СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: изучение возможностей параллельных вычислений и использование такого подхода на практике.
При выполнении лабораторной работы поставлены такие задачи:

1) реализовать алгоритм Винограда;

2) разработать и реализовать 2 версии параллельной реализации алгоритма Винограда;

3) провести замеры времени в зависимости от числа потоков и размера матриц;

4) провести сравнительный анализ реализованных алгоритмов.

1 Аналитический раздел

В данном разделе будет представлено понятие умножения матрицы, рассмотрен алгоритм Винограда и способы его параллельной реализации.

Матрицей A размера $m \times n$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов [1]. Умножение матриц возможно, только когда число столбцов первой матрицы равно числу строк второй.

Алгоритм Винограда.

При рассмотрении результата умножения матриц видно, что каждый элемент – скалярное произведение векторов, представляющих строки и столбцы матриц [2].

Рассмотрим 2 вектора $V = (v1, v2, v3, v4)$ и $W = (w1,w2,w3,w4)$. Их скалярное произведение равно:

$$V \bullet W = v1w1 + w2w2 + v3w3 + v4w4 \tag{1.1}$$

Или:

$$V \bullet W = (v1+w2)(v2+w1)+(v3+w4)(v4+w3)-v1v2-v3v4-w1w2-w3w4 \tag{1.2}$$

Можно заметить, что 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.

Параллельный алгоритм Винограда.

При использовании алгоритма Винограда можно заметить 2 основные части, которые могут подойти для распараллеливания, это предварительный подсчёт двух массивов, которые используется в дальнейшем, при этом они сами не обращаются к собственным элементам, которые можно подсчиты-

вать параллельно. Значит, каждый поток может вычислять некоторую часть данных массивов

Другой частью, пригодной для параллельных вычислений является главный цикл, в котором происходит построчное заполнение результирующей матрицы, следует заметить, что значение каждой строки не зависит от остальных, таким образом каждый поток может выполнять вычисление отдельной строки.

Параллельное программирование.

Параллельные вычисления - способ организации компьютерных вычислений, при котором программы разрабатываются, как набор взаимодействующих вычислительных процессов, работающих асинхронно и при этом одновременно [3].

Параллельное программирование - это техника программирования, которая использует преимущества многоядерных или многопроцессорных компьютеров и является подмножеством более широкого понятия многопоточности (multithreading).

Использование параллельного программирования становится наиболее необходимым, поскольку позволяет максимально эффективно использовать возможности многоядерных процессоров и многопроцессорных систем. По ряду причин, включая повышение потребления энергии и ограничения пропускной способности памяти, увеличивать тактовую частоту современных процессоров стало невозможно. Вместо этого производители процессоров стали увеличивать их производительность за счёт размещения в одном чипе нескольких вычислительных ядер, не меняя или даже снижая тактовую частоту. Поэтому для увеличения скорости работы приложений теперь следует по-новому подходить к организации кода, а именно - оптимизировать программы под многоядерные системы.

Вывод

В данном разделе был рассмотрен алгоритм Винограда, способы его распараллеливания и понятие параллельного программирования.

2 Конструкторский раздел

В данном разделе будет рассмотрена схема алгоритма Винограда и описаны его параллельные реализации.

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма Винограда умножения матриц.

2.2 Распараллеливание программы

В первой параллельной реализации параллельно выполняется подсчёт строк результирующей матрицы, в главном потоке подсчитываются вспомогательные массивы mulV и mulH, после чего он создаёт потоки, формирующие результирующую матрицу, передаёт им параметры, и после их завершения, в случае если размеры матрицы нечётные, дополняет ее.

Во второй реализации параллельно рассчитываются массивы mulH и mulV, так как их подсчёт не зависит от их собственных значений, то их можно распараллелить. Главный поток создаёт рабочие потоки, передаёт им параметры и определяет границы участков массивов, над которыми они будут работать, эти потоки подсчитывают значения массивов, после чего управление возвращается к главному потоку.

