МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Н. Э. БАУМАНА

УДК _____
№ госрегистрации _____
Инв. №_____

УТВЕРЖДАЮ

Преподаватель

_____

«_____» _____ 2020 г.

ДИСЦИПЛИНА АНАЛИЗ АЛГОРИТМОВ

ОТЧЁТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

Конвейеризация вычислений для алгоритма Винограда
(промежуточный)

Студент _____ Ф.М. Набиев

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

Москва, 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Имеется большое количество важнейших задач, решение которых требует использования огромных вычислительных мощностей, зачастую недоступных для современных вычислительных систем.

Постоянно появляются новые задачи подобного рода и возрастают требования к точности и к скорости решения прежних задач; поэтому вопросы разработки и использования сверхмощных компьютеров (называемых суперкомпьютерами) актуальны сейчас и в будущем [3]. Но пока эти трудности пока что не удается преодолеть. Из-за этого приходится и эти по пути создания параллельных вычислительных систем, т.е. систем, в которых предусмотрена одновременная реализация ряда вычислительных процессов, связанных с решением одной задачи. На современном этапе развития вычислительной техники такой способ, по-видимому, является одним из основных способов ускорения вычислений.

Многие явления природы характеризуются параллелизмом (одновременным исполнением процессов с применением различных путей и способов). В частности, в живой природе параллелизм распространен очень широко в дублирующих системах для получения надежного результата. Параллельные процессы пронизывают общественные отношения, ими характеризуются развитие науки, культуры и экономики в человеческом обществе. Среди этих процессов особую роль играют параллельные информационные потоки [5].

Среди параллельных систем различают конвейерные, векторные, матричные, систолические, спецпроцессоры и т.п. В данной работе используются конвейерные.

В данной работе стоит задача реализации алгоритма Винограда для умножения матриц, сравнение последовательной и конвейерное реализаций.

1 Аналитический раздел

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

1.1 Описание задачи

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.

1.1.2 Конвейеризация

Конвейеризация – это техника, в результате которой задача или команда разбивается на некоторое число подзадач, которые выполняются последовательно. Каждая подкоманда выполняется на своем логическом устройстве. Все логические устройства (ступени) соединяются последовательно таким образом, что выход i-ой ступени связан с входом $(i+1)$-ой ступени, все ступени работают одновременно. Множество ступеней называется конвейером. Выигрыш во времени достигается при выполнении нескольких задач за счет параллельной работы ступеней, вовлекая на каждом такте новую задачу или команду [5].

В конвейере различают r последовательных этапов, так что когда i-я операция проходит s-й этап, то $(i+k)$-я операция проходит $(s-k)$-й этап.

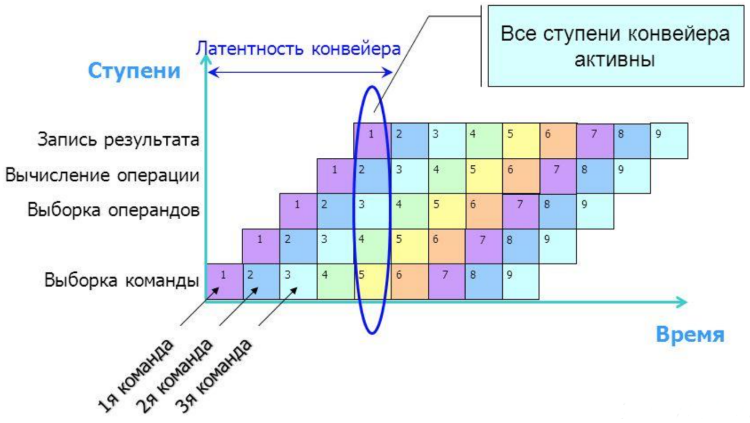

Рисунок 1.1 — Работа конвейера

1.2 Вывод

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

2 Конструкторский раздел

В данном разделе будет произведена разработка конвейера умножения матриц по алгоритму 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 конвейера умножения матриц по этому алгоритму.

2.2.1 Алгоритм Винограда

На риснуках 2.2, 2.3 изображена схема алгоритма Винограда.

