МГТУ им. Н.Э.Баумана

Лабораторная работа №2

По курсу: "Анализ алгоритмов"

Параллельное умножение матриц

Работу выполнил: Луговой Дмитрий, ИУ7-51Б

Преподаватель: Волкова Л.Л.

Москва, 2019

Оглавление

Введение

Цель работы: изучение возможности параллельных вычислений и использование такого подхода на практике. Реализация параллельного алгоритма Винограда умножения матриц. В данной лабораторной работе рассматривается алгоритм Винограда и параллельный алгоритм Винограда. Необходимо сравнить зависимость времени работы алгоритма от числа параллельных потоков и размера матриц, провести сравнение стандартного и параллельного алгоритмов.

Задачи работы

Задачами данной лабораторной являются:

1) Научиться писать многопоточные программы;

2) Применить полученные знания на практике, переписав алгоритм Винограда в несколько потоков;

3) Провести замеры скорости работы однопоточной и многопоточной реализаций и проанализировать полученные результаты.

1 | Аналитическая часть

В данном разделе содержатся описание алгоритма Винограда умножения матриц, определение потока и многопоточного программирования.

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы A и B размерностей $m \times n$, $n \times q$ соответственно:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix},$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1q} \\ b_{21} & b_{22} & \cdots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nq}. \end{bmatrix}.$$

Тогда матрица C размерностью $m \times q$

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1q} \\ c_{21} & c_{22} & \cdots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mq} \end{bmatrix},$$

в которой:

$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad (i = 1, 2, \ldots m;\ j = 1, 2, \ldots q)$

называется их произведением. Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором; в этом случае говорят, что матрицы согласованы. В частности, умножение всегда выполнимо, если оба сомножителя — [[Квадратная матрица|квадратная матрица]] одного и того же порядка.

Таким образом, из существования произведения AB вовсе не следует существование произведения BA

1.2 Алгоритм Винограда

Алгоритм Винограда — алгоритм умножения матриц, предложенный в 1987 году Ш. Виноградом. В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2,3755})$, где n — размер стороны матрицы. Алгоритм Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц. Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее. Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$. Их скалярное произведение равно: $V * W = v1w1 + v2w2 + v3w3 + v4w4$. Это равенство можно переписать в виде: $V * W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4$.

Кажется, что второе выражение задает больше работы, чем первое: вместо четырех умножений мы насчитываем их шесть, а вместо трех сложений - десять. Однако выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй. На практике это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.3 Многопоточность

Поток выполнения — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени).

На одном процессоре многопоточность обычно происходит путём временного мультиплексирования (как и в случае многозадачности): процессор переключается между разными потоками выполнения. Это переключение контекста обычно происходит достаточно часто, чтобы пользователь воспринимал выполнение потоков или задач как одновременное. В многопроцессорных и многоядерных системах потоки или задачи могут реально выполняться одновременно, при этом каждый процессор или ядро обрабатывает отдельный поток или задачу.

Потоки возникли в операционных системах как средство распараллеливания вычислений.

Параллельное выполнение нескольких работ в рамках одного интерактив-

ного приложения повышает эффективность работы пользователя. Так, при работе с текстовым редактором желательно иметь возможность совмещать набор нового текста с такими продолжительными по времени операциями, как переформатирование значительной части текста, печать документа или его сохранение на локальном или удаленном диске. Еще одним примером необходимости распараллеливания является сетевой сервер баз данных. В этом случае параллелизм желателен как для обслуживания различных запросов к базе данных, так и для более быстрого выполнения отдельного запроса за счет одновременного просмотра различных записей базы. Именно для этих целей современные ОС предлагают механизм многопоточной обработки (multithreading). Понятию «поток» соответствует последовательный переход процессора от одной команды программы к другой. ОС распределяет процессорное время между потоками. Процессу ОС назначает адресное пространство и набор ресурсов, которые совместно используются всеми его потоками.