2.3 Вывод

В данном разделе была рассмотрена схема алгоритма Винограда умножения матриц, описаны 2 метода его параллельной реализации.

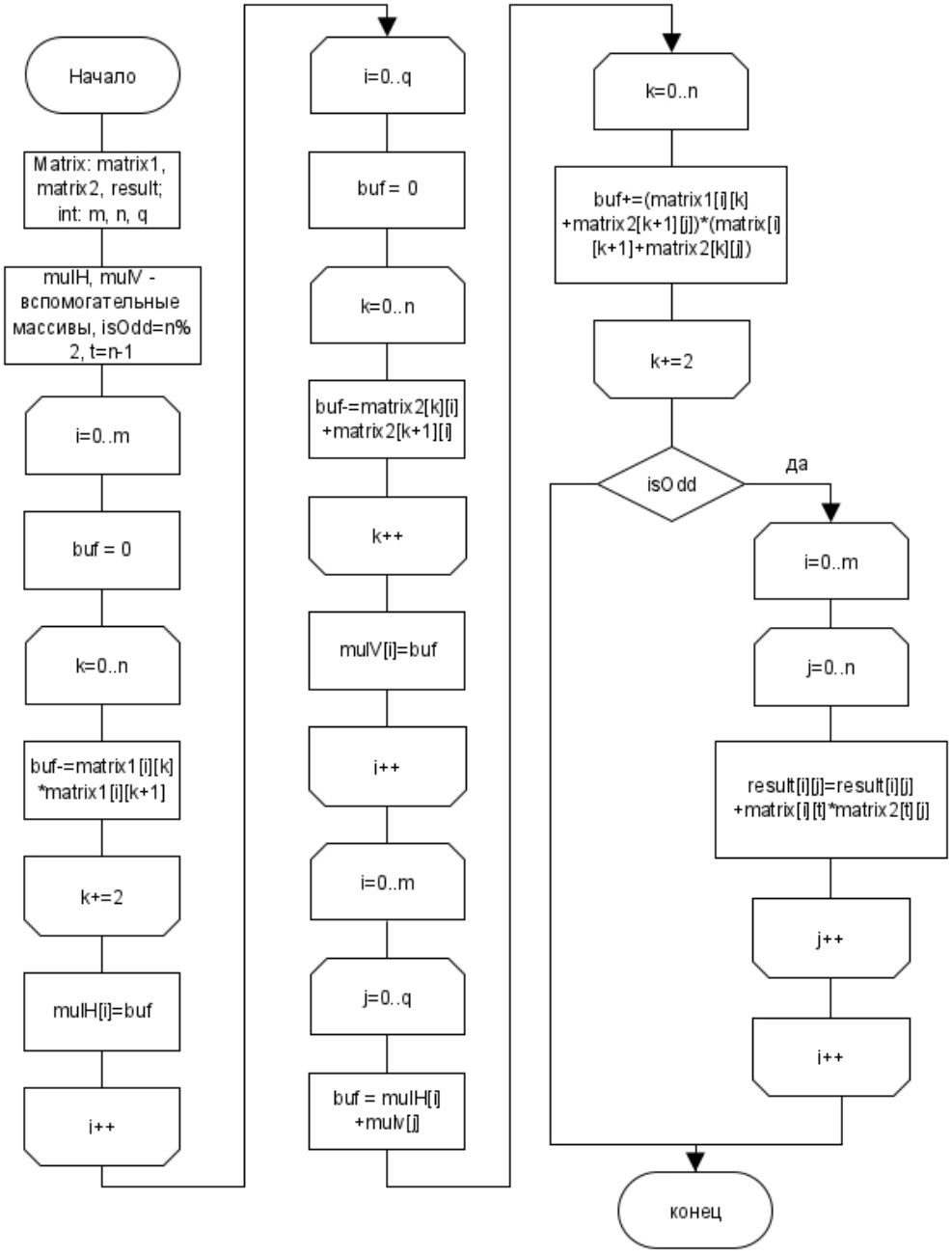

Рисунок 2.1 — Алгоритм Винограда

3 Технологический раздел

В данном разделе будут представлены листинги кода реализованных алгоритмов.

