Лабораторная работа №4
По курсу: "Анализ алгоритмов"
По теме: "Параллельное умножение матриц"

Студент: Кондрашова О.П.
Группа: ИУ7-55Б
Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Содержание

Введение

Целью данной работы является изучение и реализация параллельного алгоритма Винограда для умножения матриц. Необходимо сравнить зависимость времени работы алгоритма от числа параллельных потоков исполнения и размера матриц, провести сравнение стандартного и параллельного алгоритма.

1 Аналитическая часть

В данной части будут рассмотрены теоретические основы алгоритмов.

1.1 Описание алгоритмов

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. 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 — квадратные матрицы одного и того же порядка. Таким образом, из существования произведения AB вовсе не следует существование произведения BA.

Эти алгоритмы активно применяются во всех областях, применяющие линейную алгебру, такие как:

1. экономика;
2. физика;
3. компьютерная графика.

1.2 Алгоритм Винограда

Очевидно, что каждый $c_{i,j}$ в результирующей матрице является скалярным произведением соответствующих столбца и строки. Такое умножение допускает предварительное вычисление части результата. Имеются два вектора: $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$.

Стандартное» перемножение:
$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4$

Перемножение по Винограду:
$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4$

Данный алгоритм позволяет выполнить предварительную обработку матрицы и запомнить значения для каждой строки/столбца матриц.

Над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.3 Параллельный алгоритм Винограда

Трудоемкость алгоритма Винограда имеет сложность $O(nmk)$ для умножения матриц $n1 \times m1$ на $n2 \times m2$. Чтобы улучшить алгоритм, следует распараллелить ту часть алгоритма, которая содержит 3 вложенных цикла.

Вычисление результата для каждой строки не зависит от результата выполнения умножения для других строк. Поэтому можно распараллелить часть кода, где происходят эти действия. Каждый поток будет выполнять вычисления определенных строк результирующей матрицы.

1.4 Распараллеливание задачи на CPU

В рамках данной лабораторной работы производилось распараллеливание задачи по потокам. В CPU для данной цели используются treads.

CPU – central processing unit – это универсальный процессор, также именуемый процессором общего назначения. Он оптимизирован для достижения высокой производительности единственного потока команд. Доступ к памяти с данными и инструкциями происходит преимущественно случайным образом.

Для того, чтобы повысить производительность CPU еще больше, они проектируются специально таким образом, чтобы выполнять как можно больше инструкций параллельно. Например для этого в ядрах процессора используется блок внеочередного выполнения команд.

Но несмотря на это, CPU все равно не в состоянии осуществить параллельное выполнение большого числа инструкций, так как расходы на распараллеливание инструкций внутри ядра оказываются очень существенными.

Именно поэтому процессоры общего назначения имеют не очень большое количество исполнительных блоков.

1.5 Вывод

Были рассмотрены поверхностно стандартная и параллельная реализации алгоритма Винограда.

2 Конструкторская часть

В данной части будут представлены схемы алгоритмов.

2.1 Схемы алгоритмов

На рис. 1 представлена схема стандартного алгоритма Винограда:

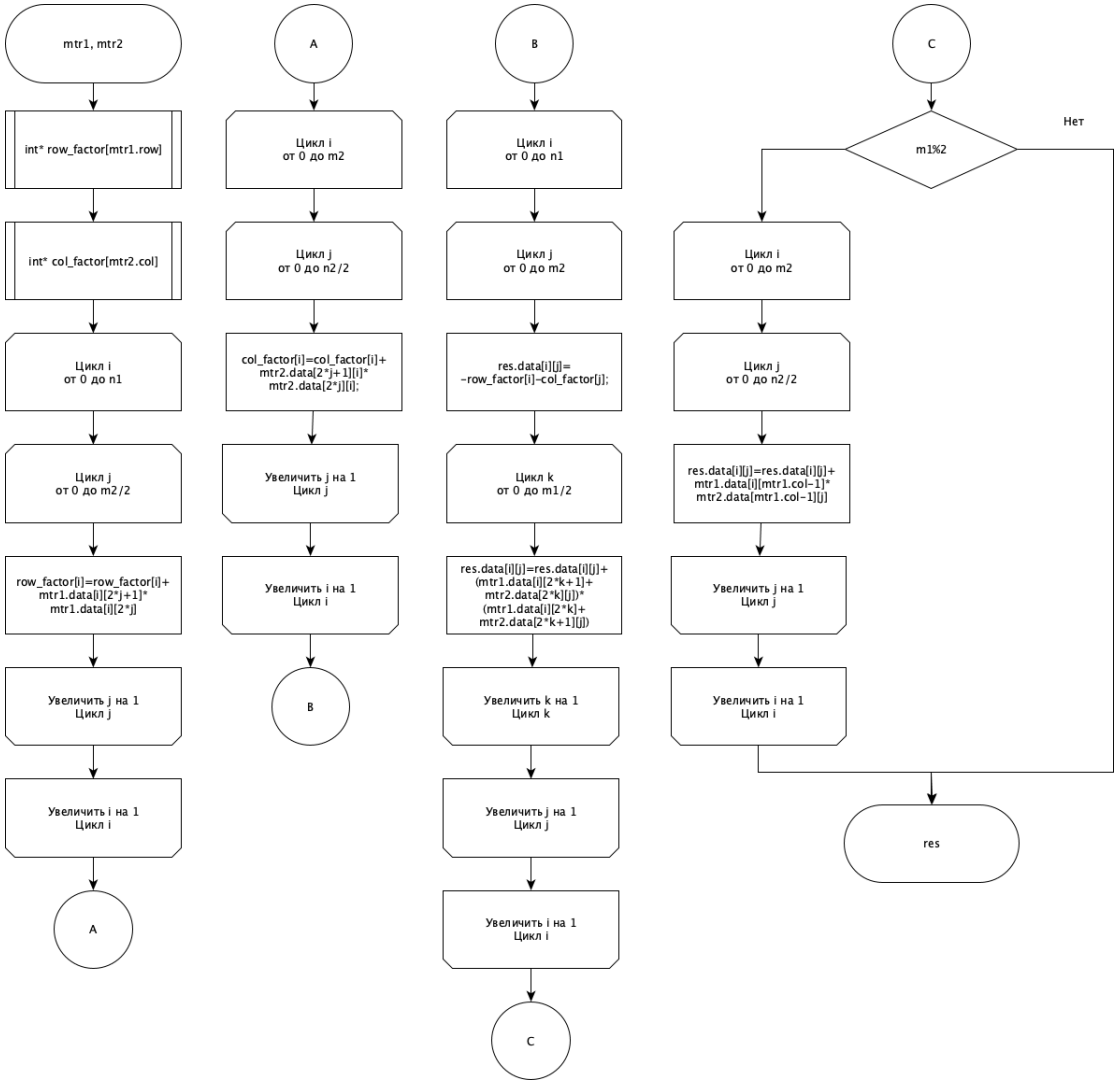

Рис. 1: Стандартный алгоритм Винограда

На рис. 2 представлена схема стандартного алгоритма Винограда:

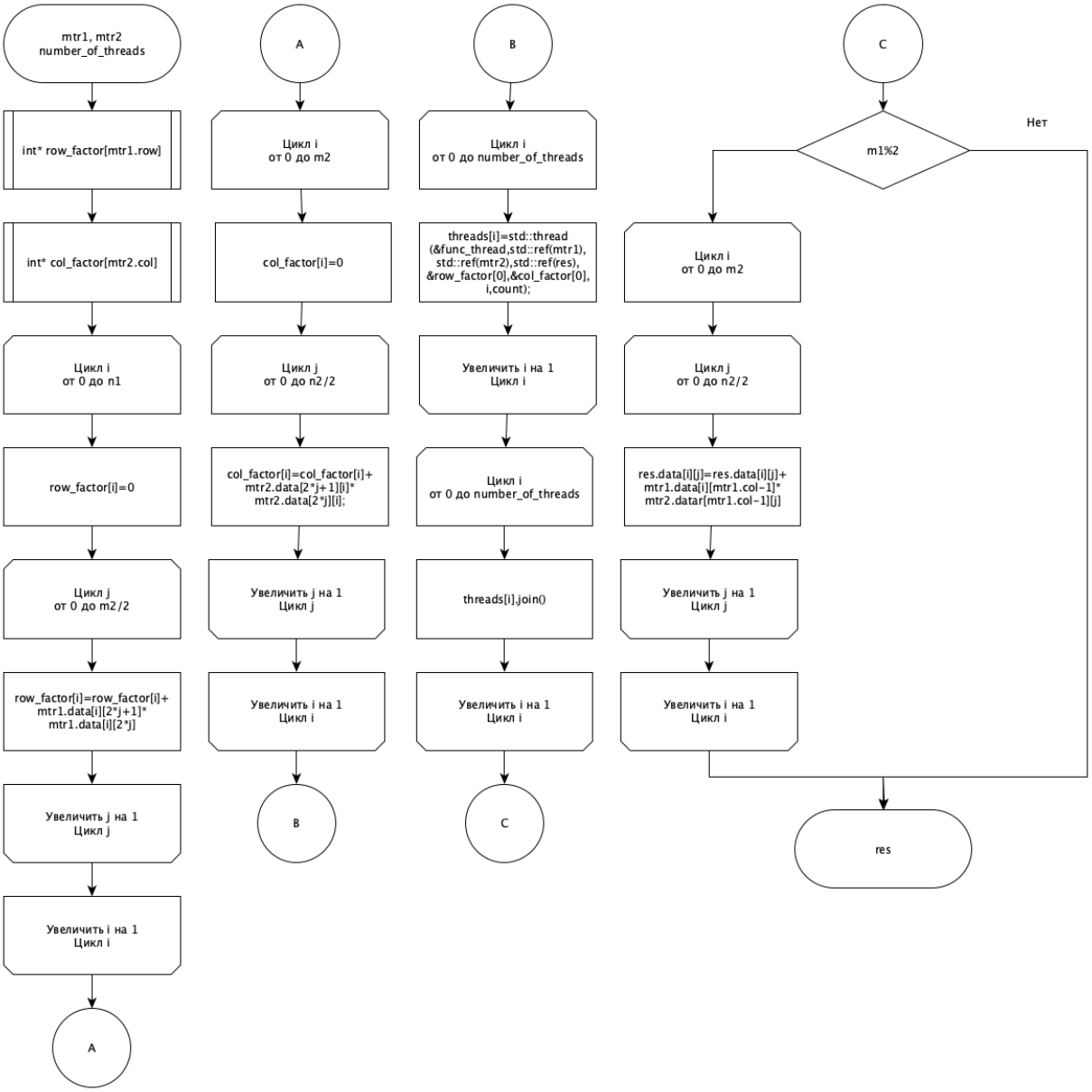

Рис. 2: Параллельный алгоритм Винограда

На рис. 3 представлена схема фнкции параллельного вычисления матрицы:

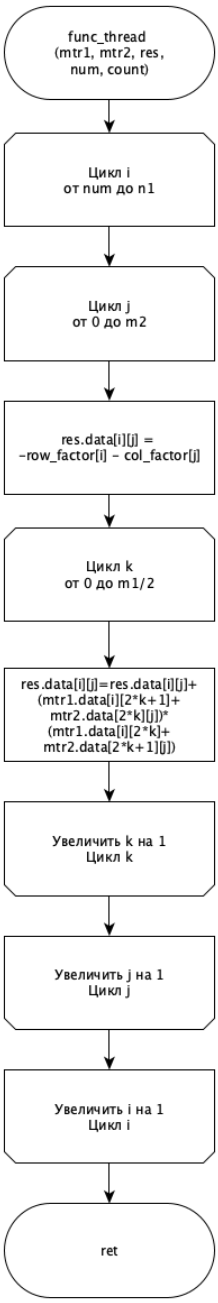

Рис. 3: Параллельное вычисление матрицы

2.2 Вывод

В данной части были рассмотрены схемы алгоритмов.

3 Технологическая часть

В этом разделе будут изложены требования к программному обеспечению и листинги алгоритмов.

3.1 Требования к программному обеспечению

Входные данные: matr1 - первая матриц, matr2 - вторая матрица. Выходные данные: произведение двух матриц.

На рис. 4 представлена IDEF0 диаграмма умножения двух матриц:

Рис. 4: IDEF0 диаграмма умножения двух матриц

3.2 Средства реализации

Данная программа разработана на языке C++, поддерживаемом многими операционными системами. Проект выполнен в среде Xcode.

Для отключения оптимизации компилятора указан флаг «-o0».

Для работы с потоками используется библиотека поддержки потоков std::thread.
Наблюдатель joinable проверяет потенциальную возможность работы потока в параллельном контексте.
Операция join ожидает завершения потока.

3.3 Листинги функций

В данном разделе представлены листинги реализованных алгоритмов.

