МГТУ им. Баумана

Лабораторная работа №4

По курсу: "Анализ алгоритмов"

Параллельное умножение матриц

Работу выполнила: Серёгина Дарья, ИУ7-54Б

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

Москва, 2020

Оглавление

Введение

Цель работы: изучение возможности параллельных вычислений и использование такого подхода на практике. Реализация парралельного 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.

1 Аналитическая часть

Матрицей A размера $[m * n]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы.[1] Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить. У произведения будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй.

Пусть даны две прямоугольные матрицы A и B размеров $[m * n]$ и $[n * k]$ соответственно. В результате произведение матриц A и B получим матрицу C размера $[m * k]$.

$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j}$ называется произведением матриц A и B [1].

1.1 Алгоритм Винограда

Подход Алгоритма Винограда является иллюстрацией общей методологии, начатой в 1979-х годах на основе билинейных и трилинейных форм, благодаря которым большинство усовершенствований для умножения матриц были получены [2].

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$.

Их скалярное произведение равно (1.1)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.1}$$

Равенство (1.1) можно переписать в виде(1.2)

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{1.2}$$

Менее очевидно, 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. Это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.1.1 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.

1.2 Параллельное программирование

При использовании многопроцессорных вычислительных систем с общей памятью обычно предполагается, что имеющиеся в составе системы процессоры обладают равной производительностью, являются равноправными при доступе к общей памяти, и время доступа к памяти является одинаковым (при одновременном доступе нескольких процессоров к одному и тому же элементу памяти очередность и синхронизация доступа обеспечивается на аппаратном уровне). Многопроцессорные системы подобного типа обычно именуются симметричными мультипроцессорами (symmetric multiprocessors, SMP).

Перечисленному выше набору предположений удовлетворяют также активно развиваемые в последнее время многоядерные процессоры, в которых каждое ядро представляет практически независимо функциони рующее вычислительное устройство.

Обычный подход при организации вычислений для многопроцессорных вычислительных систем с общей памятью – создание новых параллельных методов на основе обычных последовательных программ, в ко-

торых или автоматически компилятором, или непосредственно программистом выделяются участки независимых друг от друга вычислений. Возможности автоматического анализа программ для порождения параллельных вычислений достаточно ограничены, и второй подход является преобладающим. При этом для разработки параллельных программ могут применяться как новые алгоритмические языки, ориентированные на параллельное программирование, так и уже имеющиеся языки, расширенные некоторым набором операторов для параллельных вычислений.

Широко используемый подход состоит и в применении тех или иных библиотек, обеспечивающих определенный программный интерфейс (application programming interface, API) для разработки параллельных программ. В рамках такого подхода наиболее известны Windows Thread API. Однако первый способ применим только для ОС семейства Microsoft Windows, а второй вариант API является достаточно трудоемким для использования и имеет низкоуровневый характер [3].

1.2.1 Организация взаимодействия параллельных потоков

Потоки исполняются в общем адресном пространстве параллельной программы. Как результат, взаимодействие параллельных потоков можно организовать через использование общих данных, являющихся доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения. В случае же, когда общие данные могут изменяться несколькими потоками, необходимы специальные усилия для организации правильного взаимодействия.

1.3 Вывод

Был рассмотрен алгоритм Винограда и возможность его оптимизации с помощью распараллеливания потоков. Была рассмотрена технология параллельного программирования и организация взаимодействия параллельных потоков.

2 | Конструкторская часть

Требования к вводу: На вход подаются две матрицы
Требования к программе:

- корректное умножение двух матриц;
- при матрицах неправилыных размеров программа не должна аварийно завершаться.

2.1 Схемы алгоритмов

В данной части будет рассмотрена схема алгоритма Винограда (Рис. 2. 1) и выбранный спрособ распараллеливания.