3.1 Средства реализации

В данной работе используется язык программирования C++. Среда разработки Visual Studio Code. Для замера процессорного времени используется функция QueryPerformanceCounter из библиотеки windows.h. Параллельные вычисления выполняются с использование библиотеки <thread> [4].

Замеры времени были произведены на: Intel(R) Core(TM) i5-8250U CPU @1.60GHz 1.80 Ghz, 4 ядра, 8 логических процессоров.

3.2 Листинг кода

В листинге 3.1приведен код последовательной реализации алгоритма Винограда.

Листинг 3.1 — Алгоритм Винограда

```
1  void multiplyMatricesWinograd(int **matrix1, int m, int n, int **matrix2,
       int q, int **result)
2  {
3      int buf;
4      const int isOdd = n % 2;
5      const int t = n - 1;
6      if (isOdd)
7      n -= 1;
8
9      int *mulH = new int[m];
10     for (int i = 0; i < m; i++)
11     {
12         buf = 0;
13         for (int k = 0; k < n; k += 2)
14         buf -= matrix1[i][k] * matrix1[i][k + 1];
15         mulH[i] = buf;
16     }
17
18     int *mulV = new int[q];
19     for (int i = 0; i < q; i++)
```

```
20      {
21          buf = 0;
22          for (int k = 0; k < n; k += 2)
23          buf -= matrix2[k][i] * matrix2[k + 1][i];
24          mulV[i] = buf;
25      }
26
27      for (int i = 0; i < m; i++)
28      {
29          for (int j = 0; j < q; j++)
30          {
31              buf = mulH[i] + mulV[j];
32              for (int k = 0; k < n; k += 2)
33              buf += (matrix1[i][k] + matrix2[k + 1][j]) * (matrix1[i][k + 1]
                    + matrix2[k][j]);
34              result[i][j] = buf;
35          }
36      }
37
38      if (isOdd)
39      {
40          for (int i = 0; i < m; i++)
41          for (int j = 0; j < q; j++)
42          result[i][j] += matrix1[i][t] * matrix2[t][j];
43      }
44
45      delete[] mulH;
46      delete[] mulV;
47  }
```

В листинге 3.2 представлен код параллельной реализации алгоритма Винограда, в которой потоки выполняют подсчёт строк в главном цикле.

Листинг 3.2 — Первая параллельная реализация алгоритма Винограда

```
1  void parallelMainCycle(int **matrix1, int **matrix2, int *mulH, int *mulV,
       int rs, int re, int columns, int rows, int **result)
2  {
3      int buf;
4      for (int i = rs; i < re; i++)
5      {
6          for (int j = 0; j < rows; j++)
7          {
8              buf = mulH[i] + mulV[j];
9              for (int k = 0; k < columns; k += 2)
10             buf += (matrix1[i][k] + matrix2[k + 1][j]) * (matrix1[i][k + 1]
                   + matrix2[k][j]);
```

```
11              result[i][j] = buf;
12          }
13      }
14  }
15
16  void multiplyMatricesWinogradP1(int **matrix1, int m, int n, int **matrix2,
        int q, int **result, int nThreads)
17  {
18      int buf;
19      const int isOdd = n % 2;
20      const int t = n - 1;
21      if (isOdd)
22      n -= 1;
23
24      int *mulH = new int[m];
25      for (int i = 0; i < m; i++)
26      {
27          buf = 0;
28          for (int k = 0; k < n; k += 2)
29          buf -= matrix1[i][k] * matrix1[i][k + 1];
30          mulH[i] = buf;
31      }
32
33      int *mulV = new int[q];
34      for (int i = 0; i < q; i++)
35      {
36          buf = 0;
37          for (int k = 0; k < n; k += 2)
38          buf -= matrix2[k][i] * matrix2[k + 1][i];
39          mulV[i] = buf;
40      }
41
42      thread* threads = new thread[nThreads];
43      int rows = m / nThreads;
44      int rs = 0;
45      for (int t = 0; t < nThreads; t++)
46      {
47          int re = t == nThreads - 1 ? m : (rs + rows);
48          threads[t] = thread(parallelMainCycle, matrix1, matrix2, mulH,
                mulV, rs, re, n, q, result);
49          rs = re;
50      }
51
52      for (int i = 0; i < nThreads; i++)
53      threads[i].join();
54
```

```
55      if (isOdd)
56      {
57          for (int i = 0; i < m; i++)
58          for (int j = 0; j < q; j++)
59          result[i][j] += matrix1[i][t] * matrix2[t][j];
60      }
61
62      delete[] threads;
63      delete[] mulH;
64      delete[] mulV;
65  }
```