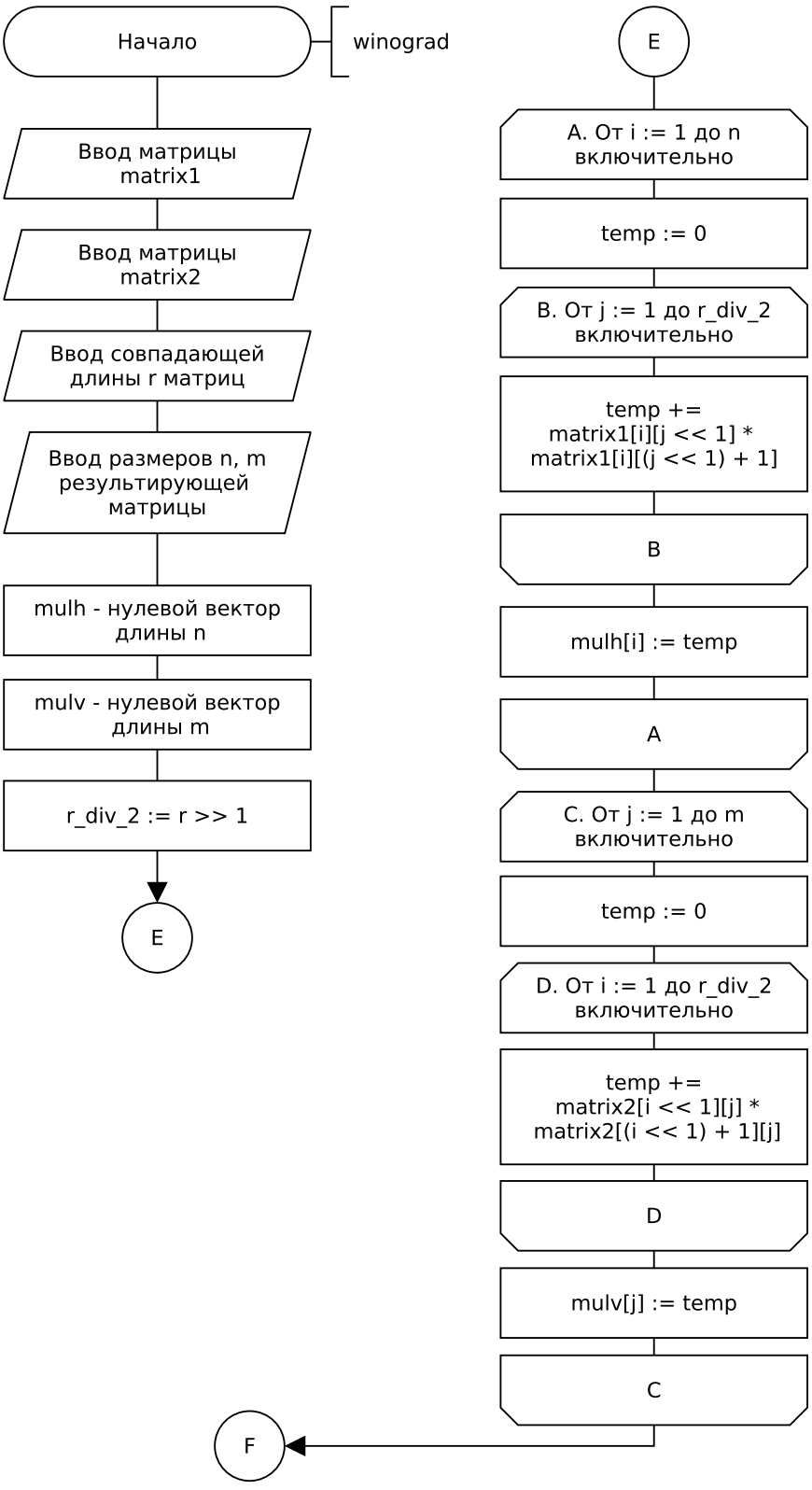

Рисунок 2.2 — Алгоритм Винограда, часть 1

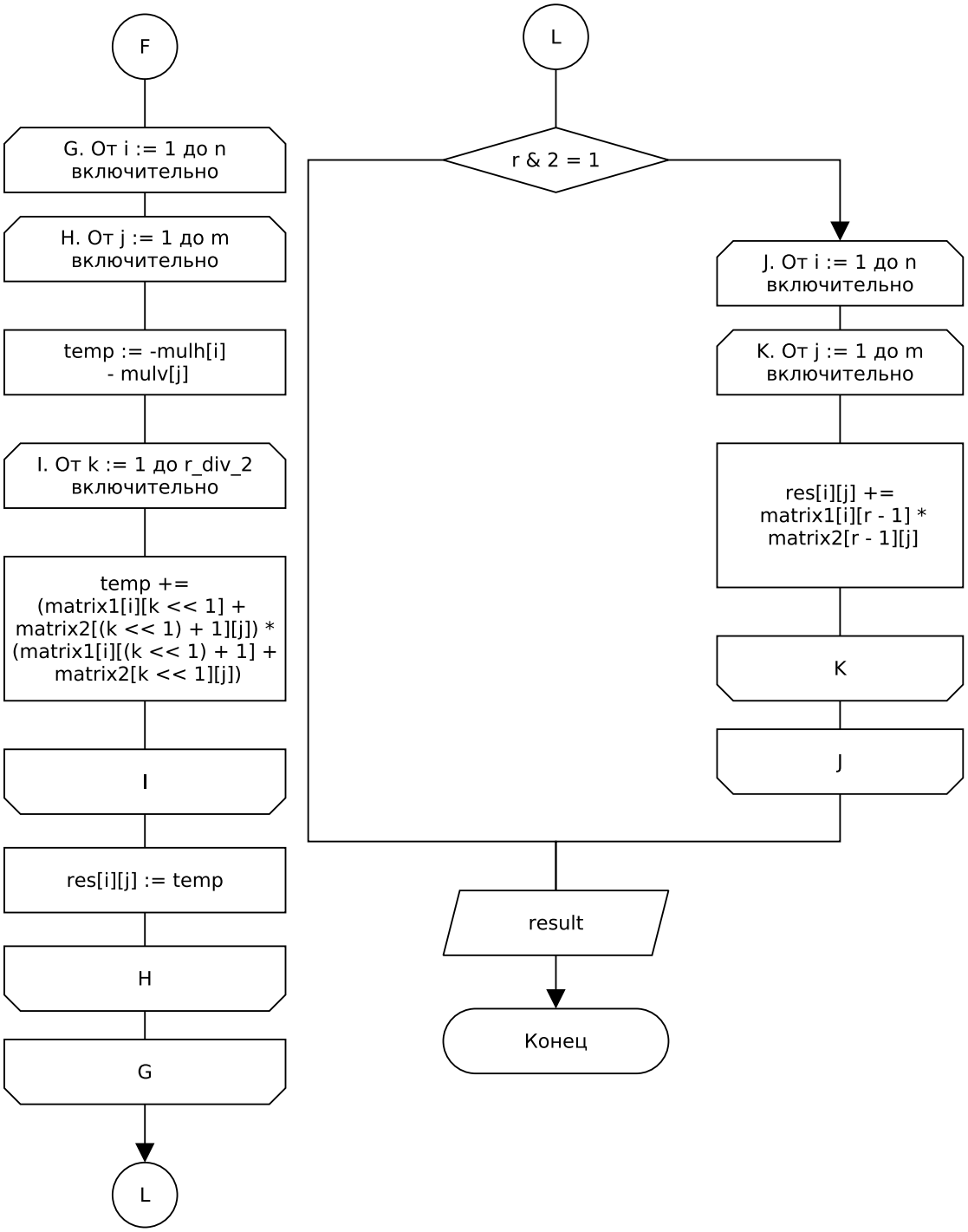

Рисунок 2.3 — Алгоритм Винограда, часть 2

2.2.2 Конвейер

Для того, чтобы распараллелить алгоритм Винограда по линиям конвейера выделим следующие этапы:

а) ввод матриц, выделение необходимой для вычислений памяти;

б) вычисление вектора mulh;

в) вычисление вектора mulv;

г) вычисление первой половины матрицы-произведения;

д) вычисление второй половины матрицы-произведения;

е) проверка чётности и вычисления оставшейся части матрицы.

На рисунке 2.4 приведена схема конвейера, вычисляющего произведение матриц по алгоритму Винограда.

