Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 1

по курсу «Разработка параллельных и распределенных программ»

Распараллеливание алгоритма вычисления произведения двух матриц.

Студент: Пишикина М.В.

Группа: ИУ9-51Б

Преподаватель: Царев А.С.

Москва 2024

Содержание

1 Постановка задачи

Две квадратные матрицы A и B размерности n сначала перемножить стандартным алгоритмом. Для получения матрицы C той же размености. Замерить время вычисления, сравнить с временем при вычислении элементов матрицы C не по строкам, а по столбцам. Размер матриц подобрать таким образом, чтобы время выполнения на вашей машине было не слишком непоказательно малым, но и не чересчур большим. Использовать библиотечные функции для вычисления произведений матриц нельзя. Затем конечную матрицу C условно разделить на примерно равные прямоугольные подматрицы и распараллелить программу таким образом, чтобы каждый поток занимался вычислением своей подматрицы. Матрицы A и B для этого разделить на примерно равные группы строк и столбцов соответственно. Сделать для разного количества потоков (разных разбиений), также замерить время вычисления, сравнить с вычислениями стандартным алгоритмом. Также по окончании вычислений сравнивать получившуюся матрицу с той, что была вычислена стандартным алгоритмом, для проверки правильности вычислений.

2 Практическая реализация

```
package main

import (
	"fmt"
	"math/rand"
	"sync"
	"time"
)

const n = 900

func generateMatrix(n int) [][]float64 {
	matrix := make([][]float64, n)
```

```
        for i := 0; i < n; i++ {
                matrix[i] = make([]float64, n)
                for j := 0; j < n; j++ {
                        matrix[i][j] = rand.Float64() * 100
                }
        }
        return matrix
}

// Умножение по строкам
func multiplyMatricesRow(A, B [][]float64) [][]float64 {
        C := make([][]float64, n)
        for i := 0; i < n; i++ {
                C[i] = make([]float64, n)
                for j := 0; j < n; j++ {
                        sum := 0.0
                        for k := 0; k < n; k++ {
                                sum += A[i][k] * B[k][j]
                        }
                        C[i][j] = sum
                }
        }
        return C
}

// Умножение по столбцам
func multiplyMatricesCol(A, B [][]float64) [][]float64 {
        C := make([][]float64, n)
        for i := 0; i < n; i++ {
                C[i] = make([]float64, n)
                for j := 0; j < n; j++ {
                        C[i][j] = 0
                }
```

```
        }
        for j := 0; j < n; j++ {
                for k := 0; k < n; k++ {
                        for i := 0; i < n; i++ {
                                C[i][j] += A[i][k] * B[k][j]
                        }
                }
        }
        return C
}

// Функция для сравнения матриц
func compareMatrices(A, B [][]float64) bool {
        for i := 0; i < n; i++ {
                for j := 0; j < n; j++ {
                        if A[i][j] != B[i][j] {
                                return false
                        }
                }
        }
        return true
}

// Распараллеливание
func multiplyMatricesParallel(A, B [][]float64, workers int) [][]float64 {
        C := make([][]float64, n)
        for i := 0; i < n; i++ {
                C[i] = make([]float64, n)
        }

        var wg sync.WaitGroup          // sync.WaitGroup(*) для синхронизации
        ↪  потоков
        wg.Add(workers)
```

```
        rowsPerWorker := n / workers

        for worker := 0; worker < workers; worker++ {
                go func(worker int) {
                        defer wg.Done()                 // при завершении горутины
                         ↪  уменьшается счетчик ожидания в (*)
                        startRow := worker * rowsPerWorker
                        endRow := startRow + rowsPerWorker
                        if worker == workers-1 {
                                endRow = n              // последний worker берет
                                 ↪  оставшиеся строки
                        }
                        for i := startRow; i < endRow; i++ {
                                for j := 0; j < n; j++ {
                                        sum := 0.0
                                        for k := 0; k < n; k++ {
                                                sum += A[i][k] * B[k][j]
                                        }
                                        C[i][j] = sum
                                }
                        }
                }(worker)
        }

        wg.Wait()       // ожидание завершения всех горутин
        return C
}

func main() {
        A := generateMatrix(n)
        B := generateMatrix(n)

        start := time.Now()
```

```
        C_1 := multiplyMatricesRow(A, B)
        fmt.Println("Время умножения по строкам:",
         ↪  time.Since(start).Seconds())

        start = time.Now()
        C_2 := multiplyMatricesCol(A, B)
        fmt.Println("Время умножения по столбцам:",
         ↪  time.Since(start).Seconds())

        if compareMatrices(C_1, C_2) {
                fmt.Println("Матрицы C_1 и C_2 совпадают")
        } else {
                fmt.Println("Матрицы C_1 и C_2 НЕ совпадают")
        }

        for _, workers := range []int{2, 4, 8, 16, 32} {
                start = time.Now()
                C_3 := multiplyMatricesParallel(A, B, workers)
                fmt.Printf("%d потока: %v секунд\n", workers,
                 ↪  time.Since(start).Seconds())

                // Проверка правильности
                if compareMatrices(C_1, C_3) {
                        fmt.Printf("Матрицы C_1 и C_3 (%d потока)
                         ↪  совпадают\n", workers)
                } else {
                        fmt.Printf("Матрицы C_1 и C_3 (%d потока) НЕ
                         ↪  совпадают\n", workers)
                }
        }
}
```