Создание потоков требует от ОС меньших накладных расходов, чем процессов. В отличие от процессов, которые принадлежат разным, вообще говоря, конкурирующим приложениям, все потоки одного процесса всегда принадлежат одному приложению, поэтому ОС изолирует потоки в гораздо меньшей степени, нежели процессы в традиционной мультипрограммной системе. Все потоки одного процесса используют общие файлы, таймеры, устройства, одну и ту же область оперативной памяти, одно и то же адресное пространство. Это означает, что они разделяют одни и те же глобальные переменные. Поскольку каждый поток может иметь доступ к любому виртуальному адресу процесса, один поток может использовать стек другого потока. Между потоками одного процесса нет полной защиты, потому что, во-первых, это невозможно, а во-вторых, не нужно. Чтобы организовать взаимодействие и обмен данными, потокам вовсе не требуется обращаться к ОС, им достаточно использовать общую память — один поток записывает данные, а другой читает их. С другой стороны, потоки разных процессов по-прежнему хорошо защищены друг от друга.

2 | Конструкторская часть

В этом разделе содержится схема алгоритма Винограда умножения матриц и описание распараллеливания этого алгоритма.

2.1 Схемы алгоритмов

На Рис.2.1, 2.2, 2.3 и 2.4 представлена схема оптимизированного алгоритма Винограда умножения матриц.

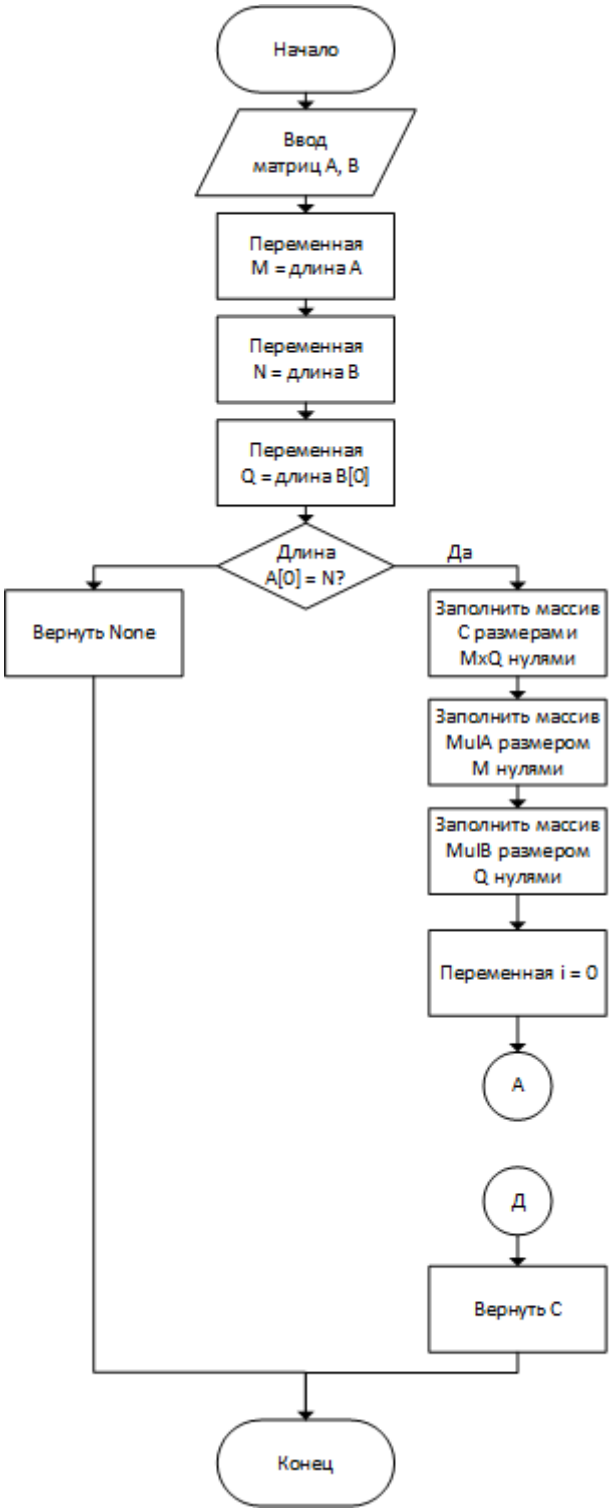

Рис. 2.1: Алгоритм Винограда умножения матриц

Рис. 2.2: Алгоритм Винограда умножения матриц(продолжение 1)