В листинге 3.2 представлен код параллельной реализации алгоритма Винограда, в которой потоки выполняют подсчёт массивов mulV и mulH.

Листинг 3.3 — Первая параллельная реализация алгоритма Винограда

```
1  void parallelMul(int **matrix1, int vs, int ve, int n, int **matrix2, int
       hs, int he, int *mulV, int *mulH)
2  {
3       int buf;
4       for (int i = vs; i < ve; i++)
5       {
6           buf = 0;
7           for (int k = 0; k < n; k += 2)
8           buf -= matrix1[i][k] * matrix1[i][k + 1];
9           mulH[i] = buf;
10      }
11
12      for (int i = hs; i < he; i++)
13      {
14          buf = 0;
15          for (int k = 0; k < n; k += 2)
16          buf -= matrix2[k][i] * matrix2[k + 1][i];
17          mulV[i] = buf;
18      }
19 }
20
21 void multiplyMatricesWinogradP2(int **matrix1, int m, int n, int **matrix2,
       int q, int **result, int nThreads)
22 {
23      int buf;
24      const int isOdd = n % 2;
25      const int t = n - 1;
26      if (isOdd)
27      n -= 1;
28
```

```
29      int *mulH = new int[m];
30      int *mulV = new int[q];
31
32      thread *threads = new thread[nThreads];
33      int vs = 0;
34      int vt = m / nThreads;
35      int hs = 0;
36      int ht = q / nThreads;
37      for (int t = 0; t < nThreads; t++)
38      {
39          int ve = t == nThreads - 1 ? m : (vs + vt);
40          int he = t == nThreads - 1 ? q : (hs + ht);
41          threads[t] = thread(parallelMul, matrix1, vs, ve, n, matrix2, hs,
                he, mulV, mulH);
42
43          vs = ve;
44          hs = he;
45      }
46
47      for (int i = 0; i < nThreads; i++)
48      threads[i].join();
49
50      for (int i = 0; i < m; i++)
51      {
52          for (int j = 0; j < q; j++)
53          {
54              buf = mulH[i] + mulV[j];
55              for (int k = 0; k < n; k += 2)
56              buf += (matrix1[i][k] + matrix2[k + 1][j]) * (matrix1[i][k + 1]
                    + matrix2[k][j]);
57              result[i][j] = buf;
58          }
59      }
60
61      if (isOdd)
62      {
63          for (int i = 0; i < m; i++)
64          for (int j = 0; j < q; j++)
65          result[i][j] += matrix1[i][t] * matrix2[t][j];
66      }
67
68      delete[] mulH;
69      delete[] mulV;
70  }
```

3.3 Проведение тестирования

Изначально было проведено тестирование стандартного алгоритма умножения матриц на квадратных матрицах размерами $0 \times 0, 1 \times 1, 3 \times 3, 6 \times 6$ и прямоугольных $1 \times 2 \bullet 2 \times 1, 2 \times 4 \bullet 4 \times 2$ и $3 \times 5 \bullet 5 \times 5$, все тесты были пройдены успешно.

После чего была проведена серия тестов на матрицах $A = M \times Q$ и $B = Q \times N$, где M, N, Q принимают значения от 1 до 10, независимо друг от друга, матрицы заполняются случайными значениями. Сначала выполнялось умножение стандартным алгоритмом, результат которого принимался за контрольное значение, после чего результаты алгоритма Винограда и оптимизированного алгоритма Винограда сравнивались с контрольным значением и при совпадении результат считался корректным. Все тесты были пройдены успешно.

