

МГТУ им. БАУМАНА

ЛАБОРАТОРНАЯ РАБОТА №4

По курсу: "АНАЛИЗ АЛГОРИТМОВ"

Параллельное умножение матриц

Работу выполнил: Мокеев Даниил, ИУ7-54

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

Москва, 2019

Оглавление

Введение	3
1 Аналитическая часть	4
1.1 Алгоритм Винограда	4
1.1.1 Параллельный алгоритм Винограда	5
1.2 Параллельное программирование	5
1.2.1 Организация взаимодействия параллельных потоков	6
1.3 Вывод	6
2 Конструкторская часть	7
2.1 Схемы алгоритмов	7
2.2 Распараллеливание программы	7
2.3 Вывод	7
3 Технологическая часть	9
3.1 Выбор ЯП	9
3.2 Описание структуры ПО	9
3.3 Сведения о модулях программы	10
3.4 Листинг кода алгоритмов	10
3.5 Вывод	12
4 Исследовательская часть	13
4.1 Примеры работы	13
4.2 Постановка эксперимента	13
4.2.1 Вывод экспериментальной части	14
Заключение	15

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.

В ходе лабораторной работы предстоит:

- изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда;
- оптимизировать алгоритм Винограда;
- дать теоретическую оценку базового алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда;
- реализовать три алгоритма умножения матриц на одном из языков программирования;
- сравнить алгоритмы умножения матриц.

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 произведения 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 $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$.

Их скалярное произведение равно (1.1)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \quad (1.1)$$

Равенство (1.1) можно переписать в виде (1.2)

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

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 функционирующее 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 неправильных размеров программа не должна аварийно завершаться.

2.1 Схемы алгоритмов

В данной части будут рассмотрена схема алгоритма Винограда.