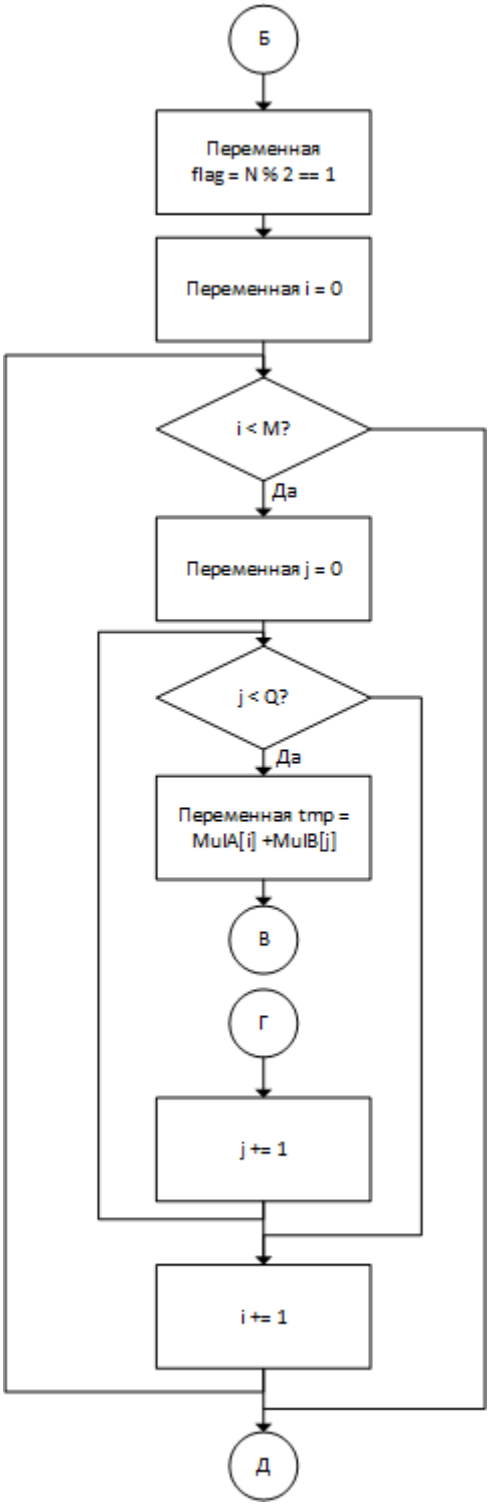

Рис. 2.3: Алгоритм Винограда умножения матриц(продолжение 2)

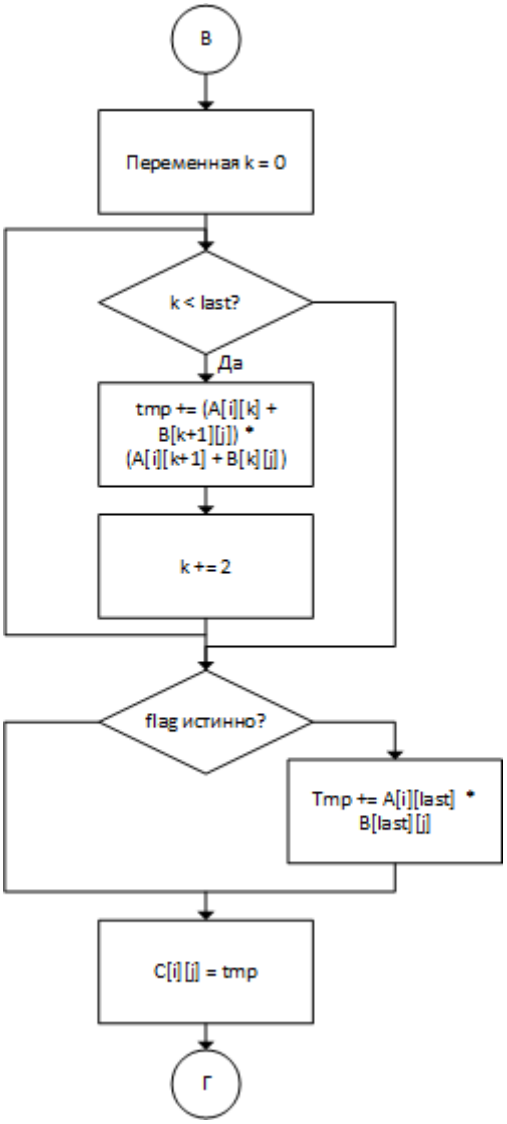

Рис. 2.4: Алгоритм Винограда умножения матриц(продолжение 3)

2.2 Распараллеливание программы

Распараллеливание программы должно ускорять время работы. Это достигается за счет реализации в узких участках (например в циклах с большим количеством независимых вычислений).

