МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Н. Э. БАУМАНА

УДК _____
№ госрегистрации _____
Инв. №_____

УТВЕРЖДАЮ

Преподаватель

_____

«_____» _____ 2020 г.

ДИСЦИПЛИНА АНАЛИЗ АЛГОРИТМОВ

ОТЧЁТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Распараллеливание вычислений для алгоритма Винограда
(промежуточный)

Студент _____ Ф.М. Набиев

Преподаватели Л.Л. Волкова, Ю.В. Строганов

Москва, 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Умножение матриц является основным инструментом линейной алгебры и имеет многочисленные применения в математике, физике, программировании [1].

В данной лабораторной работе ставятся следующие задачи:

— изучение распараллеливания вычислений и работа с потоками;

— реализация распараллеленных вычислений;

— экспериментальное сравнение работы алгоритма на разном количестве потоков.

1 Аналитический раздел

В данном разделе будет определена теоретическая база, необходимая для реализации поставленных задач.

1.1 Описание задачи

В соответствии с книгой [2] дадим определение произведения двух матриц. Пусть даны прямоугольные матрицы A и B. Размеры этих матриц $n \times r$ и $r \times m$ соответственно. Тогда результатом умножения матрицы A на матрицу B называется такая матрица C размера $n \times m$, что:

$$c_{ij} = \sum_{k=1}^{r}(a_{ik} \cdot b_{kj}) \tag{1.1}$$

где $i = \overline{1, n}$, $j = \overline{1, m}$.

Также важно заметить, что вычисление каждого нового элемента результирующей матрицы не влияет на вычисление следующих, то есть каждый элемент матрицы считается отдельно. Значит, можно произвести распараллеливание вычислений и, тем самым, ускорить их.

1.1.1 Алгоритм Винограда

Данный алгоритм представляет собой альтернативный способ умножения матриц, позволяющий уменьшить количество операций умножения при вычислениях.

В качестве примера рассмотрим два вектора длинной 4: $U = (u_1, u_2, u_3, u_4)$ и $V = (v_1, v_2, v_3, v_4)$. Их скалярное произведение равно:

$$U \times V = u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3 + u_3 \cdot v_4 \tag{1.2}$$

Левую часть этого равенства можно записать в виде:

$$(u_1 + v_2) \cdot (u_2 + v_1) + (u_3 + v_4) \cdot (u_4 + v_3) - u_1 \cdot u_2 - u_3 \cdot u_4 - v_1 \cdot v_2 - v_3 \cdot v_4 \tag{1.3}$$

Это выражение допускает предварительную обработку в случае умножения двух матриц. Для каждой строки можно вычислить выражение:

$$u_1 \cdot u_2 + u_3 \cdot u_4 \tag{1.4}$$

А для каждого столбца:

$$v_1 \cdot v_2 + v_3 \cdot v_4 \tag{1.5}$$

Таким образом, выходит, что над заранее обработанными данными необходимо выполнить лишь 2 умножения, 5 сложений и 2 вычитания. Данная логика применима и для общего случая умножения матриц.

1.2 Вывод

Умножение матриц необходимый инструмент, для которого есть пути ускорения вычислений за счет уменьшения доли умножения и распараллеливания вычислений.

2 Конструкторский раздел

В данном разделе будет произведена разработка распараллеленного алгоритма Винограда.

2.1 Функциональная модель

На рисунке 2.1 представлена функциональная модель IDEF0 уроня 1.

Рисунок 2.1 — функциональная модель IDEF0 уроня 1

2.2 Разработка алгоритмов

Изобразим схемы алгоритма Винограда и распараллеленного алгоритма Винограда.

2.2.1 Алгоритм Винограда

На риснуках 2.2, 2.3 изображена схема алгоритма Винограда.