Вывод.

В данном разделе были рассмотрены листинги кода реализованных алгоритмов, а также их тестирование.

4 Исследовательский раздел

ДОБАВИТЬ ОПИСАНИЕ ЛЕГЕНД В данном разделе будет измерено время работы алгоритмов и сделаны выводы на основе полученных данных.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Для каждого из алгоритмов был проведён замер времени на квадратных матрицах размерами от 100 до 1000, с шагом 100, матрицы заполнены случайными элементами. Для каждого теста было проведено 5 замеров, на графиках представлен усреднённый результат. На рисунках 4.1–4.3 время, обозначаемое t представлено в миллисекундах.

На рисунке 4.1 представлено сравнение времени работы реализации алгоритма умножения матриц Винограда с параллельным главным циклом. Числами обозначено количество выделенных потоков.

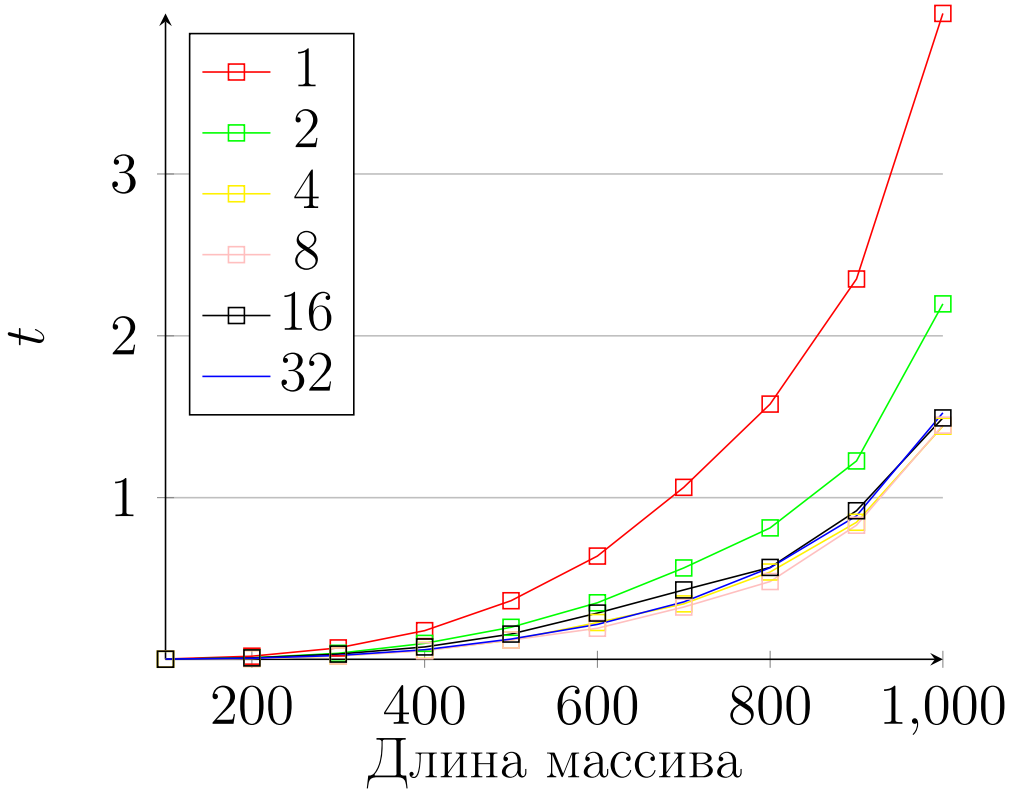

Рисунок 4.1 — Сравнение времени работы алгоритма Винограда с параллельным главным циклом

Из графика 4.1 становится ясно, что прирост производительности имеет место только при увеличении числа потоков до 4, после чего она меняется незначительно.