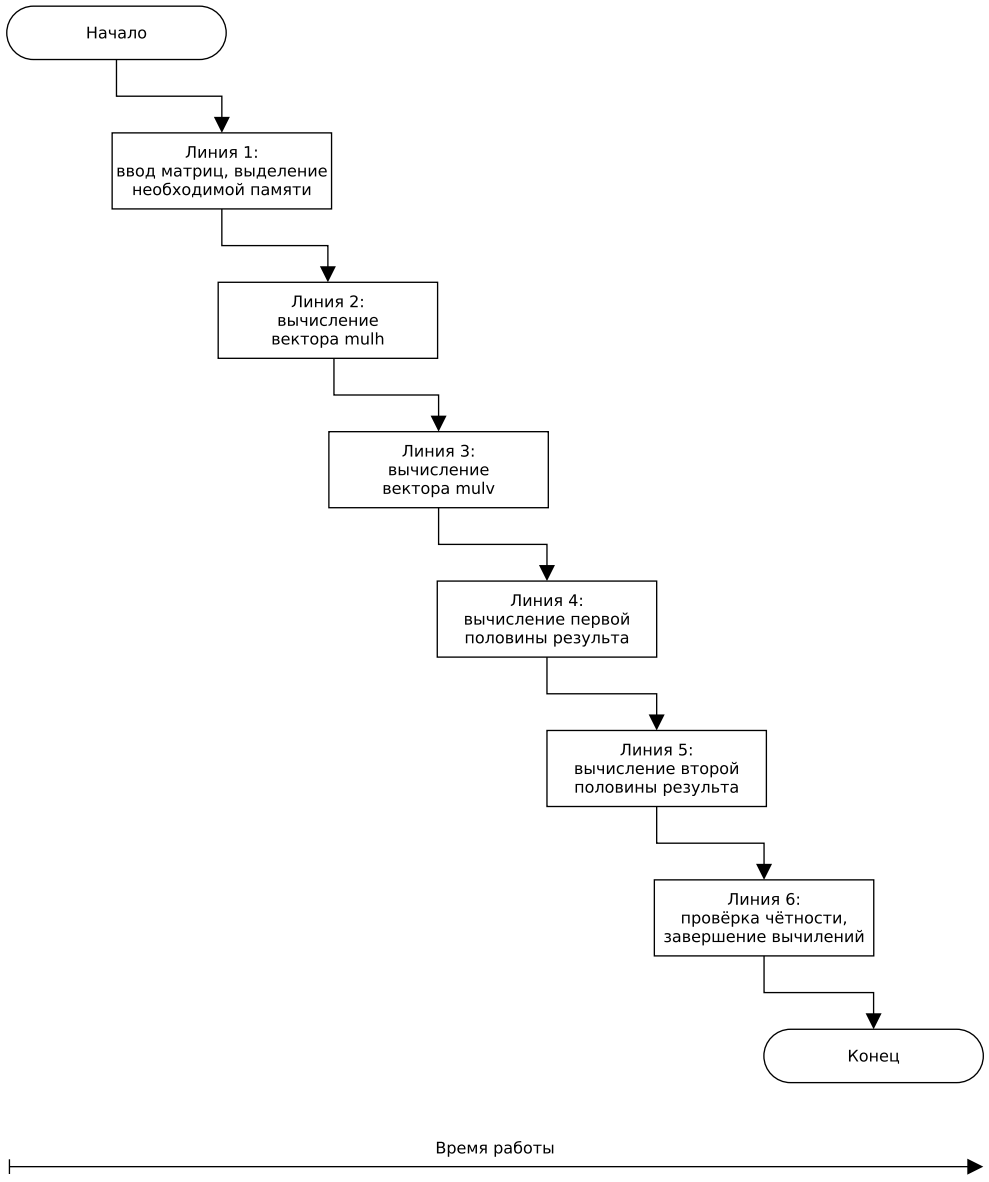

Рисунок 2.4 — Схема конвейера перемножения матриц по алгоритму Винограда

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:

— $n \in \mathbb{N}$ - 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 [1].

— Встроенная библиотека std::thread, содержащая класс нативных потоков [2].

3.3 Листинги кода

В листингах 3.1, 3.2, 3.3 приведен текст структуры и методов, описывающих умножение матриц по алгоритму Винограда.

Листинг 3.1 — Алгоритм Винограда, ч. 1

```
1  struct MatrixMul
2  {
3      MatrixMul(double **m1, double **m2, size_t n, size_t m, size_t r);
4
5      double **matrix1;
6      double **matrix2;
7
8      size_t n;
9      size_t m;
10     size_t r;
11
12     double *mulh;
13     double *mulv;
14     double **res;
15
16     void allocateMemory();
17     void calculateMulh();
18     void calculateMulv();
19     void calculatePart1();
20     void calculatePart2();
21     void calculateUneven();
22 };
```

Листинг 3.2 — Алгоритм Винограда, ч. 2

```
1  MatrixMul::MatrixMul(double **m1, double **m2,
2                       size_t n, size_t m, size_t r) :
3      matrix1(m1), matrix2(m2), n(n), m(m), r(r),
4      mulh(nullptr), mulv(nullptr), res(nullptr)
5  {
6  }
7
8  void MatrixMul::allocateMemory()
9  {
10     mulh = new double[n];
11     mulv = new double[m];
12
```

```
13      res = Util::createMatrix<double>(n, m);
14  }
15
16  void MatrixMul::calculateMulh()
17  {
18      double temp;
19      size_t r_div_2 = r >> 1;
20
21      for (size_t i = 0; i < n; i++)
22      {
23          temp = 0;
24
25          for (size_t j = 0; j < r_div_2; j++)
26          {
27              temp += matrix1[i][j << 1] * matrix1[i][(j << 1) + 1];
28          }
29
30          mulh[i] = temp;
31      }
32  }
33
34  void MatrixMul::calculateMulv()
35  {
36      double temp;
37      size_t r_div_2 = r >> 1;
38
39      for (size_t j = 0; j < m; j++)
40      {
41          temp = 0;
42
43          for (size_t i = 0; i < r_div_2; i++)
44          {
45              temp += matrix2[i << 1][j] * matrix2[(i << 1) + 1][j];
46          }
47
48          mulv[j] = temp;
49      }
50  }
```