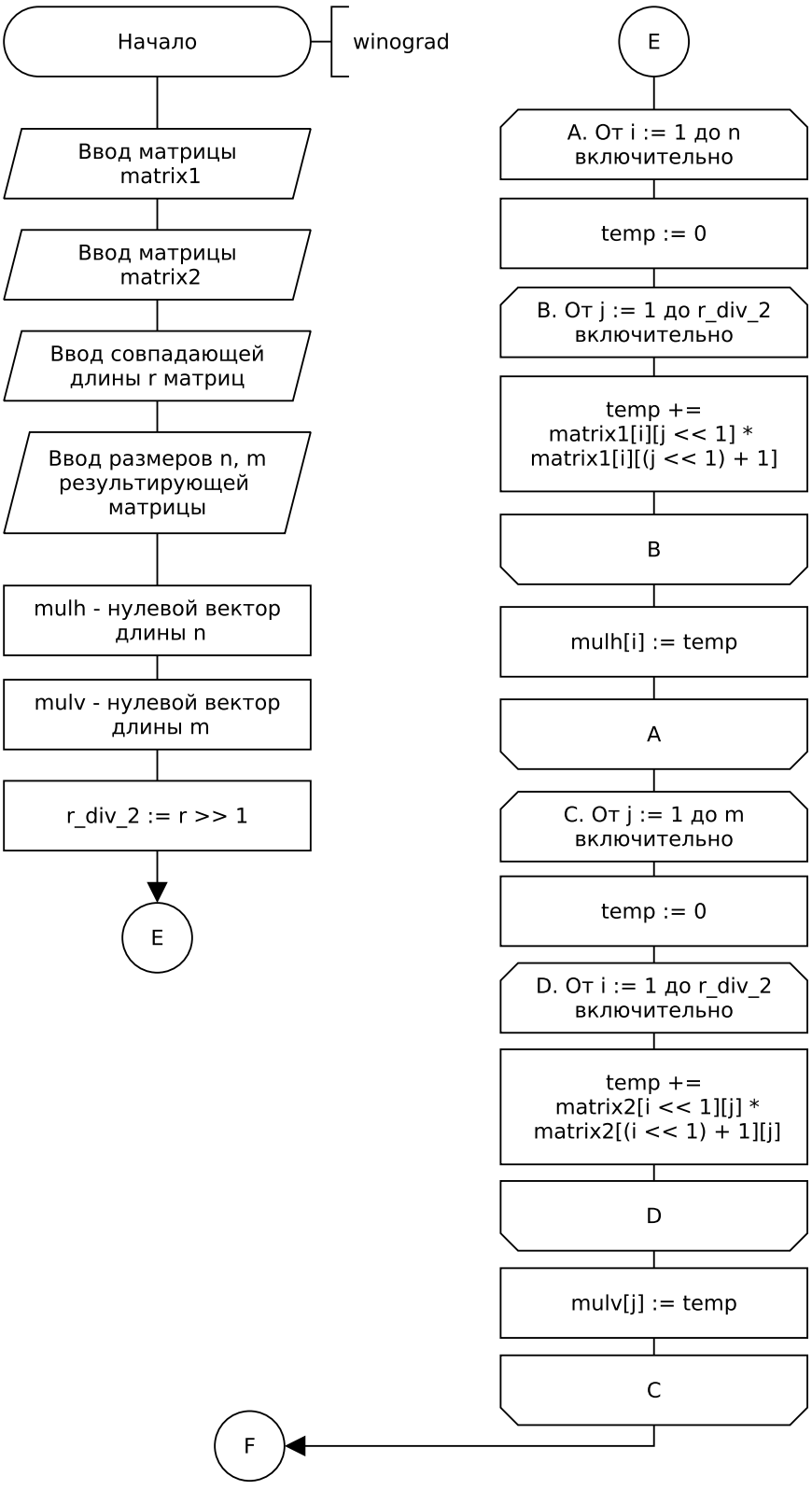

Рисунок 2.2 — Модифицированный алгоритм Винограда, часть 1

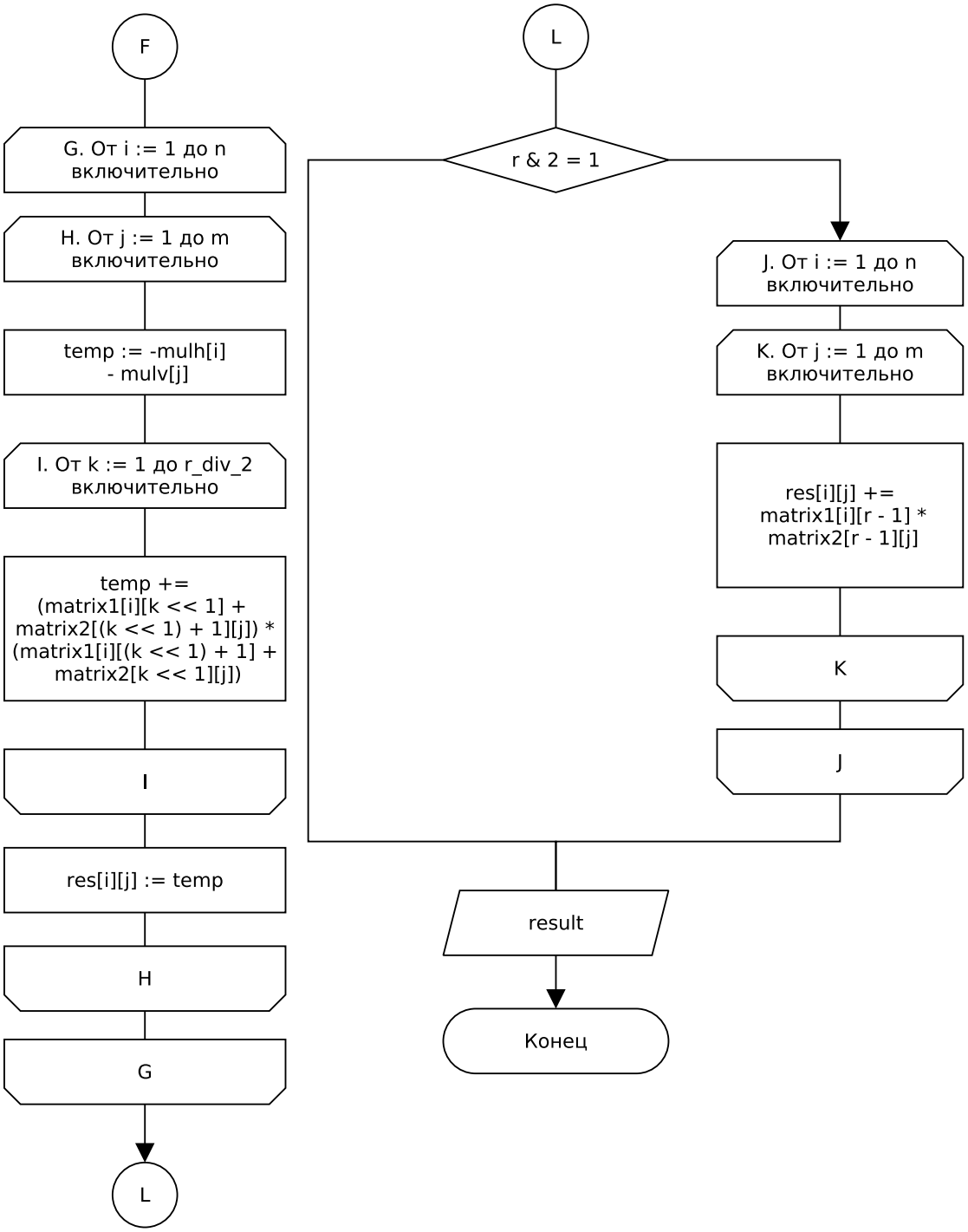

Рисунок 2.3 — Модифицированный алгоритм Винограда, часть 2

2.2.2 Параллелизированный алгоритм Винограда

Пусть, даны две матрицы: матрица 1 и матрица 2. Разобьём матрицу 1 построчно на столько подматриц, сколько потоков используется при распараллеливании. Тогда результат умножения матрицы 1 на матрицу 2

будет равен объединению произведений подматриц матрицы 1 на матрицу 2 в порядке их разбиения. На рисунке 2.4 изображён параллелизированный алгоритм Винограда, основанный на обычном.

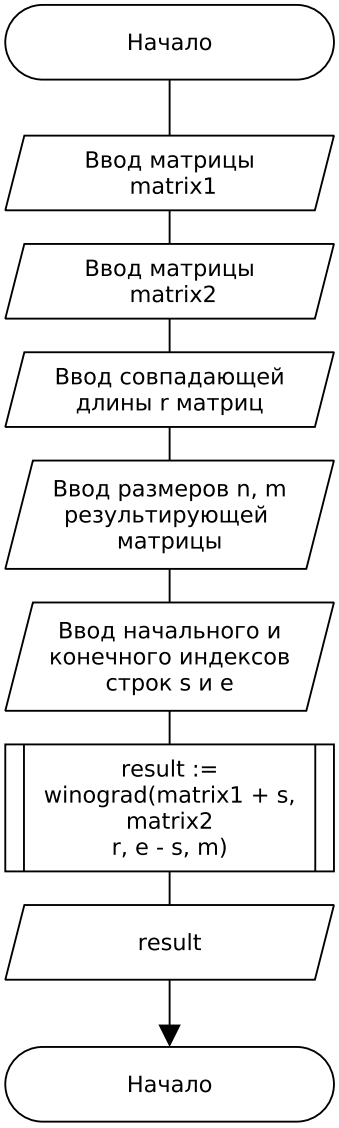

Рисунок 2.4 — Параллелизированный алгоритм Винограда

2.3 Вывод

Благодаря тому, что вычисление каждого элемента матрицы произведения независимо от остальных, удалось разработать параллелизированный алгоритм Винограда.

3 Технологический раздел

В данном разделе будут составлены требования к программному обеспечению, выбраны средства реализации и определены тестовые данные.

3.1 Требования к программному обеспечению

Требования к вводу:

— $p \in \mathbb{N}$ - число потоков;

— $n \in \mathbb{N}$ - размер перемножаемых квадратных матриц;

— $n \cdot n$ действительных чисел, разделенных символами-разделителями (пробел, перенос строки, табуляция и т.п.) - элементы первой матрицы, перечисленные построчно;

— $n \cdot n$ действительных чисел, разделенных символами-разделителями - элементы второй матрицы, перечисленные построчно.