В листинге 1 представлен стандартный алгоритм Винограда:

Листинг 1: Стандартный алгоритм Винограда

```
Matrix Matrix::vinograd_mult(const Matrix &mtr1, const Matrix
    &mtr2)
{
    Matrix res(mtr1.row, mtr2.column);

    int row_factor[mtr1.row];
    int col_factor[mtr2.column];

    for(int i = 0; i < mtr1.row; i++)
    {
        row_factor[i] = 0;
        for(int j = 0; j < mtr1.column / 2; j++)
        {
            row_factor[i] = row_factor[i] + mtr1.data[i][2 * j
                ] * mtr1.data[i][2 * j + 1];
        }
    }

    for(int i = 0; i < mtr2.column; i++)
    {
        col_factor[i] = 0;
        for(int j = 0; j < mtr2.row / 2; j++)
        {
            col_factor[i] = col_factor[i] + mtr2.data[2 * j][i
                ] * mtr2.data[2 * j + 1][i];
        }
    }

    for(int i = 0; i < mtr1.row; i++)
    {
        for(int j = 0; j < mtr2.column; j++)
        {
            res.data[i][j] = -row_factor[i] - col_factor[j];
            for(int k = 0; k < mtr1.column / 2; k++)
            {
                res.data[i][j] = res.data[i][j] +
                                 (mtr1.data[i][2 * k + 1] +
                                      mtr2.data[2 * k][j])
                                     *
                                 (mtr1.data[i][2 * k] +
                                     mtr2.data[2 * k + 1][j
                                     ]);
            }
        }
    }

    if(mtr1.column % 2)
    {
```

```
42          for(int i = 0; i < mtr1.row; i++)
43              for(int j = 0; j < mtr2.column; j++)
44                  res.data[i][j] = res.data[i][j] + mtr1.data[i
                        ][mtr1.column - 1] * mtr2.data[mtr1.column
                         - 1][j];
45      }
46
47      return res;
48  }
```

В листинге 2 представлен параллельный алгоритм Винограда:

Можно заметить, что вычисление результата для каждой строки происходит независимо от результата выполнения умножения для других строк. Поэтому возможно распараллелить участок кода, соответстующий строкам 24-32 листинга 1. Каждый поток будет выполнять вычисление некоторых строк результирующей матрицы. Это сделано потому, что проход по строкам матрицы является более эффективным с точки зрения организации данных в памяти.

Листинг 2: Параллельный алгоритм Винограда

```
1  Matrix Matrix::vinograd_mult_parallel(const Matrix &mtr1,
       const Matrix &mtr2, int count)
2  {
3      Matrix res(mtr1.row, mtr2.column);
4
5      int row_factor[mtr1.row];
6      int col_factor[mtr2.column];
7
8      for(int i = 0; i < mtr1.row; i++)
9      {
10         row_factor[i] = 0;
11         for(int j = 0; j < mtr1.column / 2; j++)
12         {
13             row_factor[i] = row_factor[i] + mtr1.data[i][2 * j
                   + 1] * mtr1.data[i][2 * j];
14         }
15     }
16
17     for(int i = 0; i < mtr2.column; i++)
18     {
19         col_factor[i] = 0;
20         for(int j = 0; j < mtr1.column / 2; j++)
21         {
22             col_factor[i] = col_factor[i] + mtr2.data[2 * j +
                   1][i] * mtr2.data[2 * j][i];
23         }
24     }
25
26     std::thread threads[count];
27
28     for(int i = 0; i < count; i++)
29     {
```

```
30          threads[i] = std::thread(&func_thread, std::ref(mtr1),
                 std::ref(mtr2), std::ref(res), &row_factor[0], &
                col_factor[0], i, count);
31      }
32      for(int i = 0; i < count; i++)
33      {
34          if(threads[i].joinable())
35          {
36              threads[i].join();
37          }
38      }
39
40
41      if(mtr1.column % 2 == 1)
42      {
43          for(int i = 0; i < mtr1.row; i++)
44              for(int j = 0; j < mtr2.column; j++)
45                  res.data[i][j] = res.data[i][j] + mtr1.data[i
                        ][mtr1.column - 1] * mtr2.data[mtr1.column
                         - 1][j];
46      }
47
48      return res;
49  }
```

В листинге 3 представлено распараллеленное вычисление тройного цикла:

Листинг 3: Распараллеленный тройной цикл

```
1   void func_thread(const Matrix &mtr1, const Matrix &mtr2,
        Matrix &res, int* row_factor, int* col_factor, int num,
        int count)
2   {
3       for(int i = num; i < mtr1.row; i += count)
4       {
5           for(int j = 0; j < mtr2.column; j++)
6           {
7               res.data[i][j] = -row_factor[i] - col_factor[j];
8               for(int k = 0; k < mtr1.column / 2; k++)
9               {
10                  res.data[i][j] = res.data[i][j] + (mtr1.data[i
                        ][2*k+1] + mtr2.data[2*k][j]) *
11                          (mtr1.data[i][2*k] + mtr2.data[2*k+1][
                                j]);
12              }
13          }
14      }
15  }
```

3.4 Вывод

В данном разделе были представлены стандартный и параллельный алгоритмы Винограда.