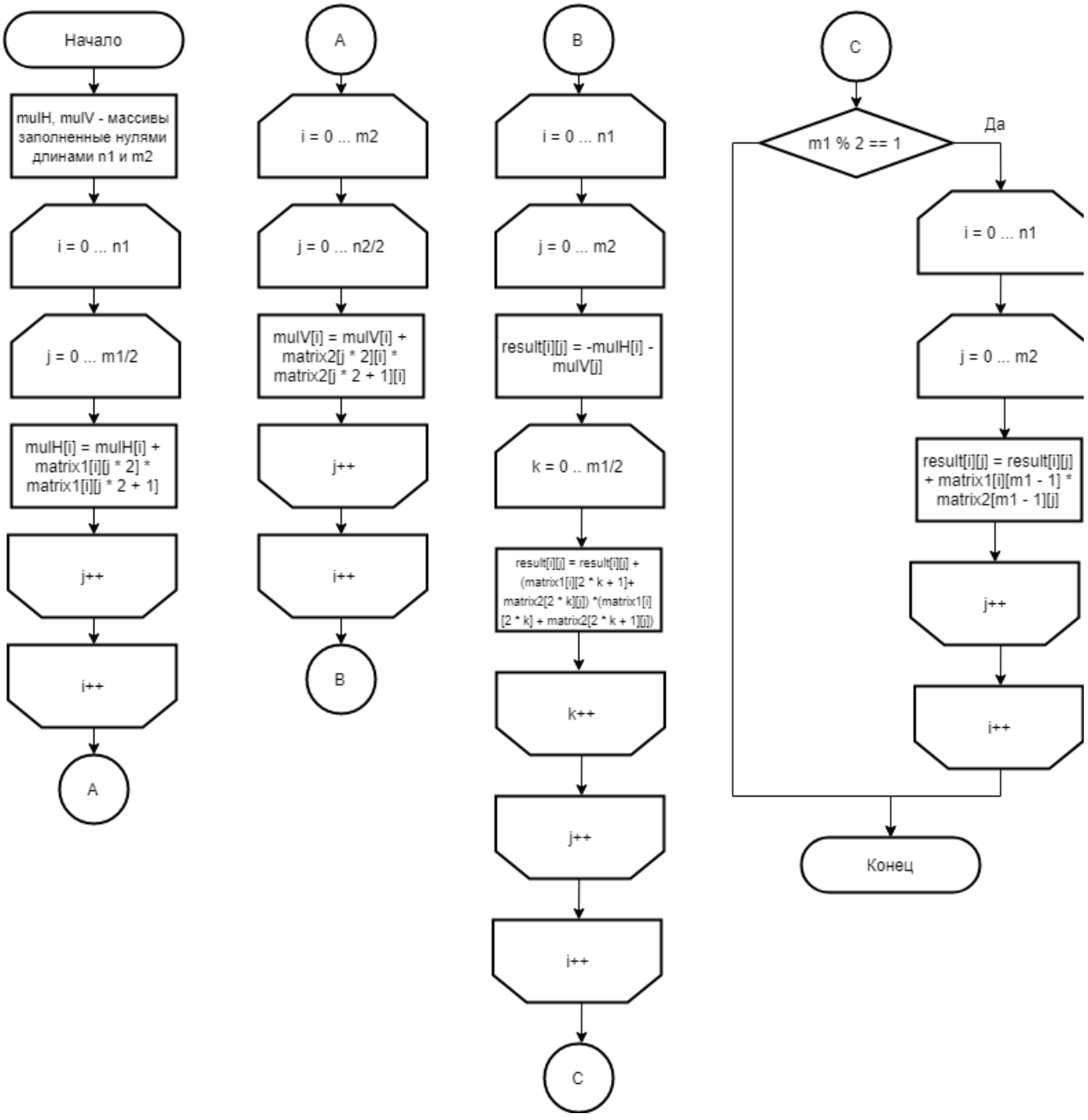

Рис. 2.1: Схема алгоритма Винограда

2.2 Распараллеливание программы

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

В предложенном алгоритме данным участком будет являться тройной цикл поиска результата. Данный блок программы как раз предлагается распараллелить. На (Рис. 2.1) это участок между B и C.

2.3 Вывод

В данном разделе была рассмотрена схема алгоритма Винограда и способ ее распараллеливания.

3 | Технологическая часть

Замеры времени были произведены на: Intel(R) Core(TM) i5-8300H, 4 ядра, 8 логических процессоров.

3.1 Выбор ЯП

В качестве языка программирования был выбран C# [4] так как этот язык поддерживает управление потоками на уровне ОС(незеленые потоки). Средой разработки Visual Studio. Время работы алгоритмов было замерено с помощью класса Stopwatch. Многопоточное программирование было реализовано с помощью пространства имен System.Threading.