Требования к выводу:

— результат умножения матриц.

3.2 Средства реализации

Для реализации программы вычисления редакционного расстояния мной был выбран язык программирования C++. В рамках текущей задачи данный язык программирования имеет ряд существенных преимуществ:

— Статическая типизация;

— Близость к низкоуровневому C при наличии многих возможностей высокоуровненных языков;

— Встроенная библиотека std::chrono, позволяющая измерять процессорное время [3].

— Встроенная библиотека std::thread, содержащая класс нативных потоков [4].

3.3 Листинги кода

Листинг 3.1 — Алгоритм Винограда

```
1  void MatrixMul(double **res, double **matrix1, double **matrix2,
2                 size_t n, size_t m, size_t r)
3  {
4      double temp;
5
6      double *mulh = new double[n]();
7      double *mulv = new double[m]();
8
9      double r_div_2 = r >> 1;
10
11     for (size_t i = 0; i < n; i++)
12     {
13         temp = 0;
14
15         for (size_t j = 0; j < r_div_2; j++)
16         {
17             temp += matrix1[i][j << 1] * matrix1[i][(j << 1) + 1];
18         }
19
20         mulh[i] = temp;
21     }
22
23     for (size_t j = 0; j < m; j++)
24     {
25         temp = 0;
26
27         for (size_t i = 0; i < r_div_2; i++)
28         {
29             temp += matrix2[i << 1][j] * matrix2[(i << 1) + 1][j];
30         }
31
32         mulv[j] = temp;
33     }
34
35     for (size_t i = 0; i < n; i++)
36     {
37         for (size_t j = 0; j < m; j++)
38         {
39             temp = -(mulh[i] + mulv[j]);
40
41             for (size_t k = 0; k < r_div_2; k++)
```

```
42              {
43                  temp += (matrix1[i][k << 1] + matrix2[(k << 1) + 1][j]) *
44                          (matrix1[i][(k << 1) + 1] + matrix2[k << 1][j]);
45              }
46
47              res[i][j] = temp;
48          }
49      }
50
51      if (r & 1)
52      {
53          for (size_t i = 0; i < n; i++)
54              for (size_t j = 0; j < m; j++)
55                  res[i][j] += matrix1[i][r - 1] * matrix2[r - 1][j];
56      }
57
58      delete[] mulh;
59      delete[] mulv;
60  }
```

Листинг 3.2 — Параллелизированный алгоритм Винограда

```
1  struct MatrixDescriptor
2  {
3      int n;
4      int m;
5      double **data;
6  };
7
8  class Multythread
9  {
10 public:
11     static
12     void MatrixMul(MatrixDescriptor &res,
13                    MatrixDescriptor &md1,
14                    MatrixDescriptor &md2,
15                    size_t start, size_t end)
16     {
17         ::MatrixMul(res.data + start,
18                     md1.data + start,
19                     md2.data,
20                     end - start,
21                     md2.m, md1.m);
22     }
23 };
```

3.4 Примеры работы

На рисунках 3.1-3.5 приведены примеры работы.

```
# ./fn_aa_lab_04
```

Рисунок 3.1 — Пустой ввод

```
 # ./fn_aa_lab_04
0
```

Рисунок 3.2 — Невозможное кол-во потоков

```
 # ./fn_aa_lab_04
5 4
```

Рисунок 3.3 — Длина матрицы меньше кол-ва потоков

```
 # echo 1 $(echo 4 | ./gen) | ./fn_aa_lab_04
Threads quantity: 1
Matrix len: 4
13.000  5.000 11.000  2.000
 7.000 19.000 12.000 15.000
 7.000  2.000  1.000  0.000
 3.000 18.000 15.000  7.000

 7.000 17.000 19.000 15.000
18.000 14.000 19.000 14.000
 5.000 14.000 11.000 13.000
16.000  1.000  5.000  8.000

268.000 447.000 473.000 424.000
691.000 568.000 701.000 647.000
90.000 161.000 182.000 146.000
532.000 520.000 599.000 548.000
```