Листинг 3.3 — Алгоритм Винограда, ч. 3

```
1  void MatrixMul::calculatePart1()
2  {
3      double temp;
4
```

```
 5       size_t r_div_2 = r >> 1;
 6       size_t n_div_2 = n >> 1;
 7
 8       for (size_t i = 0; i < n_div_2; i++)
 9       {
10           for (size_t j = 0; j < m; j++)
11           {
12               temp = -(mulh[i] + mulv[j]);
13
14               for (size_t k = 0; k < r_div_2; k++)
15               {
16                   temp += (matrix1[i][k << 1] + matrix2[(k << 1) + 1][j]) *
17                           (matrix1[i][(k << 1) + 1] + matrix2[k << 1][j]);
18               }
19
20               res[i][j] = temp;
21           }
22       }
23   }
24
25   void MatrixMul::calculatePart2()
26   {
27       double temp;
28
29       size_t r_div_2 = r >> 1;
30       size_t n_div_2 = n >> 1;
31
32       for (size_t i = n_div_2; i < n; i++)
33       {
34           for (size_t j = 0; j < m; j++)
35           {
36               temp = -(mulh[i] + mulv[j]);
37
38               for (size_t k = 0; k < r_div_2; k++)
39               {
40                   temp += (matrix1[i][k << 1] + matrix2[(k << 1) + 1][j]) *
41                           (matrix1[i][(k << 1) + 1] + matrix2[k << 1][j]);
42               }
43
44               res[i][j] = temp;
45           }
46       }
47   }
48
49   void MatrixMul::calculateUneven()
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
61      mulh = nullptr;
62      mulv = nullptr;
63  }
```

В листингах 3.4, 3.5 приведен текст структуры и методов, описывающих конвейер умножения матриц по алгоритму Винограда.

Листинг 3.4 — Конвейер перемножения матриц по алгоритму Винограда, ч. 1

```
1   class MatrixMulReader
2   {
3   public:
4       virtual struct MatrixMul operator()() = 0;
5       virtual operator bool() = 0;
6   };
7
8   class Conveyor
9   {
10  public:
11      static
12      void do1(MatrixMulReader *mmr)
13      {
14          while (*mmr)
15          {
16              auto mm = (*mmr)();
17              mm.allocateMemory();
18
19              m12.lock();
20              q12.push(mm);
21              m12.unlock();
22          }
23
24          isP1Ended = true;
25      }
26      static
```

```
27      void do2()
28      {
29          while (!q12.empty() || !isP1Ended)
30          {
31              if (!q12.empty())
32              {
33                  m12.lock();
34                  struct MatrixMul mm = q12.front();;
35                  q12.pop();
36                  m12.unlock();
37
38                  mm.calculateMulh();
39
40                  m23.lock();
41                  q23.push(mm);
42                  m23.unlock();
43              }
44          }
45
46          isP1Ended = false;
47          isP2Ended = true;
48      }
49      static
50      void do3()
51      {
52          while (!q23.empty() || !isP2Ended)
53          {
54              if (!q23.empty())
55              {
56                  m23.lock();
57                  struct MatrixMul mm = q23.front();
58                  q23.pop();
59                  m23.unlock();
60
61                  mm.calculateMulv();
62
63                  m34.lock();
64                  q34.push(mm);
65                  m34.unlock();
66              }
67          }
68
69          isP2Ended = false;
70          isP3Ended = true;
71      }
```