4 Исследовательская часть

В данной части представлены результаты исследования быстродействия алгоритмов.

4.1 Постановка эксперимента

Были проведены исследования зависимости времени работы трех алгоритмов от размеров перемножаемых матриц и количества использованных потоков. Замеры времени проводились для матриц четной размерности размером от 100 до 1000 с шагом 100 и матриц нечетной размерности размером от 101 до 1001 с шагом 100. Количество потоков - от 1 до 16, т.к. компьютер, на котором проводились вычисления, содержит 4 логических ядра.

Временные замеры проводятся путём многократного проведения эксперимента и деления результирующего времени на количество итераций эксперимента.

4.2 Анализ времени работы

На рис. 5 представлен график времени работы алгоритма на матрицах четной размерности:

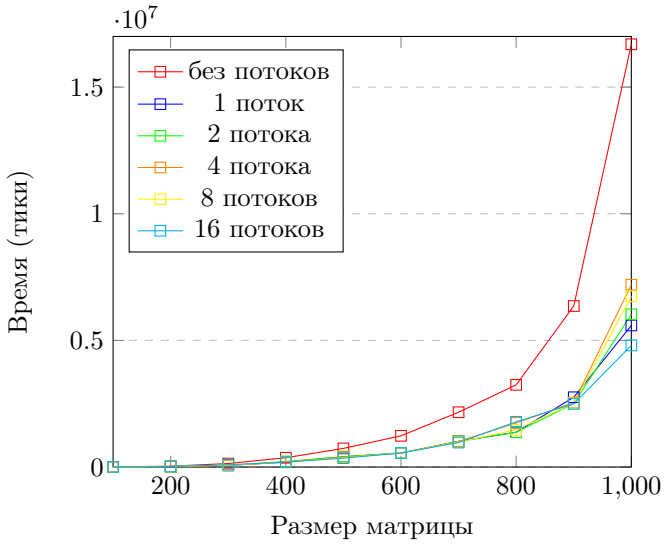

Рис. 5: График времени работы алгоритмов на матрицах четной размерности

На рис. 6 представлен график времени работы алгоритма на матрицах нечетной размерности:

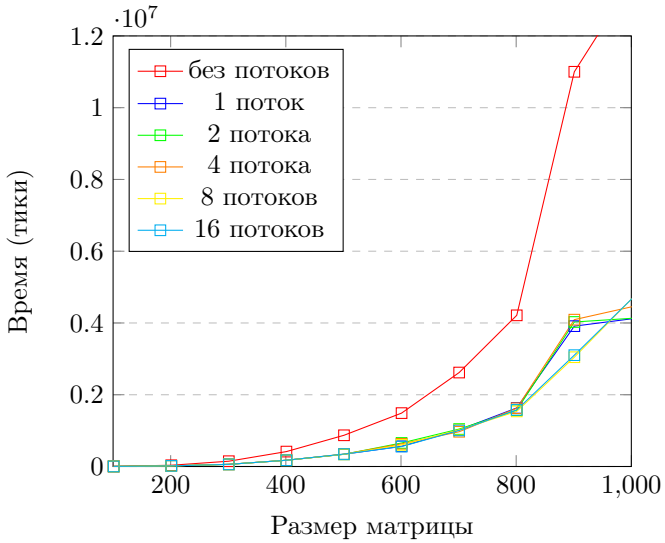

Рис. 6: График времени работы алгоритмов на матрицах четной размерности

4.3 Вывод

При увеличении количества потоков (до 2, 4, 8, 16), время работы программы существенно сокращается

Заключение

В ходе работы был изучен параллельный алгоритм умножения матриц: алгоритм Винограда. Выполнено сравнение зависимости всех рассматриваемых алгоритмов от числа параллельных потоков и размера матриц. В ходе исследования было установлено, что многопоточный алгоритм Винограда выполняется быстрее, чем стандартный алгоритм.

Список литературы

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.-М.:Техносфера, 2009.

[2] Henry Cohn, Robert Kleinberg, Balazs Szegedy, and Chris Umans. Group-theoretic Algorithms for Matrix Multiplication. arXiv:math.GR/0511460. Proceedings of the 46th Annual Symposium on Foundations of Computer Science, 23-25 October 2005, Pittsburgh, PA, IEEE Computer Society, pp. 379—388..

[3] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation, 9:251-280, 1990.