Рисунок 3.4 — Работа одного потока на матрицах длины 4

```
 # echo 8 $(echo 10 | ./gen) | ./fn_aa_lab_04
Threads quantity: 8
Matrix len: 10
 8.000  7.000  0.000 18.000 17.000  7.000 11.000  2.000  9.000 11.000
10.000 16.000 19.000  3.000 10.000  0.000  3.000 13.000 11.000 13.000
14.000 13.000  3.000 11.000 14.000  9.000  1.000 14.000 19.000 11.000
 2.000  8.000 11.000 12.000  6.000 17.000  5.000  3.000 19.000 14.000
 2.000 11.000  1.000 14.000 16.000  0.000 14.000 14.000  1.000 19.000
 5.000 13.000 18.000 10.000  3.000 15.000 16.000  1.000 15.000  6.000
15.000 13.000 13.000  4.000  7.000  7.000 11.000 19.000  5.000 13.000
13.000  9.000  6.000  8.000 15.000  8.000  5.000  3.000 19.000  2.000
 4.000 17.000 13.000  2.000  0.000 16.000  5.000 18.000 17.000  9.000
 4.000 17.000  5.000 19.000 11.000  5.000 18.000  5.000 18.000 19.000

 7.000  2.000 16.000 13.000  5.000 14.000 10.000  3.000  6.000  6.000
 2.000  1.000 18.000  4.000 12.000 16.000  6.000  7.000  0.000  8.000
 3.000  5.000  0.000 11.000  4.000  9.000  0.000  8.000  1.000 13.000
 8.000  2.000 12.000  4.000 13.000 18.000  0.000  1.000  1.000 14.000
 9.000  0.000  9.000 16.000 10.000  0.000 15.000 16.000 13.000  2.000
 4.000  8.000 13.000 17.000  2.000  8.000  1.000 12.000 13.000  6.000
 5.000  4.000 10.000 12.000  7.000 16.000 14.000  2.000 11.000  3.000
 8.000 12.000  8.000  5.000 11.000 13.000 12.000  3.000 12.000 15.000
 1.000 18.000 13.000 13.000 15.000 10.000 17.000  4.000 15.000  4.000
14.000 19.000  1.000 18.000 14.000  0.000 11.000 18.000 18.000 19.000

629.000 554.000 968.000 1052.000 930.000 896.000 836.000 709.000 856.000 740.000
585.000 750.000 864.000 1053.000 968.000 948.000 874.000 780.000 800.000 992.000
673.000 873.000 1213.000 1215.000 1139.000 1089.000 1063.000 818.000 1059.000 961.000
545.000 891.000 930.000 1186.000 949.000 916.000 758.000 809.000 962.000 903.000
744.000 651.000 826.000 986.000 1021.000 881.000 916.000 777.000 914.000 958.000
469.000 713.000 1025.000 1158.000 875.000 1159.000 745.000 691.000 813.000 841.000
687.000 781.000 1016.000 1163.000 962.000 1136.000 950.000 773.000 947.000 1060.000
454.000 581.000 1028.000 1037.000 848.000 897.000 868.000 625.000 803.000 592.000
493.000 935.000 1026.000 1076.000 948.000 1093.000 832.000 723.000 935.000 977.000
762.000 945.000 1235.000 1329.000 1328.000 1288.000 1139.000 891.000 1126.000 1105.000
```

Рисунок 3.5 — Работа восьми потоков на матрицах длины 10

3.5 Описание тестирования

В таблице 3.1 приведены тестовые данные.

Таблица 3.1 — Тестовые данные

Первая матрица	Вторая матрица	Ожидаемый результат
1 2 3 4	1 2 3 4	7 10 15 22
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	30 36 42 66 81 96 102 126 150
1 0 0 0 1 0 0 0 1	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9

В таблицах 3.2, 3.3 приведены результаты тестирования. Все тесты были успешно пройдены.

Таблица 3.2 — Результаты тестирования алгоритма Винограда

Первая матрица	Вторая матрица	Полученный результат
1 2 3 4	1 2 3 4	7 10 15 22
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	30 36 42 66 81 96 102 126 150
1 0 0 0 1 0 0 0 1	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9

Таблица 3.3 — Результаты тестирования параллелизированного алгоритма Винограда

Первая матрица	Вторая матрица	Полученный результат
1 2 3 4	1 2 3 4	7 10 15 22
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	30 36 42 66 81 96 102 126 150
1 0 0 0 1 0 0 0 1	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9

3.6 Вывод

Для реализации программы были выбраны средства разработки: язык программирование C++, библиотеки std::chrono, std::thread, а так же подготовлены тестовые данные и успешно проведено тестирование.

4 Исследовательский раздел

В данном разделе будут производиться эксперементы над корректно реализованной программой.

4.1 Эксперименты по замеру времени

На рисунках 4.1, 4.2 приведены графики сравнения непараллелизированной реализации алгоритма Винограда и параллелизированной на разном кол-ве потоков для чётных длин матриц, на 4.3, 4.4 - для нечётных.