Листинг 3.5 — Конвейер перемножения матриц по алгоритму Винограда, ч. 2

```
1       static
2       void do4()
3       {
4           while (!q34.empty() || !isP3Ended)
5           {
6               if (!q34.empty())
7               {
8                   m34.lock();
9                   struct MatrixMul mm = q34.front();
10                  q34.pop();
11                  m34.unlock();
12
13                  mm.calculatePart1();
14
15                  m45.lock();
16                  q45.push(mm);
17                  m45.unlock();
18              }
19          }
20
21          isP3Ended = false;
22          isP4Ended = true;
23      }
24      static
25      void do5()
26      {
27          while (!q45.empty() || !isP4Ended)
28          {
29              if (!q45.empty())
30              {
31                  m45.lock();
32                  struct MatrixMul mm = q45.front();
33                  q45.pop();
34                  m45.unlock();
35
36                  mm.calculatePart2();
37
38                  m56.lock();
39                  q56.push(mm);
40                  m56.unlock();
41              }
42          }
43
44          isP4Ended = false;
```

```
45          isP5Ended = true;
46      }
47      static
48      void do6()
49      {
50          while (!q56.empty() || !isP5Ended)
51          {
52              if (!q56.empty())
53              {
54                  m56.lock();
55                  struct MatrixMul mm = q56.front();
56                  q56.pop();
57                  m56.unlock();
58
59                  mm.calculateUneven();
60
61                  q67.push(mm);
62              }
63          }
64
65          isP5Ended = false;
66      }
67
68      static std::queue<struct MatrixMul> q12;
69      static std::queue<struct MatrixMul> q23;
70      static std::queue<struct MatrixMul> q34;
71      static std::queue<struct MatrixMul> q45;
72      static std::queue<struct MatrixMul> q56;
73      static std::queue<struct MatrixMul> q67;
74
75      static std::mutex m12;
76      static std::mutex m23;
77      static std::mutex m34;
78      static std::mutex m45;
79      static std::mutex m56;
80      static std::mutex m67;
81
82      static bool isP1Ended;
83      static bool isP2Ended;
84      static bool isP3Ended;
85      static bool isP4Ended;
86      static bool isP5Ended;
87      static bool isP6Ended;
88      static bool isP7Ended;
89  };
```

3.4 Примеры работы

На рисунках 3.1-3.5 приведены примеры работы.

```
 ~/Documents/Repositories/bmstu/AlgoesAnalysis/lab_05/build  master ! ?
# ./fn_aa_lab_05
```

Рисунок 3.1 — Пустой ввод

```
  ~/Documents/Repositories/bmstu/AlgoesAnalysis/lab_05/build  master ! ?
 # ./fn_aa_lab_05
0
```

Рисунок 3.2 — Невозможная длина матрицы

```
  ~/Documents/Repositories/bmstu/AlgoesAnalysis/lab_05/build  master ! ?
 # echo 2 | ./gen | ./fn_aa_lab_05
Matrix len: 2
 3.000 10.000
 4.000  3.000

 5.000 15.000
18.000 16.000

195.000 205.000
74.000 108.000
```

Рисунок 3.3 — Умножения квадратных матриц длины 2

```
  ~/Documents/Repositories/bmstu/AlgoesAnalysis/lab_05/build  master ! ?
 # echo 5 | ./gen | ./fn_aa_lab_05
Matrix len: 5
15.000  3.000 11.000  7.000 13.000
18.000 15.000  8.000 18.000  1.000
 4.000  9.000 12.000  2.000  5.000
 3.000 17.000 17.000  0.000  3.000
16.000 14.000  5.000  6.000  6.000

19.000 10.000  2.000 14.000  8.000
10.000 12.000  7.000 14.000  2.000
 0.000  7.000 14.000 19.000  8.000
 6.000 19.000 12.000  9.000  8.000
19.000 10.000 15.000 12.000 19.000

604.000 526.000 484.000 680.000 517.000
619.000 768.000 484.000 788.000 401.000
273.000 320.000 338.000 488.000 257.000
284.000 383.000 408.000 639.000 251.000
594.000 537.000 362.000 641.000 358.000
```

Рисунок 3.4 — Умножения квадратных матриц длины 5

```
 ~/Documents/Repositories/bmstu/AlgoesAnalysis/lab_05/build  master ! ?
 # echo 10 | ./gen | ./fn_aa_lab_05
Matrix len: 10
17.000  8.000 14.000  9.000  2.000 14.000  1.000 19.000 15.000 14.000
19.000 12.000  2.000  6.000 18.000  0.000 17.000 11.000  8.000 17.000
 5.000  7.000  5.000  8.000 16.000 13.000 10.000  1.000 14.000 15.000
 0.000 10.000 15.000  2.000 12.000  1.000  6.000  6.000 11.000  5.000
 1.000  2.000 17.000  1.000 19.000  8.000 19.000 17.000 17.000 16.000
12.000  3.000 11.000  6.000  5.000 13.000 16.000  1.000  0.000  3.000
14.000  1.000  0.000  1.000  4.000 15.000  4.000  9.000 10.000 19.000
10.000 15.000 18.000 10.000  2.000 17.000 19.000 15.000 15.000 18.000
10.000  6.000 13.000 17.000  1.000 14.000 13.000  0.000 19.000 10.000
12.000  8.000 13.000 16.000 19.000 16.000  1.000  2.000 15.000  9.000

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

1279.000 917.000 821.000 833.000 822.000 979.000 1220.000 1318.000 1163.000 959.000
1187.000 702.000 665.000 581.000 751.000 1319.000 1070.000 1135.000 966.000 1162.000
1162.000 768.000 703.000 736.000 463.000 1093.000 1016.000 985.000 874.000 1017.000
771.000 430.000 498.000 600.000 335.000 687.000 694.000 670.000 795.000 543.000
1571.000 648.000 803.000 902.000 526.000 1325.000 1244.000 1096.000 1247.000 886.000
927.000 462.000 487.000 473.000 432.000 745.000 610.000 471.000 552.000 643.000
969.000 664.000 535.000 516.000 574.000 785.000 912.000 922.000 710.000 853.000
1686.000 1091.000 1086.000 1008.000 872.000 1430.000 1420.000 1447.000 1398.000 1162.000
1296.000 927.000 830.000 870.000 541.000 986.000 850.000 1042.000 799.000 868.000
1263.000 923.000 825.000 919.000 592.000 1111.000 1164.000 1150.000 1054.000 1157.000
```