В предложенном алгоритме данными участками являются тройной цикл поиска результата(участок от Б до Д), цикл вычисления сумм перемноженных пар строк первой матрицы и сумм перемноженных пар столбцов второй матрицы(участок от А до Б).

Данные участки программы как раз предлагается распараллелить.

2.3 Вывод

Была приведена схема алгоритма Винограда и выявлены участки программы, которые могут быть распараллелены.

3 | Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода

3.1 Требования к ПО

На вход поступают две целочисленные матрицы, на выходе должен возвращаться результат их умножения и код завершения.

3.2 Средства реализации

Для реализации представленных алгоритмов был выбран язык C++. Время работы алгоритмов было замерено с помощью функции steady_clock() из библиотеки chrono. Для тестирования использовался компьютер на базе процессора Intel Core i5 (4 физических ядра, 8 логических).

3.3 Листинги кода

В Листинге 3.1 показана реализация алгоритма Винограда умножения матриц.

Листинг 3.1: Функция умножения матриц алгоритмом Винограда

```
1  int vinograd(std::vector<std::vector<int>> &C, const ::vector<std::vector<int>>
       &A,
2          const std::vector<std::vector<int>> &B)
3  {
4      int m = A.size();
5      int n = B.size();
6      if (m == 0 || n == 0)
7          return ERR_EMPTY;
8      if (A[0].size() != n)
9          return ERR_SIZE;
10     int q = B[0].size();
11     std::vector<int> mulA(m,0);
12     for (int i = 0; i < m; i++)
13     {
14         for (int j = 0; j < n - 1; j += 2)
15         {
```

```
16              mulA[i] -= A[i][j] * A[i][j + 1];
17          }
18      }
19      std::vector<int> mulB(q,0);
20      for (int i = 0; i < q; i++)
21      {
22          for (int j = 0; j < n - 1; j += 2)
23          {
24              mulB[i] -= B[j][i] * B[j + 1][i];
25          }
26      }
27      int last = n - 1;
28      bool flag = n % 2 == 1;
29      C = std::vector<std::vector<int>> (m, std::vector<int> (q, 0));
30      for (int i = 0; i < m; i++)
31      {
32          for (int j = 0; j < q; j++)
33          {
34              int tmp = mulA[i] + mulB[j];
35              for (int k = 0; k < n - 1; k += 2)
36              {
37                  tmp += (A[i][k] + B[k + 1][j]) * (A[i][k + 1] + B[k][j]);
38              }
39              if (flag)
40              {
41                  tmp += A[i][last] * B[last][j];
42              }
43              C[i][j] = tmp;
44          }
45      }
46      return OK;
47  }
```

В Листингах 3.2, 3.3, 3.4. 3.5 показана реализация многопоточного алгоритма Винограда умножения матриц.

Листинг 3.2: Функция умножения матриц многопоточным алгоритмом Винограда

```
1  int threadedVinograd(std::vector<std::vector<int>> &C, const std::
       vector<int>> &A,
2                       const std::vector<std::vector<int>> &B, const int &nThreads)
3  {
4      int m = A.size();
5      int n = B.size();
6      if (m == 0 || n == 0)
7          return ERR_EMPTY;
8      if (A[0].size() != n)
9          return ERR_SIZE;
10     int q = B[0].size();
11     std::vector<std::thread> threads;
12     std::vector<int> mulA(m,0);
```

```
13      double start = 0;
14      double del = m / static_cast<double>(nThreads);
15      for (int i = 0; i < nThreads; i++)
16      {
17          threads.push_back(std::thread(computeMulA, std::ref(mulA), A, round(start),
18                                        round(start + del)));
19          start += del;
20      }
21      for (auto &thread: threads)
22      {
23          thread.join();
24      }
25      start = 0;
26      del = q / static_cast<double>(nThreads);
27      std::vector<int> mulB(q,0);
28      for (int i = 0; i < nThreads; i++)
29      {
30          threads[i] = std::thread(computeMulB, std::ref(mulB), B,
31                                   round(start), round(start + del));
32          start += del;
33      }
34      for (auto &thread: threads)
35      {
36          thread.join();
37      }
38
39      C = std::vector<std::vector<int>> (m, std::vector<int> (q, 0));
40      start = 0;
41      del = m / static_cast<double>(nThreads);
42      for (int i = 0; i < nThreads; i++)
43      {
44          threads[i] = std::thread(computeResult, std::ref(C), A, B,
45                                   mulA, mulB, round(start), round(start + del));
46          start += del;
47      }
48      for (auto &thread: threads)
49      {
50          thread.join();
51      }
52      return OK;
53  }
```

