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)»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №1

Наспараллеливание алгоритма вычисления произведения двух матриц.

Выполнил
студент группы ИУ9-51Б
Лисов Алексей

Москва, 2024

1 Условие

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 (раз- ных разбиений), также замерить время вычисления, сравнить с вычислениями стандартным алгоритмом. Также по окончании вычислений сравнивать полу- чившуюся матрицу с той, что была вычислена стандартным алгоритмом, для проверки правильности вычислений.

2 Код решения

Имя Go файла: main.go Содержимое Go файла:

```
package main

import (
        "log"
        "runtime"
        "time"

        "github.com/BaldiSlayer/rprp/lab1/internal/experiment"
        "gonum.org/v1/plot"
        "gonum.org/v1/plot/plotter"
        "gonum.org/v1/plot/plotutil"
        "gonum.org/v1/plot/vg"
)

const (
        maxValues          = 200
        some          int = 1650
        maxGoroutines      = 7
)
```

```
func durationToFloat64(s time.Duration) float64 {
        return s.Seconds()
}

func typeFind(i int) string {
        if i == 0 {
                return "Normal"
        }

        if i == 1 {
                return "Cols"
        }

        if i == 2 {
                return "Rows"
        }

        if i == 3 {
                return "Blocks"
        }

        return ""
}

func main() {
        runtime.GOMAXPROCS(2)

        log.Print(runtime.GOMAXPROCS(0))

        exp := experiment.New(2*some, 1*some, maxValues, 1*some, 2*some, maxValues)
        log.Print("Начало эксперимента")

        lines := make([]plotter.XYs, 4)
        for g := 1; g < maxGoroutines; g++ {
                log.Printf("Количество горутин: %d", g)

                results, err := exp.Run(g)
                if err != nil {
                        log.Fatalf("error while running experiment: %v", err)
```

```
                }

                lines[0] = append(lines[0], plotter.XY{X: float64(g), Y: durationTo
                lines[1] = append(lines[1], plotter.XY{X: float64(g), Y: durationTo
                lines[2] = append(lines[2], plotter.XY{X: float64(g), Y: durationTo
                lines[3] = append(lines[3], plotter.XY{X: float64(g), Y: durationTo
        }

        p := plot.New()
        p.Title.Text = "Результаты"
        p.X.Label.Text = "Количество потоков"
        p.Y.Label.Text = "Время в миллисекундах"

        for i, points := range lines {
                line, err := plotter.NewLine(points)
                if err != nil {
                        log.Fatalf("%v", err)
                }

                colorIndex := i % len(plotutil.SoftColors)
                line.Color = plotutil.SoftColors[colorIndex]

                p.Legend.Add(typeFind(i), line)
                p.Add(line)
        }

        p.Legend.Top = true

        // Сохраняем график в файл.
        if err := p.Save(4*vg.Inch, 4*vg.Inch, "plot.png"); err != nil {
                log.Fatalf("ошибка при сохранении в файл: %v", err)
        }
}
```