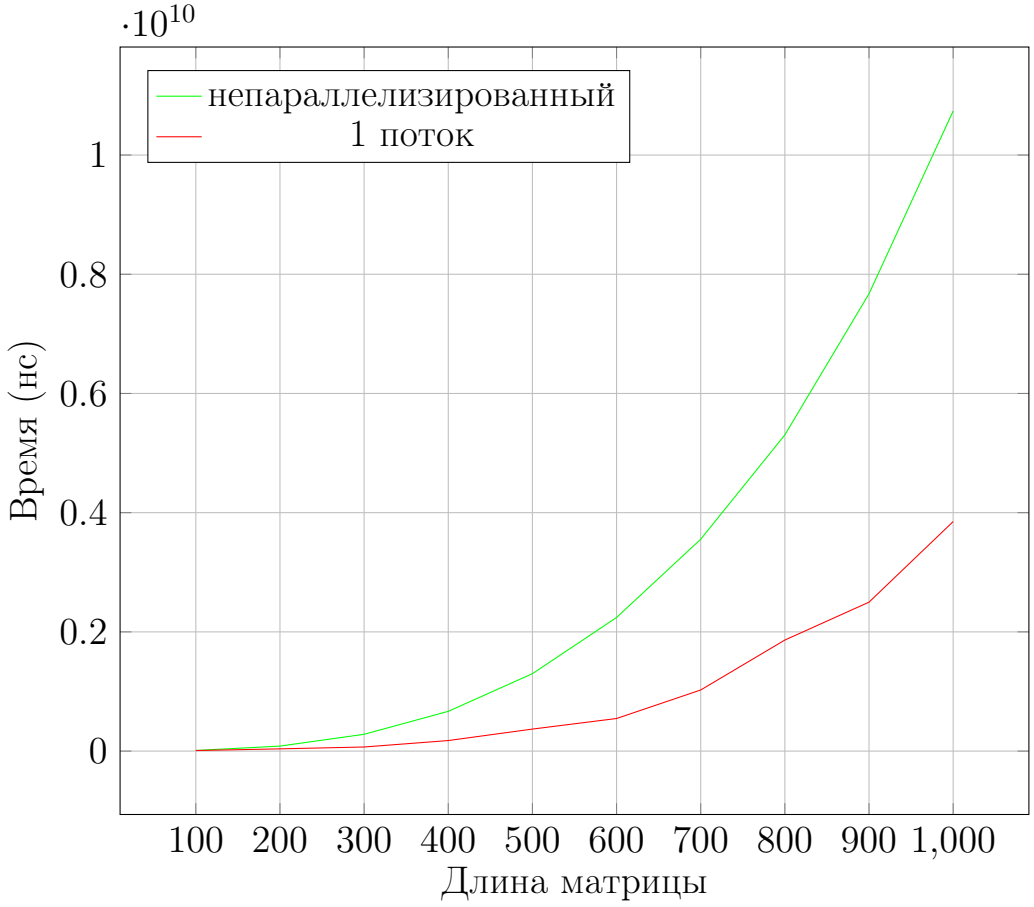

Рисунок 4.1 — Чётная длина, сравнение параллельных и последовательных вычислений

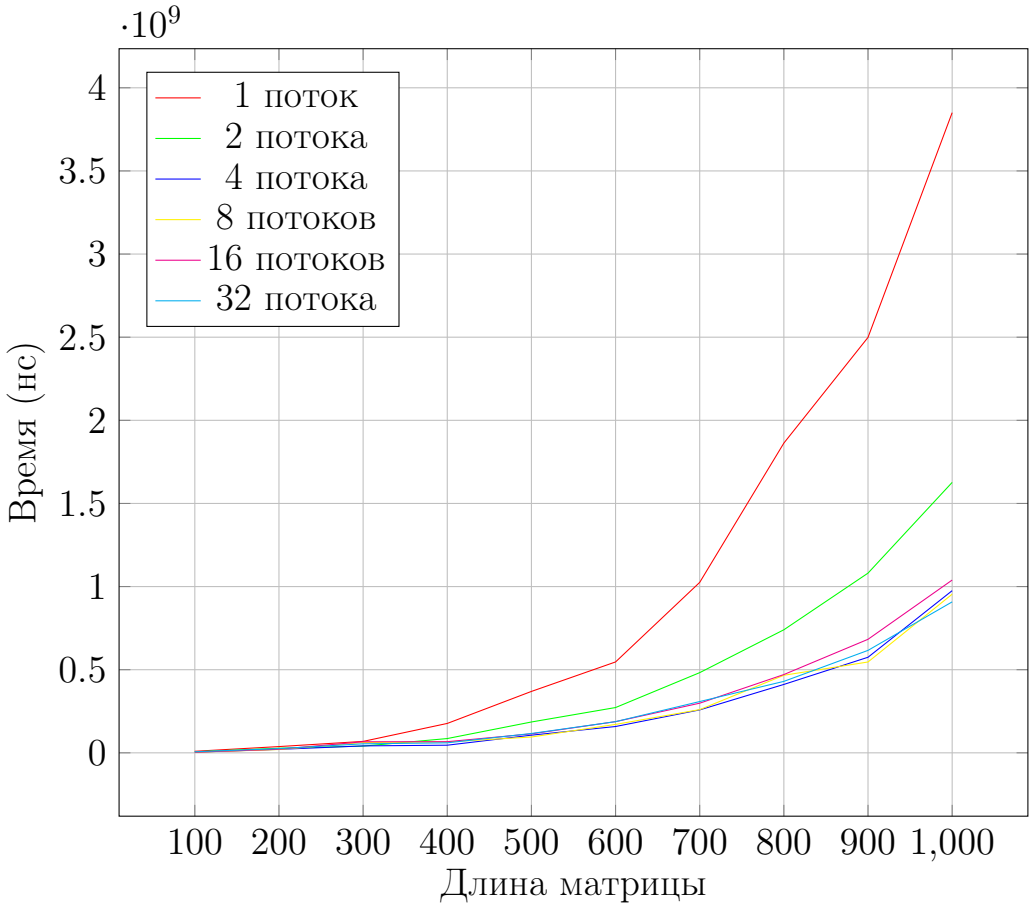