Рисунок 3.5 — Умножения квадратных матриц 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 конвейерного 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

Данный эксперимент состоит в перемножении определенного количества квадратных матриц одинакового размера. Результаты изображены на графиках зависимости времени выполнения от количества матриц.

На рисунке 4.1 приведен график сравнения последовательного умножения матриц и конвейерного. Размер матриц равен 100.

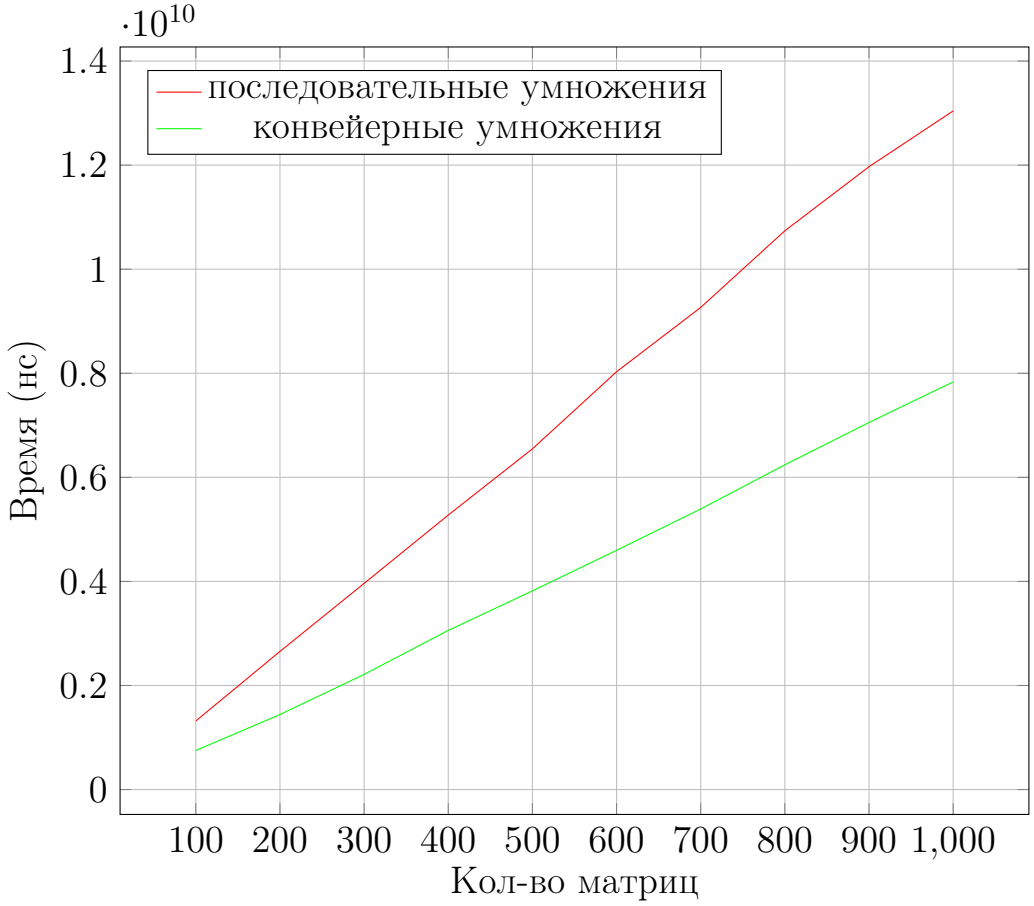

Рисунок 4.1 — Чётная длина, сравнение последовательных и конвейерных вычислений

На рисунке 4.2 приведен график сравнения последовательного умножения матриц и конвейерного. Размер матриц равен 101.