Имя Go файла: experiment.go
Содержимое Go файла:

```
package experiment

import (
        "errors"
        "github.com/BaldiSlayer/rprp/lab1/internal/matrix"
```

```
        "log"
        "time"
)

type Experiment struct {
        rowsA     int
        colsA     int
        maxValueA int

        rowsB     int
        colsB     int
        maxValueB int

        a matrix.Matrix
        b matrix.Matrix
}

func New(rowsA, colsA, maxValueA, rowsB, colsB, maxValueB int) *Experiment {
        return &Experiment{
                rowsA:     rowsA,
                colsA:     colsA,
                maxValueA: maxValueA,
                rowsB:     rowsB,
                colsB:     colsB,
                maxValueB: maxValueB,
                a:         matrix.NewRandomMatrix(rowsA, colsA, maxValueA),
                b:         matrix.NewRandomMatrix(rowsB, colsB, maxValueB),
        }
}

type result struct {
        m       matrix.Matrix
        elapsed time.Duration
}

// measureTime замеряет время выполнения функции
func measureTime(
        f func(a, b *matrix.Matrix, maxGorutines, w, h int) (matrix.Matrix, error),
        a, b *matrix.Matrix,
        maxGorutines, w, h int,
```

```
) (result, error) {
        start := time.Now()
        res, err := f(a, b, maxGorutines, w, h)
        elapsed := time.Since(start)

        return result{
                m:       res,
                elapsed: elapsed,
        }, err
}

type Measures struct {
        Normal time.Duration
        Cols   time.Duration
        Rows   time.Duration
        Blocks time.Duration
}

func (e *Experiment) Run(maxGorutines int) (Measures, error) {
        normal, err := measureTime(WrapMultiplyMatrices, &e.a, &e.b, 0, 0, 0)
        if err != nil {
                return Measures{}, err
        }

        log.Print("Закончен подсчет обычным способом")

        cols, err := measureTime(WrapMultiplyMatricesByCols, &e.a, &e.b, maxGoruti
        if err != nil {
                return Measures{}, err
        }

        log.Print("Закончен подсчет по строкам")

        rows, err := measureTime(WrapMultiplyMatricesByRows, &e.a, &e.b, maxGoruti
        if err != nil {
                return Measures{}, err
        }

        log.Print("Закончен подсчет по столбцам")
```

```
        // сделать автовыбор w и h
        blocks, err := measureTime(WrapMultiplyMatricesWithBlocks, &e.a, &e.b, max(
        if err != nil {
                return Measures{}, err
        }

        // вынести в отдельную функцию
        equal := normal.m.String() == cols.m.String() && cols.m.String() == rows.m
        if !equal {
                return Measures{}, errors.New("matrices are not equal")
        }

        return Measures{
                Normal: normal.elapsed,
                Cols:   cols.elapsed,
                Rows:   rows.elapsed,
                Blocks: blocks.elapsed,
        }, nil
}

func WrapMultiplyMatrices(a, b *matrix.Matrix, _, _, _ int) (matrix.Matrix, error)
        defer log.Print("Закончен обычный подсчет")

        log.Print("Начат обычный подсчет")

        return matrix.MultiplyMatrices(a, b)
}

func WrapMultiplyMatricesByCols(a, b *matrix.Matrix, maxGorutines, _, _ int) (matri
        defer log.Print("Закончен подсчет по строкам")

        log.Print("Начат подсчет по строкам")

        return matrix.MultiplyMatricesByCols(a, b, maxGorutines)
}

func WrapMultiplyMatricesByRows(a, b *matrix.Matrix, maxGorutines, _, _ int) (matri
        defer log.Print("Закончен подсчет по столбцам")

        log.Print("Начат подсчет по столбцам")
```

```
        return matrix.MultiplyMatricesByRows(a, b, maxGorutines)
}

func WrapMultiplyMatricesWithBlocks(a, b *matrix.Matrix, maxGorutines, w, h int) (r
        defer log.Print("Закончен подсчет по блокам")

        log.Print("Начат подсчет по блокам")

        return matrix.MultiplyMatricesWithBlocks(a, b, maxGorutines, w, h)
}
```