Рисунок 4.2 — Чётная длина, сравнение параллельных вычислений при разном кол-ве потоков

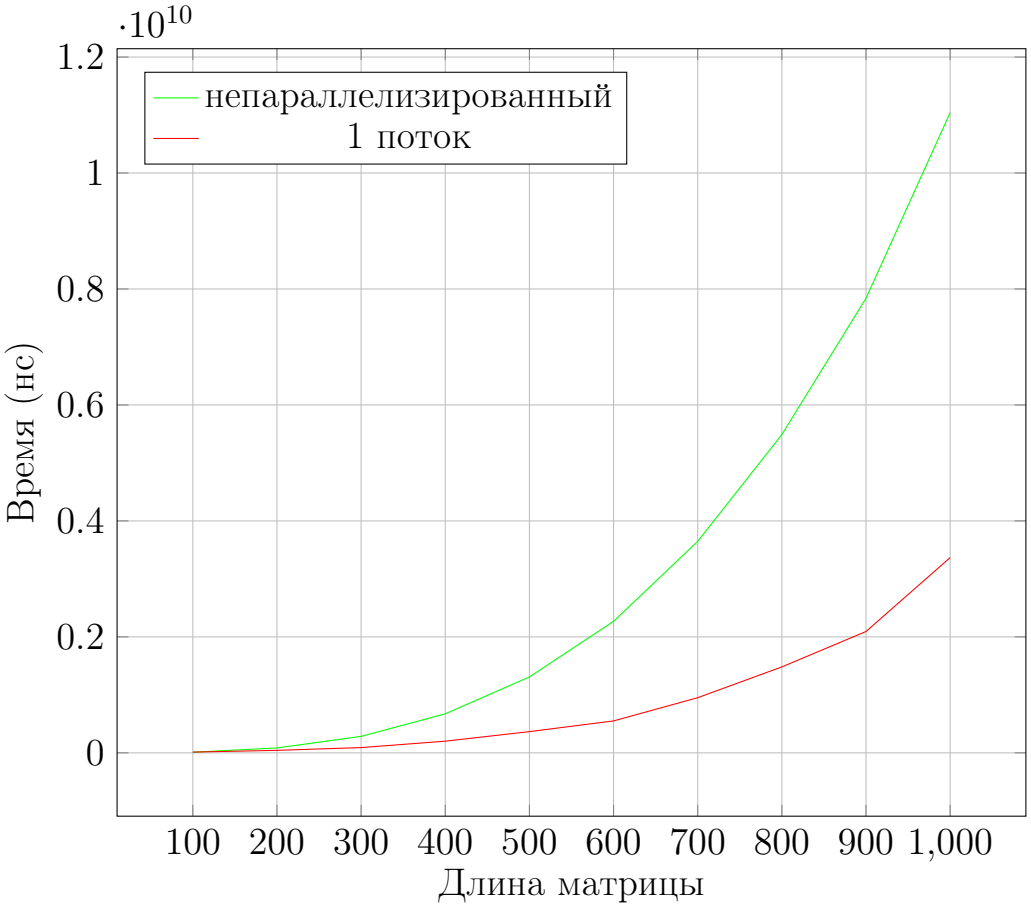

Рисунок 4.3 — Нечётная длина, сравнение параллельных и последовательных вычислений