3.2 Сведения о модулях программы

Программа состоит из:

- Program.cs - главный файл программы, в котором располагается точка входа в программу и функция замера времени;
- Mult.cs - файл класса Mult, в которм находится обычный алгоритм Винограда;
- ParallelMult.cs - файл класса ParallelMult, в котором находится параллельная реализация алгоритма Винограда.

3.3 Листинг кода алгоритмов

Листинг 3.1: Алгоритм Винограда

```
void Vinograd_n_thread(matrix_type &a, matrix_type &b,
    matrix_type &c, int n)
{
  vector<int> row(a.n);
  vector<int> column(b.m);

  vector<thread> threads;

  unsigned int n1 = a.n / 2;
  zeroing(c.matrix, c.n, c.m);

  double length_part = (double) a.n / n;
  if (length_part < 1) {
    length_part = 1;
  }

  for (int i = 0; i < a.n; i++) {
    row.push_back(0);
  }
  for (int i = 0; i < b.m; i++) {
    column.push_back(0);
  }

  thread thr_mulH(create_mulH, ref(a.matrix), ref(row), 0,
      ref(a.n), ref(a.m));
  thread thr_mulV(create_mulV, ref(b.matrix), ref(column),
      0, ref(b.m), ref(b.n));

  thr_mulH.join();
  thr_mulV.join();

  for (int i = 1; i <= n; i++) {
    threads.push_back(thread(calculate1, ref(a), ref(b),
        ref(c), ref(row), ref(column), (a.n * (i - 1)) / n,
        (a.n * i) / n));
  }

  for (int i = 0; i < threads.size(); ++i) {
    if (threads[i].joinable()) {
```

```
35        threads[i].join();
36      }
37    }
38  }
```

Листинг 3.2: Основные вычисления для многопоточной реализации алгоритма Винограда

```
1  void create_mulH(int **&A, vector <int>& row, const
       unsigned int &M_start, const unsigned int &M_end, const
       unsigned int &N)
2  {
3
4    for (unsigned i = M_start; i < M_end; i++) {
5      //cout << this_thread::get_id()<< endl;
6      for (unsigned k = 0; k < N / 2; k++) {
7        row[i] += A[i][2 * k] * A[i][2 * k + 1];
8      }
9    }
10 }
11
12 void create_mulV(int **&B, vector <int>& column, const
       unsigned int &Q_start, const unsigned int &Q_end, const
       unsigned int &N)
13 {
14
15   for (unsigned i = Q_start; i < Q_end; i++) {
16     //cout << this_thread::get_id()<< endl;
17     for (unsigned k = 0; k < N / 2; k++) {
18       column[i] += B[2 * k][i] * B[2 * k + 1][i];
19     }
20   }
21 }
22
23 void calculate(int **&A, int **&B, int **&C, vector <int> &
       row, vector <int> &column, const unsigned int &M, const
       unsigned int &N, const unsigned int &Q)
24 {
25   for (unsigned i = 0; i < M; i++)
26   for (unsigned j = 0; j < Q; j++) {
27     if (N % 2 == 0)
```

```
28      C[i][j] = -row[i] - column[j];
29      else
30      C[i][j] = -row[i] - column[j] + A[i][N - 1] * B[N - 1][
            j];
31
32      for (unsigned k = 0; k < N / 2; k++) {
33        C[i][j] = C[i][j] + (A[i][k << 1] + B[k << 1 | 1][j])
              *
34        (A[i][k << 1 | 1] + B[k << 1][j]);
35      }
36    }
37
38  }
39
40  void calculate1(matrix_type &a, matrix_type &b, matrix_type
        &c, vector <int> &row, vector <int> &column, const
      unsigned int n_start, unsigned int n_end)
41  {
42    int sum = 0;
43
44    for (unsigned i = n_start; i < n_end; i++) {
45      //cout << this_thread::get_id()<< endl;
46      for (unsigned j = 0; j < b.m; j++) {
47
48        sum = -row[i] - column[j];
49
50        for (unsigned k = 0; k < a.m / 2; k++) {
51          sum += (a.matrix[i][2*k] + b.matrix[2*k+1][j]) *
52          (a.matrix[i][2*k+1] + b.matrix[2*k][j]);
53        }
54
55        if (a.m % 2 == 1)
56        sum += a.matrix[i][a.m - 1] * b.matrix[b.n - 1][j];
57
58        c.matrix[i][j] = sum;
59      }
60    }
61
62  }
```

3.4 Вывод

В данном разделе были рассмотрены основные сведения о модулях программы, листинг кода.