Имя Go файла: matrix.go Содержимое Go файла:

```
package matrix

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

type Matrix struct {
        data [][]int
        rows int
        cols int
}

var ErrCantMultiply = fmt.Errorf("failed to multiply matrix: the dimensions do not
var ErrBlocks = fmt.Errorf("failed to multiply matrix: blocks can't divide matrix

// NewMatrix создает новую пустую матрицу заданных размеров
func NewMatrix(rows, cols int) Matrix {
        data := make([][]int, rows)
        for i := range data {
                data[i] = make([]int, cols)
        }
        return Matrix{data: data, rows: rows, cols: cols}
}
```

```
// NewRandomMatrix создает новую матрицу заполненную случайными числами
// вы можете добавить параметры для определения размера матрицы и диапазона случай
func NewRandomMatrix(rows, cols, maxValue int) Matrix {
        source := rand.NewSource(time.Now().UnixNano())
        rng := rand.New(source)

        m := Matrix{
                data: make([][]int, rows),
                rows: rows,
                cols: cols,
        }

        for i := range m.data {
                m.data[i] = make([]int, cols)
                for j := range m.data[i] {
                        m.data[i][j] = rng.Intn(maxValue) // случайные числа от 0 
                }
        }

        return m
}

// String реализует способ печати матрицы
func (m Matrix) String() string {
        var result string
        for _, row := range m.data {
                result += fmt.Sprintln(row)
        }
        return result
}

// MultiplyMatrices перемножает две матрицы
func MultiplyMatrices(a, b *Matrix) (Matrix, error) {
        if a.cols != b.rows {
                return Matrix{}, ErrCantMultiply
        }

        result := NewMatrix(a.rows, b.cols)
```

```
        for i := 0; i < result.rows; i++ {
                for j := 0; j < result.cols; j++ {
                        sum := 0
                        for k := 0; k < a.cols; k++ {
                                sum += a.data[i][k] * b.data[k][j]
                        }
                        result.data[i][j] = sum
                }
        }

        return result, nil
}

func multiplyWorker(m1, m2, result *Matrix, row, col int) {
        sum := 0
        for k := 0; k < m1.cols; k++ {
                sum += m1.data[row][k] * m2.data[k][col]
        }

        result.data[row][col] = sum
}

func MultiplyMatricesByRows(m1, m2 *Matrix, maxGoroutines int) (Matrix, error) {
        if m1.cols != m2.rows {
                return Matrix{}, ErrCantMultiply
        }

        semaphore := make(chan struct{}, maxGoroutines)

        result := NewMatrix(m1.rows, m2.cols)

        var wg sync.WaitGroup
        wg.Add(result.rows)

        for i := 0; i < result.rows; i++ {
                semaphore <- struct{}{}

                go func(i int) {
                        defer wg.Done()
                        defer func() { <-semaphore }()
```

```
                        for j := 0; j < result.cols; j++ {
                                multiplyWorker(m1, m2, &result, i, j)
                        }
                }(i)
        }

        wg.Wait()

        return result, nil
}

func MultiplyMatricesByCols(m1, m2 *Matrix, maxGoroutines int) (Matrix, error) {
        if m1.cols != m2.rows {
                return Matrix{}, ErrCantMultiply
        }

        semaphore := make(chan struct{}, maxGoroutines)

        result := NewMatrix(m1.rows, m2.cols)

        var wg sync.WaitGroup
        wg.Add(result.rows)

        for i := 0; i < result.cols; i++ {
                semaphore <- struct{}{}

                go func(i int) {
                        defer wg.Done()
                        defer func() { <-semaphore }()

                        for j := 0; j < result.rows; j++ {
                                multiplyWorker(m1, m2, &result, i, j)
                        }
                }(i)
        }

        wg.Wait()

        return result, nil
```

```
}

func abc(m1, m2, result *Matrix, lt, lb, rt, rb int) {
        for i := lt; i < lb; i++ {
                for j := rt; j < rb; j++ {
                        //
                        sum := 0
                        for k := 0; k < m1.cols; k++ {
                                sum += m1.data[i][k] * m2.data[k][j]
                        }
                        result.data[i][j] = sum
                        //
                }
        }
}

// MultiplyMatricesWithBlocks перемножает матрицы с разделением на блоки.
func MultiplyMatricesWithBlocks(m1, m2 *Matrix, maxGoroutines int, w, h int) (Matri
        if m1.cols != m2.rows {
                return Matrix{}, ErrCantMultiply
        }

        if m1.rows%w != 0 || m2.cols%h != 0 {
                return Matrix{}, ErrBlocks
        }

        result := NewMatrix(m1.rows, m2.cols)

        semaphore := make(chan struct{}, maxGoroutines)

        var wg sync.WaitGroup
        wg.Add((m1.rows / w) * (m2.cols / h))

        for i := 0; i < m1.rows/w; i++ {
                for j := 0; j < m2.cols/h; j++ {
                        semaphore <- struct{}{}

                        go func(i, j int) {
                                defer wg.Done()
                                defer func() { <-semaphore }()
```

```
                                abc(m1, m2, &result, i*w, i*w+w, j*h, j*h+h)
                        }(i, j)
                }
        }

        wg.Wait()

        return result, nil
}
```