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

2.3 Вывод

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

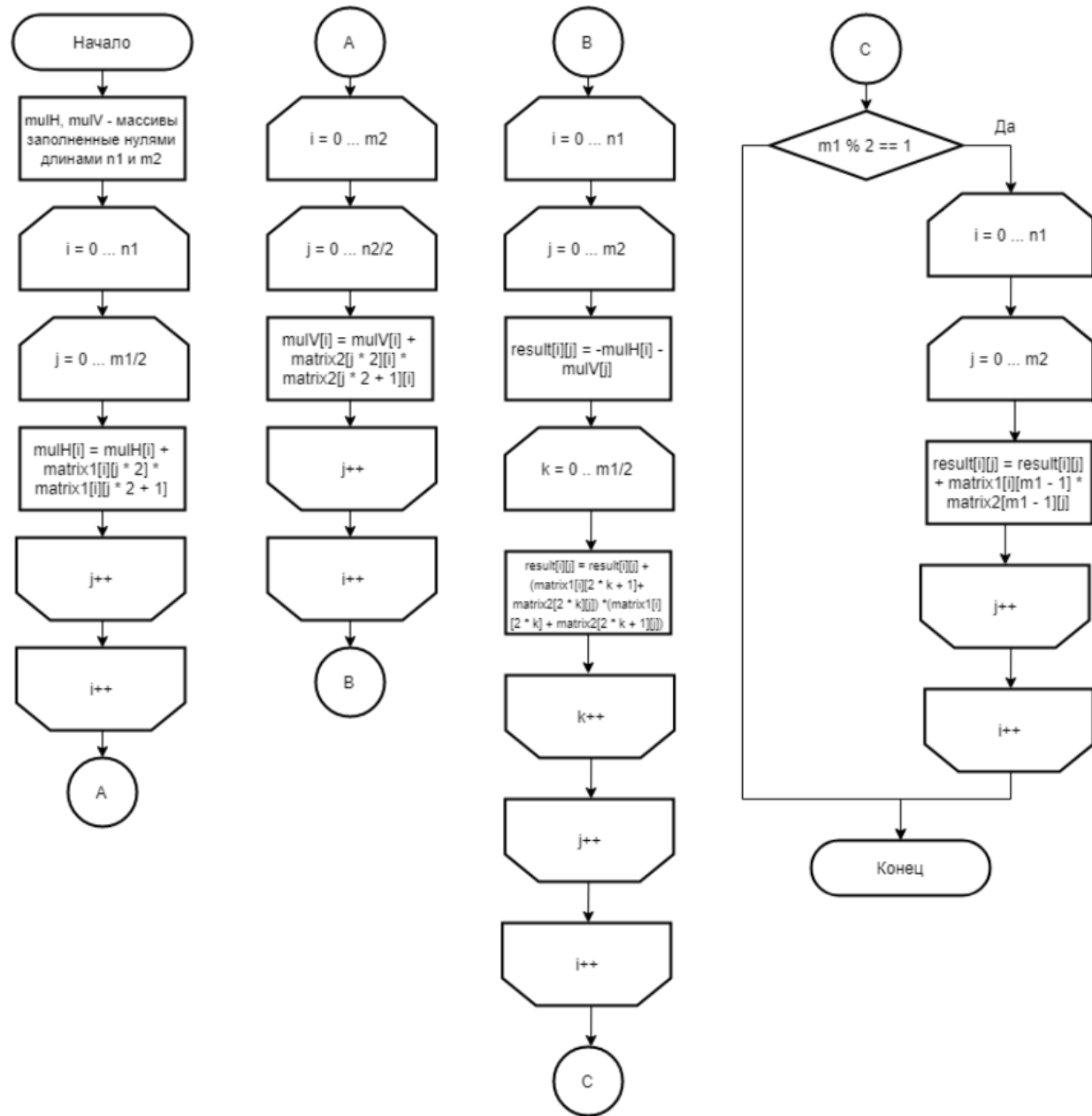


Рис. 2.1: Схема алгоритма Винограда

3 | Технологическая часть

3.1 Выбор ЯП

Я выбрал в качестве языка программирования Golang, потому как он достаточно удобен, быстр и успешно использует концепции мультипоточного программирования.

Время работы алгоритмов было замерено с помощью функции `Now()` из библиотеки `time`.

3.2 Описание структуры ПО

В данном разделе будет представлена функциональная схема программы (Рис. 3.1.)



Рис. 3.1: Функциональная схема умножения матриц (IDEF0 диаграмма 1 уровня)

3.3 Сведения о модулях программы

Программа состоит из:

- lab04.go- главный файл программы, в котором располагается точка входа в программу.
- matr.go - файл содержащий функции предобработки матриц и измерение времени.

3.4 Листинг кода алгоритмов

В данном разделе будет представлен листинги кода алгоритма Винограда (3.1), распараллеленного алгоритма Винограда (3.2)

Листинг 3.1: Алгоритм Винограда

```
1 func winograd(mtr1, mtr2 *mtr) *mtr{
2     if mtr2.rows != mtr1.cols{
3         panic("Wrong size of matrix")
4     }
5     row1 := mtr1.rows; col1 := mtr1.cols
6     row2:= mtr2.rows; col2 := mtr2.cols
7
8     row_factor := make([]int, row1, row1)
9     col_factor := make([]int, col2, col2)
10
11     for i:=0; i<row1;i++){
12         for j:=0;j<col1 / 2;j++){
13             row_factor[i] += mtr1.buff[i][2*j] * mtr1.buff[i][2*j
14             +1]
15         }}
16
17     for i:=0; i<col2;i++){
18         for j:=0;j<row2 / 2;j++){
19             col_factor[i] += mtr2.buff[2*j][i]*mtr2.buff[2*j+1][i
20             ]
21         }}
22
23     answer := new_mtr(row1, col2)
```

```

22
23   for i:=0;i<row1;i++{
24       for j:=0;j<col2;j++{
25           answer.buff[i][j]+= - row_factor[i] - col_factor[j]
26           for k:=0;k<col1 / 2;k++{
27               answer.buff[i][j] += ((mtr1.buff[i][2 * k] + mtr2.
                buff[2 * k + 1][j]) * (mtr1.buff[i][2 * k + 1] +
                mtr2.buff[2 * k][j]))
28       }}}
29
30   if (row2 % 2) != 0{
31       for i:=0;i<row1;i++{
32           for j:=0;j<col2;j++{
33               answer.buff[i][j] += mtr1.buff[i][col1 - 1] * mtr2.
                buff[col1 - 1][j]
34       }}}
35   return answer
36 }

```

Листинг 3.2: Распараллеленый Алгоритм Винограда

```

1 func winograd_parallel(mtr1, mtr2 *mtr, threads int) *mtr{
2     if mtr2.rows != mtr1.cols{
3         panic("Wrong size of matrix")
4     }
5     row1 := mtr1.rows; col1 := mtr1.cols
6     row2:= mtr2.rows; col2 := mtr2.cols
7
8     row_factor := make([]int, row1, row1)
9     col_factor := make([]int, col2, col2)
10
11     for i:=0; i<row1;i++{
12         for j:=0;j<col1 / 2;j++{
13             row_factor[i] += mtr1.buff[i][2*j] * mtr1.buff[i][2*j
                +1]
14         }}
15     for i:=0; i<col2;i++{
16         for j:=0;j<row2 / 2;j++{
17             col_factor[i] += mtr2.buff[2*j][i]*mtr2.buff[2*j+1][i
                ]
18         }}