4 | Исследовательская часть

4.1 Сравнительный анализ на основе замеров времени

Был проведен замер времени работы функций.

Так как в C++ функции, в которой работает поток, можно передать одну переменную типа object я задумалась о способе адрессации данных.

В первом эксперименте я хотела эмпирически узнать на сколько сильно разная адрессация повлияет на время работы:

В первом случае производилось получение полей класса переданной структуры каждый раз. Во втором случае данные изначально записывались в переменные.

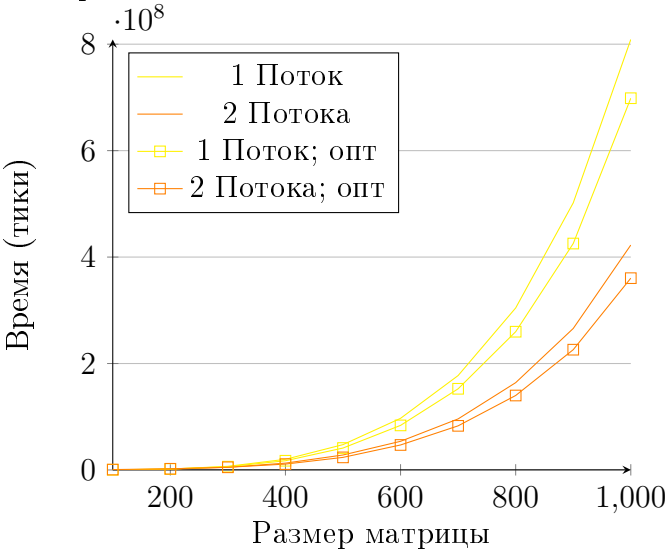

Рис. 4.1: Сравнение времени работы при разной адрессации

переменных на 1 и 2 потоках

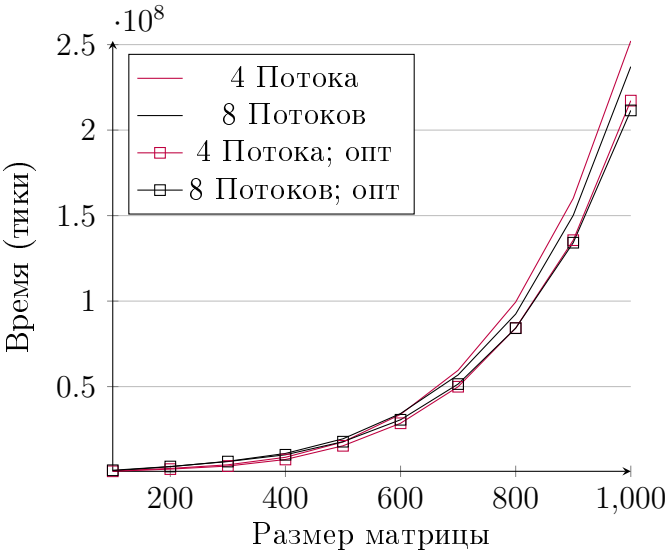

Рис. 4.2: Сравнение времени работы при разной адрессации переменных на 4 и 8 потоках

На графиках видно, что оптимизированная реализация работает на 10% быстрее неоптимизированой, целесообразнее использовать предварительное присвоение полей класса в переменные.

Во втором эксперименте я хотела узнать на сколько целесообразно распараллеливать вычисления mulV и mulH.

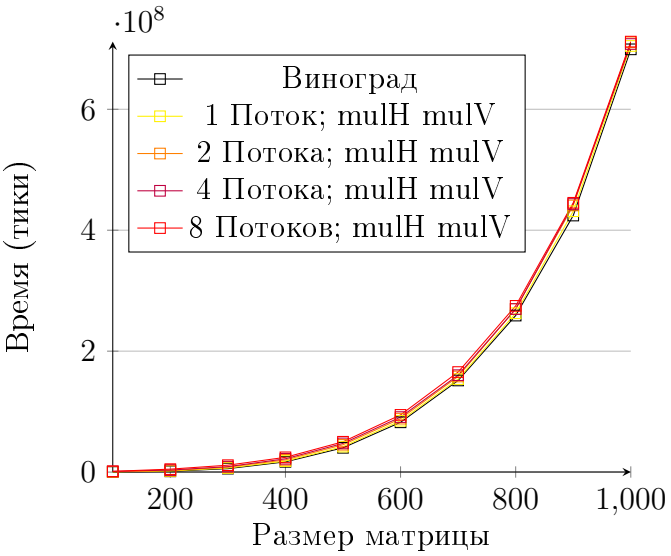

Рис. 4.3: Сравнение времени работы обычного Винограда и распараллеленного mulH mulV