На рисунке 4.2 представлено сравнение времени работы реализации алгоритма умножения матриц Винограда с параллельным вычислением массивов mulH и mulV. Числами обозначено количество выделенных потоков.

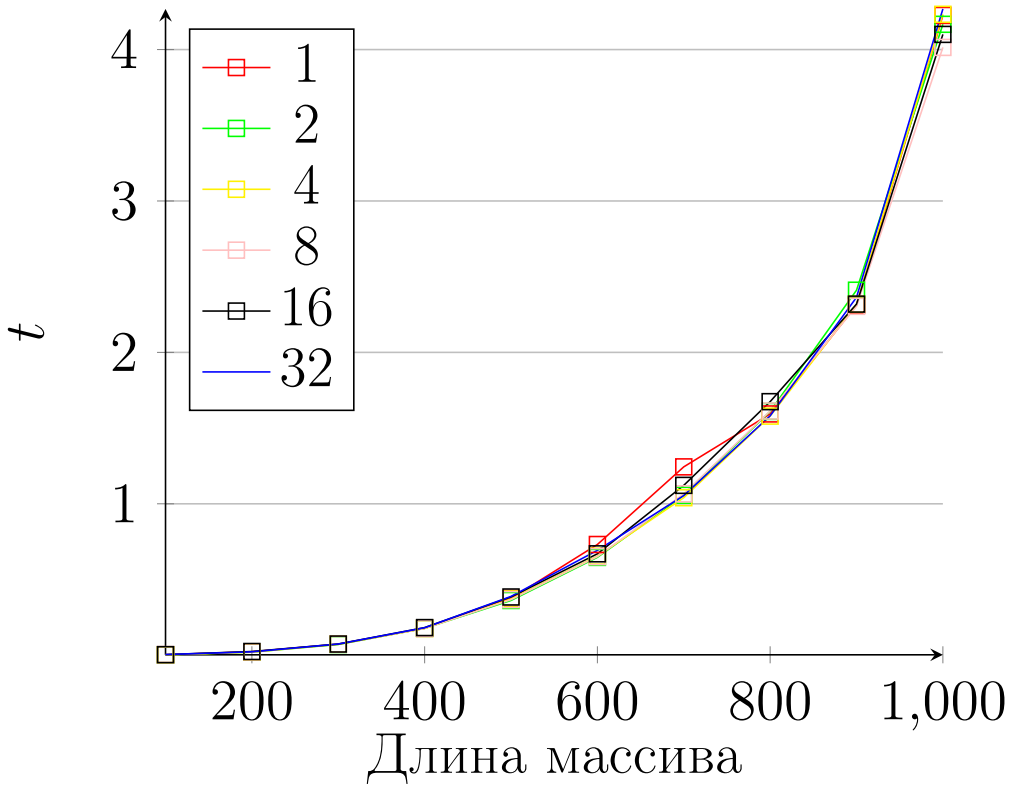

Рисунок 4.2 — Сравнение времени работы алгоритма Винограда с параллельным вычислением mulV и mulH

Из графика 4.2 видно, что распараллелиливание вычисления массивов mulV и mulH не имеет смысла, так как практически уменьшает затраты по времени.

На рисунке 4.3 представлено сравнение времени работы последовательной реализации алгоритма Винограда и его параллельных реализаций. На графике введены обозначения: «последовательный» – последовательный алгоритм Винограда, «Main» – параллельно выполняется главный цикл, «Mul» – параллельно считаются массивы mulV и mulH, цифры рядом - количество выделенных потоков.