Листинг 3.3: Функция вычисления сумм строк первой матрицы

```
1  void computeMulA(std::vector<int> &mulA, std::vector<std::vector<int>> A, int
       startRow, int endRow)
2  {
3      int n = A[0].size();
4      for (int i = startRow; i < endRow; i++)
5      {
6          for (int j = 0; j < n - 1; j += 2)
```

```
        {
            mulA[i] -= A[i][j] * A[i][j + 1];
        }
    }
}
```

Листинг 3.4: Функция вычисления сумм столбцов второй матрицы

```
void computeMulB(std::vector<int> &mulB, std::vector<std::vector<int>> B, int
    startCol, int endCol)
{
    int n = B.size();
    for (int i = startCol; i < endCol; i++)
    {
        for (int j = 0; j < n - 1; j += 2)
        {
            mulB[i] -= B[j][i] * B[j + 1][i];
        }
    }
}
```

Листинг 3.5: Функция вычисления результирующей матрицы

```
void computeResult(std::vector<std::vector<int>> &C, std::vector<std::vector<int
    >> A,
                   std::vector<std::vector<int>> B, std::vector<int> mulA,
                   std::vector<int> mulB, int startRow, int endRow)
{
    int n = B.size();
    int q = B[0].size();
    int last = n - 1;
    bool flag = n % 2 == 1;
    for (int i = startRow; i < endRow; i++)
    {
        for (int j = 0; j < q; j++)
        {
            int tmp = mulA[i] + mulB[j];
            for (int k = 0; k < n - 1; k += 2)
            {
                tmp += (A[i][k] + B[k + 1][j]) * (A[i][k + 1] + B[k][j]);
            }
            if (flag)
            {
                tmp += A[i][last] * B[last][j];
            }
            C[i][j] = tmp;
        }
    }
}
```

4 | Экспериментальная часть

В данном разделе приведены примеры работы программы, постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных.

4.1 Примеры работы

Пример 1
Матрица A:
1 2 3
4 5 6
Матрица B:
1
2
3
Результирующая матрица:
14
32

Пример 2
Матрица A:
5 2
1 4
Матрица B:
0 3
-6 1
Результирующая матрица:
-12 17
-24 7

Пример 3
Матрица A:
2 7
1 3
Матрица B:
-3 7
1 -2
Результирующая матрица:
1 0
0 1

4.2 Функциональное тестирование

Было проведено функциональное тестирование программы, результаты которого занесены в Таблицу 4.1,1 столбец которой - номер тестового случая, 2 и 3 столбцы - виды матриц, поступающих на вход, 4 столбец - ожидаемый результат, 5 столбец - полученный результат.

№	A	B	Ожидаемый результат	Полученный результат
1	Случайная	Пустая	Код ошибки	Код ошибки
2	Пустая	Случайная	Код ошибки	Код ошибки
3	Пустая	Пустая	Код ошибки	Код ошибки
4	Случайная	Нулевая	Нулевая	Нулевая
5	Нулевая	Случайная	Нулевая	Нулевая
6	Единичная	Квадратная	B	B
7	Квадратная	Единичная	A	A
8	Размера $M \times N$	Размера $M \times N$	Код ошибки	Код ошибки

Таблица 4.1: Тестовые случаи

Программа успешно прошла все тестовые случаи, все полученные результаты совпали с ожидаемыми.

4.3 Сравение алгоритмов по времени

Для экспериментов использовались матрицы, размер которых варьируется от 100×100 до 1000×1000 с шагом 100 для матриц четных размеров и от 101×101 до 1001×1001 с шагом 100 для матриц нечетных размеров. Количество повторов каждого эксперимента = 100. Результат одного эксперимента рассчитывается как средний из результатов проведенных испытаний с одинаковыми входными данными.