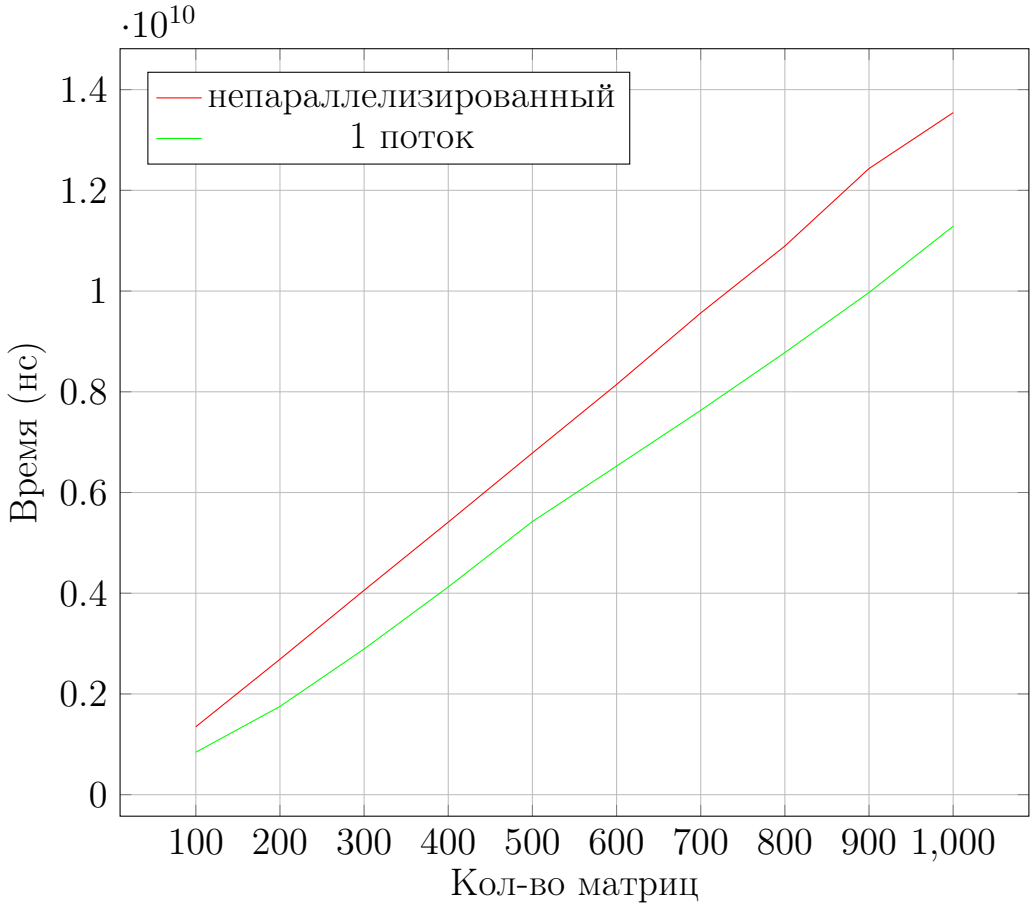

Рисунок 4.2 — Нечётная длина, сравнение последовательных и конвейерных вычислений

Эксперимент ставился пять раз, в графиках приведены средние значения времени.

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 полученных графиков, можно заключить, что конвейерные вычисления дают прирост в скорости по сравнению с последовательными.

ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы был реализован алгоритм Винограда, разработана и реализована конвейерная версия алгоритма Винограда, которая в ходе эксперимента по замеру времени показала прирост в скорости выполнения по сравнению с последовательной реализацией.

Таким образом, справедливо сделать вывод, что конвейерные вычисления работают быстрее последовательных. Выигрыш во времени достигается при выполнении нескольких задач за счет параллельной работы ступеней, вовлекая на каждом такте новую задачу или команду.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация по chrono [Электронный ресурс]. - Режим доступа: http://www.cplusplus.com/reference/chrono/ Дата обращения: 25.10.2019

2. Документация по thread [Электронный ресурс]. - Режим доступа: https://ru.cppreference.com/w/cpp/thread/thread Дата обращения: 25.10.2019

3. Воеводин В.В. Математические модели и методы в параллельных процессах. М., 1986. 296 с.

4. Yukiya Aoyama, Jun Nakano. RS/6000 SP:Practical MPI Programming. IBM. Technical Support Organization., 2000. 221

5. Конвеерные вычисления [Электронный ресурс]. - Режим доступа: http://www.myshared.ru/slide/674082/(дата обращения: 03.12.2019)