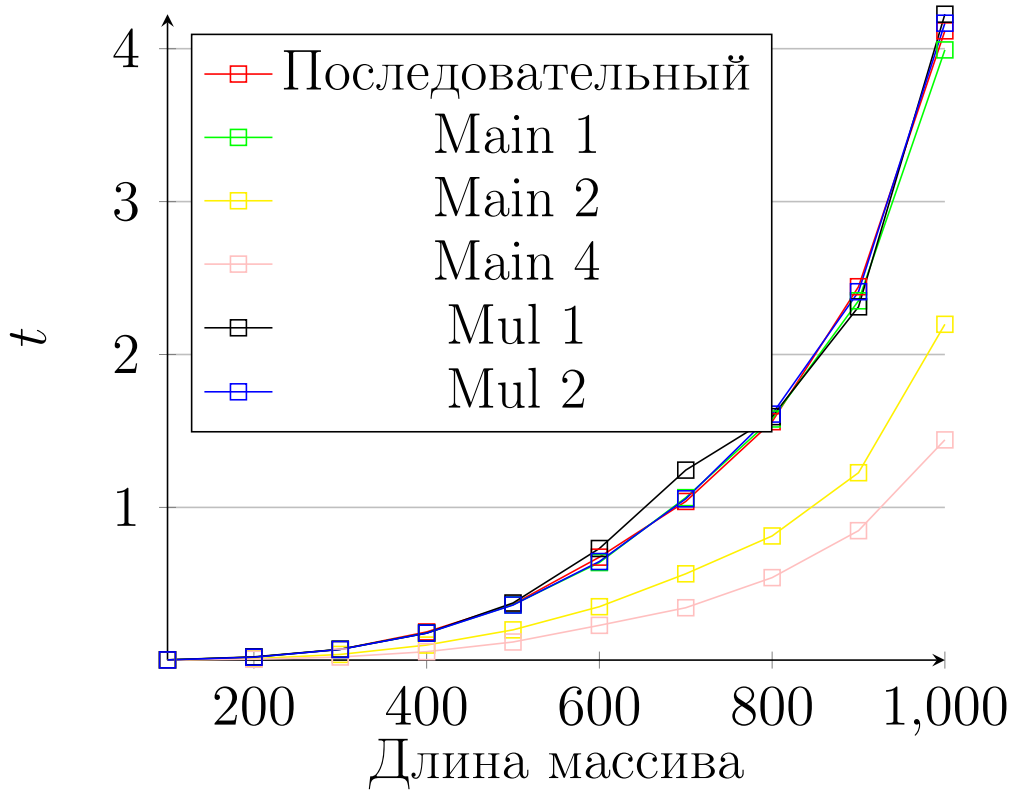

Рисунок 4.3 — Сравнение времени работы алгоритма Винограда и его параллельных реализаций

Из рисунка 4.3 становится ясно, что наиболее эффективным является умножение матриц алгоритмом Винограда с параллельным главным циклом, в то время как использование параллельных вычислений при подсчёте массивов mulV и mulH не даёт результатов по сравнению с последовательной реализацией.

Вывод

В итоге, можно сказать о том, что параллельное вычисление главного цикла в алгоритме Винограда является самым эффективным способом уменьшения затрачиваемого времени из рассмотренных.

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы цель была достигнута – параллельные вычисления были изучены возможности и использованы на практике. Все поставленные задачи были выполнены: реализован алгоритм Винограда, разработаны и реализованы 2 версии его параллллельной реализации, проведены замеры времени, проведен сравнительный анализ.

Было определено, что наиболее эффективным по времени оказался алгоритм Винограда с параллельным вычислением главного цикла, а также установлено, что использование более 4 потоков не имеет смысла.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. И.В. Белоусов. Матрицы и определители: учебное пособие по линейной алгебре. — 2006. — 101 с.

2. Умножение матриц[Электронный ресурс]. — 2005. — Режим доступа: http://algolib.narod.ru/Math/Matrix.html (дата обращения: 2.10.2020).

3. Алексеев Александр. Академия Microsoft: Основы параллельного программирования с использованием Visual Studio 2010[Электронный ресурс]. — 2016. — Режим доступа: https://intuit.ru/studies/courses/4807/1055/lecture/16369 (дата обращения: 2.10.2020).

4. Varun. C++11 Multithreading [Электронный ресурс]. — 2015. — Режим доступа: https://thispointer.com//c-11-multithreading-part-1-three-different-ways-to-create-threads/ (дата обращения: 2.10.2020).