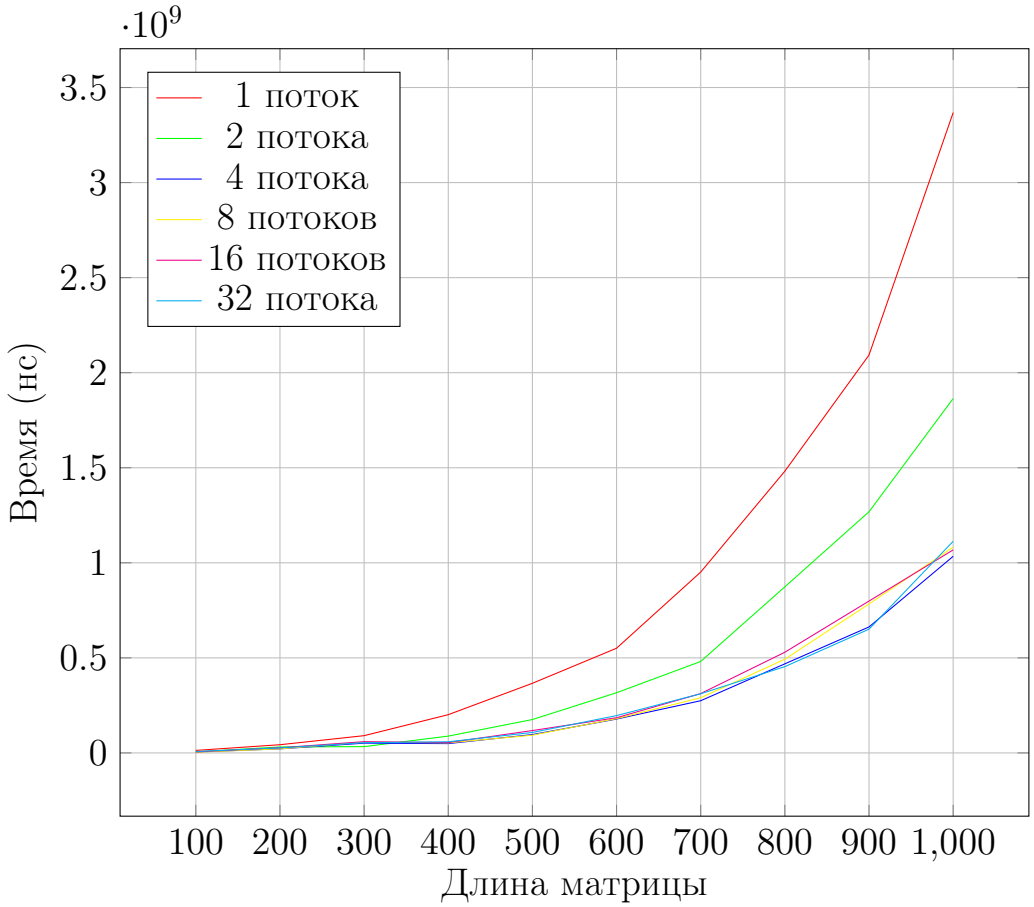

Рисунок 4.4 — Нечётная длина, сравнение параллельных вычислений при разном кол-ве потоков

Данный эксперимент проводился на ноутбуке, подключённом к сети питания. Модель процессора ноутбука: Intel i5-8400H с максимальной тактовой частотой 2.500 ГГц в обычном режиме и 8 логическими ядрами.

4.2 Вывод

Исходя из полученых графиков, можно заключить, что распараллеливание значительно ускоряет вычисления, по при этом не имеет смысла разбивать их на число потоков большее, чем количество логических ядер процессора, на котором эти вычисления проводятся.

ЗАКЛЮЧЕНИЕ

В ходе данной работы было проведено сравнение расчета произведения матриц на нескольких потоках, а именно на 1, 2, 4, 8, 16 и 32 и были сделаны следующие выводы:

— распараллеленные вычисления работают существенно быстрее последовательных;

— исользование числа потоков больше, чем число потоков процессора не дает выйгрыша по времени и может даже работать медленнее.

Все цели, поставленные на эту работы, были выполнены:

— изучено распараллеливания вычислений и работа с потоками;

— реализация распараллеленных вычислений;

— экспериментальное сравнение работы алгоритма на разном количестве потоков.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Анисимов Н.С., Строганов Ю.В. - Реализация алгоритма умножения матриц по Винограду на языке Haskell

2. Корн Г., Корн Т. - Алгебра матриц и матричное исчисление

3. Документация по chrono [Электронный ресурс]. - Режим доступа: http://www.cplusplus.com/reference/chrono/ Дата обращения: 30.10.2019

4. Документация по thread [Электронный ресурс]. - Режим доступа: https://ru.cppreference.com/w/cpp/thread/thread Дата обращения: 30.10.2019