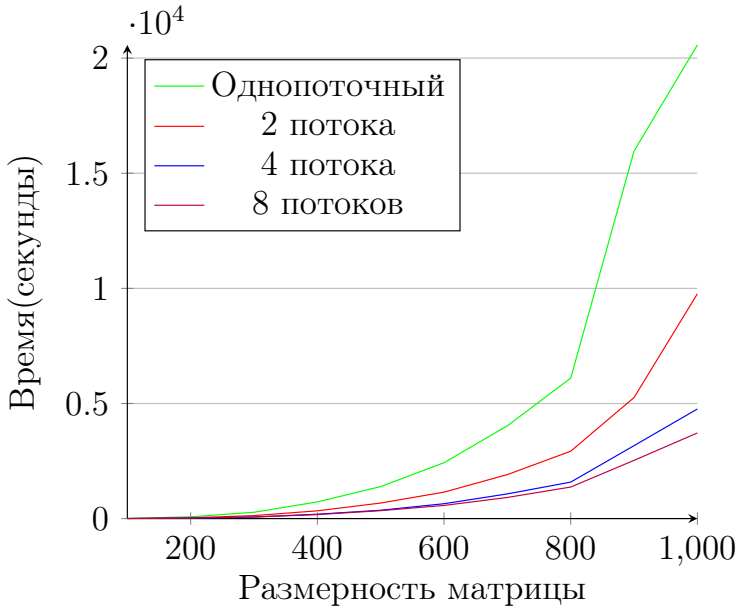

Рис. 4.1: Алгоритм Винограда умножения матриц(четных размеров)

На Рис. 4.1 видно, что с увеличением размеров матриц разница между многопоточной и однопоточной реализациями алгоритма Винограда растет, двухпоточная реализация работает в ≈ 2 раза быстрее однопоточной, четырехпоточная - в ≈ 4 раза быстрее однопоточной, однако с ростом числа потоков рост преимущества многопоточной реализации уменьшается, и при 8 потоках алгоритм работает в ≈ 5.5 раз быстрее однопоточного.

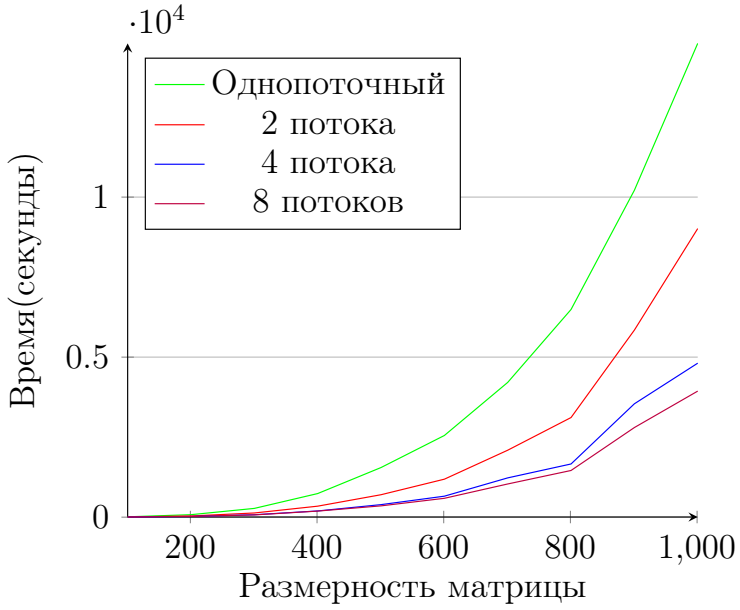

Рис. 4.2: Алгоритм Винограда умножения матриц(нечетных размеров)

На Рис. 4.2 видно, что многопоточный алгоритм Винограда сохраняет свое превосходство над однопоточной реализацией, и разница во времени реализаций сохраняется.

4.4 Вывод

Программа успешно прошла все тестовые случаи, все полученные результаты совпали с ожидаемыми. Было показано преимущество параллельной реализации алгоритма Винограда, с ростом числа поток преимущество данной реализации по сравнению с однопоточной реализацией увеличивается, однако рост с увеличением числа потоков замедляется.

Заключение

В ходе лабораторной работы я изучил возможности параллельных вычислений и использовал такой подход на практике. Был реализован алгоритм Винограда умножения матриц с помощью параллельных вычислений. Было произведено сравнение работы обычного алгоритма Винограда и параллельной реализации при увеличении количества потоков. Выяснилось, что увеличение потоков до 8-ми сокращает время работы в 5.5 раз по сравнению с однопоточной реализацией. Однако с увеличением количества потоков рост преимущества по сравнению с однопоточной реализацией замедляется.