```

```

19  answer := new_mtr(row1, col2)
20  in := make(chan int); quit := make(chan bool)
21  mult := func(){
22      for{
23          select {
24              case i := <- in:
25                  for j:=0;j<col2;j++){
26                      answer.buff[i][j]+= - row_factor[i] - col_factor[
27                          j]
28                      for k:=0;k<col1 / 2;k++){
29                          answer.buff[i][j] += ((mtr1.buff[i][2 * k] + mtr2
30                              .buff[2 * k + 1][j]) * (mtr1.buff[i][2 * k +
31                                  1] + mtr2.buff[2 * k][j]))
32                      }}
33              case <- quit:
34                  return
35          }
36      }
37  }
38  for i:=0;i<threads;i++){
39      go mult()
40  }
41  for i:=0;i<mtr1.rows;i++){
42      in <- i
43  }
44  for i:= 0 ; i<threads; i++){
45      quit<-true
46  }
47  return answer
48 }

```

3.5 Вывод

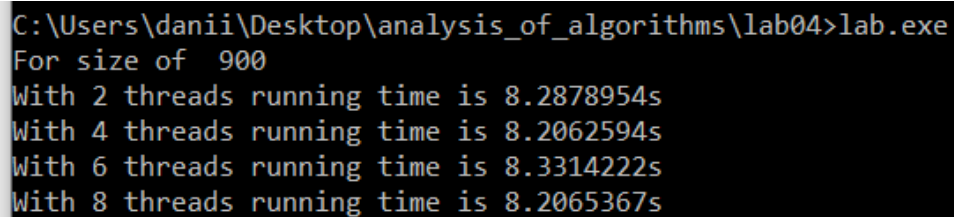
В данном разделе была рассмотрена структура ПО и листинги кода программы.

4 | Исследовательская часть

Был проведен замер времени работы алгоритмов с использованием разного количества потоков. Исследования были проведены на процессоре Intel Core i5-6200U

4.1 Примеры работы

В данном разделе приведен пример работы программы (Рис. 4.1)



```
C:\Users\danii\Desktop\analysis_of_algorithms\lab04>lab.exe
For size of 900
With 2 threads running time is 8.2878954s
With 4 threads running time is 8.2062594s
With 6 threads running time is 8.3314222s
With 8 threads running time is 8.2065367s
```

Рис. 4.1: Пример работы программы

4.2 Постановка эксперимента

Проведем сравнение для каждого из алгоритмов. Для замера времени будем использовать функцию `time.Now()`

Сравним результаты для обычного Винограда и Винограда с параллельным главным циклом:

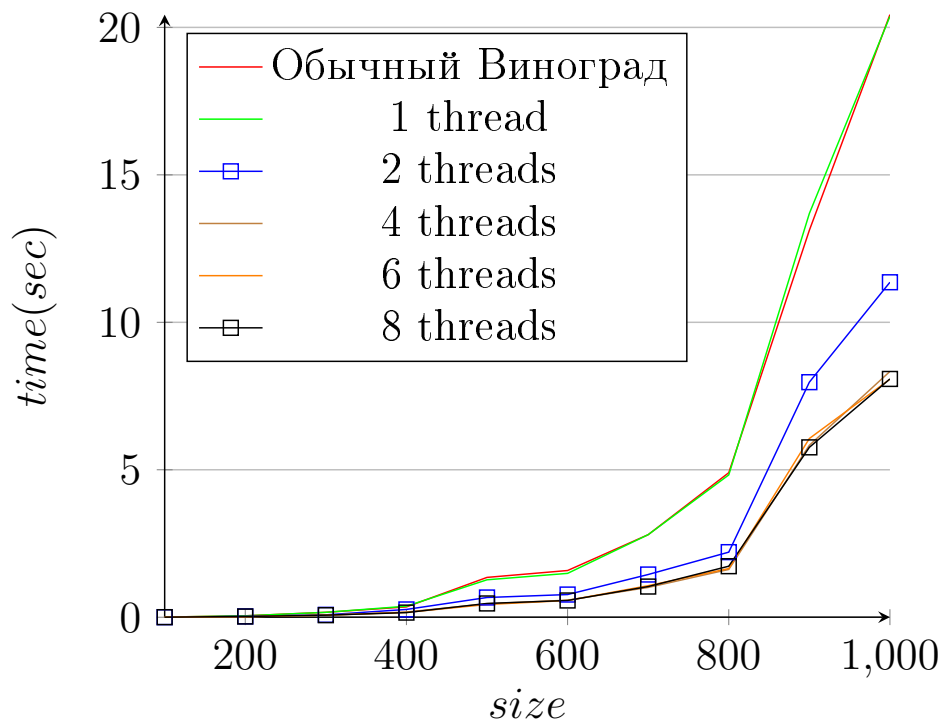


Рис. 4.2: Сравнение использования разного количества потоков

4.2.1 Вывод экспериментальной части

Эксперимент показывает, что использование одного потока эквивалентно обычной версии алгоритма. Использование двух потоков значительно ускоряет работу алгоритма. Данный эффект наблюдается и при увеличении числа потоков, однако при использовании более четырех потоков дальнейшего ускорения не происходит.

Заключение

В ходе лабораторной работы были изучены возможности параллельных вычислений, 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 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/>