3 Характеристика устройства

Устройство: MacBook Pro 2020

Операционная система: macOS Sonoma

Процессор: Intel Core i5

Характеристика процессора: 4-ядерный чип, частота 2 ГГц

Оперативная память: 16GB LPDDR4X

4 Время работы программы

Время умножения по строкам: 3.157595062

Время умножения по столбцам: 6.254248264

Матрицы C1 и C2 совпадают

2 потока: 1.597707362 секунд

4 потока: 0.793722605 секунд

8 потока: 0.732268748 секунд

16 потока: 0.763583577 секунд

32 потока: 0.813922683 секунд

Матрицы C1 и C3 совпадают во всех случаях

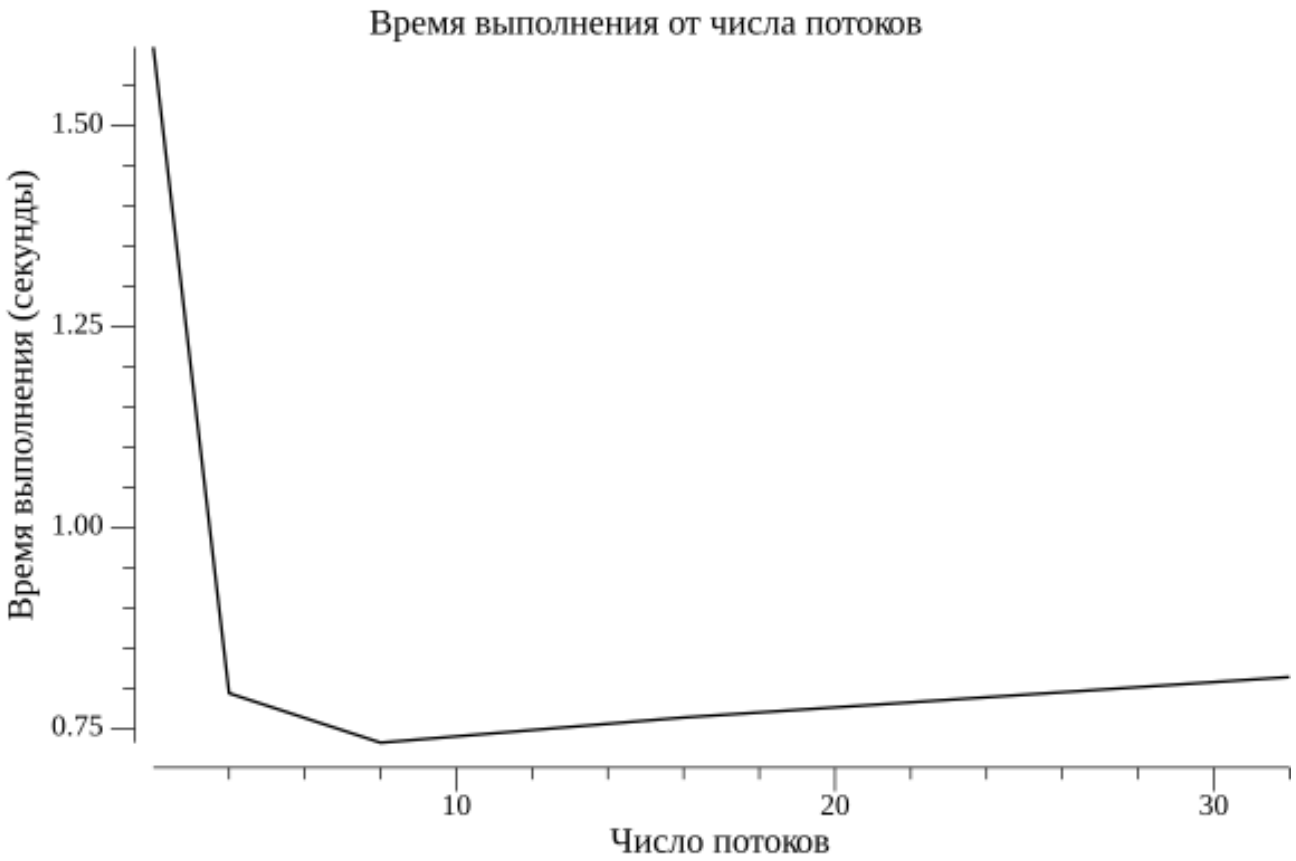
Время выполнения от числа потоков
Время выполнения (секунды)
1.50
1.25
1.00
0.75
10
20
30
Число потоков

Рис. 1 — График