На графиках видно, что распараллеливание mulH и mulV не имеет смысла в рамках оптимизации, так как разница всех результатов не превышает 1%. Поэтому дальнейшее сравнение будет производиться для распараллеленного главного цикла с оптимизированной адрессацией.

Третий эксперимент производится для лучшего случая на квадратных матрицах размером от 100 x 100 до 1000 x 1000 с шагом 100. Сравним результаты для обычного Винограда и Винограда с распараллеленным главным циклом:

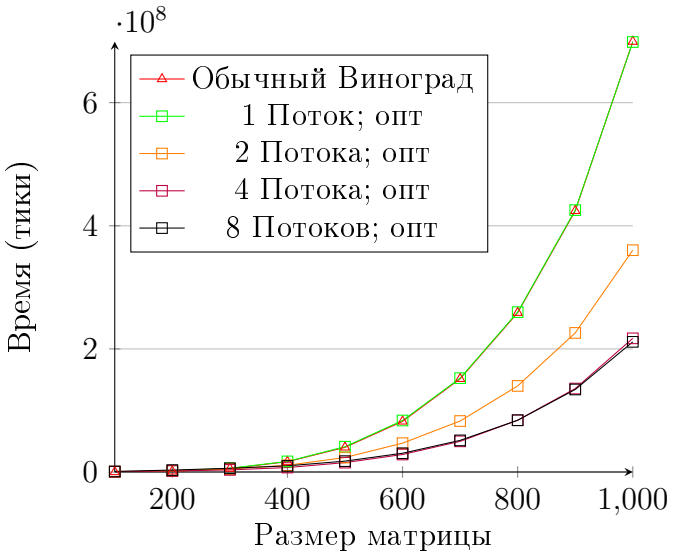

Рис. 4.4: Сравнение времени работы для обычного Винограда и Винограда с распараллеленным главным циклом

4.2 Вывод

По результатам исследования обычный алгоритм Винограда приблизительно равен однопоточному его эквиваленту (разница менее 0.5%), а многопоточная реализация алгоритма Винограда работает быстрее при увеличении количества потоков (2 потока работает в 2 раза быстрее одного, 4 потока работает на 30% быстрее двух). Это увеличение становится незначительным после 4х потоков (менее 1% разницы).

Заключение

В ходе лабораторной работы я изучила возможности параллельных вычислений и использовала такой подход на практике. Реализовала алгоритм Винограда умножения матриц с помощью параллельных вычислений.

Был проведен эксперимент для поиска оптимального способа адресации в результате которого оказалось, что эффективнее производить обращение к переменным, предварительно присвоив им поля класса.

Также был проведен анализ целесообразности распараллеливания циклов поиска mulH mulV, в результате которого оказалось, что при таком подходе разница результатов при увеличении потоков не превышает 1%.

Ввиду результатов предыдущих исследовиний следующий эксперимент проводился с использованием оптимизированой адрессации, распараллеливание было произведено только для главного цикла поиска результата. Было произведено сравнение работы обычного алгоритма Винограда и параллельной реализации при увеличении количества потоков. Выяснилось, что увеличение потоков до 4х сокращает время работы на 70% по сравнению с однопоточной реализацией. Однако дальнейшее увеличение количества потоков не дает значительного выигрыша во времени (разница менее 1%).

Литература

[1] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16

[2] Le Gall, F. (2012), "Faster algorithms for rectangular matrix multiplication Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), pp. 514–523

[3] Константин Баркалов, Владимир Воеводин, Виктор Гергель. Intel Parallel Programming [Электронный ресурс], - режим доступа https://www.intuit.ru/studies/courses/4447/983/lecture/14925

[4] Руководство по языку C#[Электронный ресурс], - режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/