3 Результаты

Характеристики устройства

- Процессор - Apple M2 Pro
- Оперативная память - 16 Гб
- Операционная система - MacOS Sonoma 14.5

В *Go* нельзя выставлять приоритет потокам

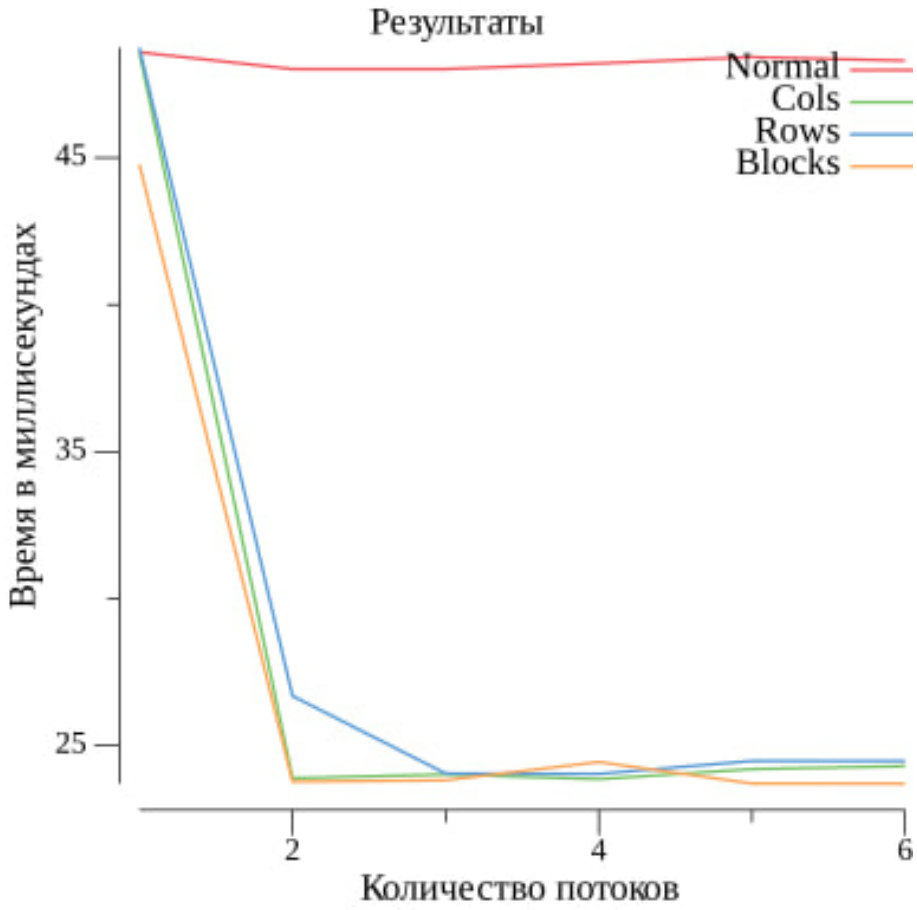
Результаты
Normal
Cols
Rows
Blocks
Время в миллисекундах
45
35
25
2
4
6
Количество потоков

Рис. 1: График