алгоритма Винограда по быстродействию существенно, почти в 10 раз (если вычисляются произведения больших матриц), опережает другие варианты, если количество используемых программных потоков близко к числу потоков процессора.

Другим важным выводом является тот факт, что не всегда параллельность оправдывает себя: другой вариант разбиения алгоритма не дал никакого полезного эффекта в вычислениях, хотя код при этом был усложнён и увеличен в размерах. Не ставилось задачи изучения затрачиваемой памяти, однако из общих знаний, полученных в рамках обучения, известно, что создание новых потоков требует времени и памяти из-за копирования данных. Информацию, полученную от потоков, необходимо синхронизировать. Всё это ограничивает применение многопоточности.

Исходя из результатов лабораторной работы, есть смысл создавать параллельные версии алгоритмов только тогда, когда это даёт существенный выигрыш в производительности. В противном случае лучше и необходимо применять линейные программы.

Список литературы

1. Функция _rdtsc() – документация Майкрософт [эл. ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/intrinsics/rdtsc?view=vs-2019 (дата обращения: 27.09.2020).

2. Дж. Макконнелл "Основы современных алгоритмов. 2-е дополненное издание". (дата обращения: 20.10.2020).

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