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 №5

«Синхронизация потоков.»

Выполнил
студент группы ИУ9-51Б
Лисов Алексей

Москва, 2024

1 Условие

1. Использование барьерной синхронизации - задача "эволюция". Дан двумерный массив клеток, каждая из которых либо содержит организм (1), либо пуста (0), изначально он заполняется случайными значениями. Каждая клетка проверяет состояние своих соседей (их 8) и изменяет своё по правилам:

   - Живая клетка,вокруг которой < 2 живых клеток, умирает от одиночества.
   - Живая клетка, вокруг которой есть 2 или 3 живых клеток, выживает.
   - Живая клетка, вокруг которой > 3 живых клеток, умирает от перенаселения.
   - Пустая клетка, рядом с которой равно 3 живых соседа, оживает. Реализовать заданное количество шагов моделирования при помощи n потоков. Каждый поток должен вычислить значения в заданной ему полосе матрицы. На каждом шаге результат моделирования необходимо записывать в новую матрицу. По окончании очередного шага необходимо скопировать содержимое новой матрицы в исходную. Шаги между потоками синхронизировать с помощью барьера (ни один из потоков не должен начинать следующий шаг, пока все не закончили текущий). Учесть следующие моменты:
     - у клеток,находящихсянапервойстроке,первомстолбце,последней строке и последнем столбце, соседями являются клетки с противоположной стороны матрицы
     - каждый поток видит только свою часть матрицы, поэтому если ему необ- ходим элемент, которого в его части матрицы нет, он должен каким-либо образом составлять запрос на то, чтобы тот поток, в котором этот элемент есть, его ему предоставил.

2. Использование синхронизаций чтения и записи. Написать программу, в которой каждый из n потоков генерирует заданное количество псевдослучайных чисел от 0 до 1000, каждое из которых он должен помещать в конец однонаправленного связного списка, но только если такого числа там ещё нет. Синхро- низацию доступа к списку необходимо выполнять при помощи двух отдельных 3 блокировок: чтения и записи. Проверку числа на принадлежность списку необходимо выполнить в блокировке на чтение. Далее, если число не обнаружено, выполнить блокировку на запись и перед добавлением числа вновь проверить его на принадлежность списку. По окончании работы потоков ещё раз основным потоком

проверить отсутствие повторяющихся чисел в списке. В отчёт внести только код программы

2 Код решения

Первая часть

```
//Имя Go файла: main.go
//Содержимое Go файла:
package main

import (
        "github.com/BaldiSlayer/rprp/lab5/internal/app"
        "runtime"
)

func main() {
        runtime.GOMAXPROCS(16)
        app.New().Run()
}

----------------------

//Имя Go файла: app.go
//Содержимое Go файла:
package app

import (
        "fmt"
        "github.com/BaldiSlayer/rprp/lab5/internal/conc"
        "github.com/BaldiSlayer/rprp/lab5/internal/defaults"
        "github.com/BaldiSlayer/rprp/lab5/internal/seq"
        "github.com/BaldiSlayer/rprp/lab5/internal/tmeas"
        "math/rand"
)

type App struct{}

func New() *App {
        return &App{}
}
```

```
func (app *App) Run() {
	matrix, newMatrix := getMatrices()

	measureSequential(matrix, newMatrix)

	measureParallel(matrix, newMatrix)
}

func getMatrices() ([][]int, [][]int) {
	matrix := make([][]int, defaults.Rows)
	newMatrix := make([][]int, defaults.Rows)
	for i := range matrix {
		matrix[i] = make([]int, defaults.Cols)
		newMatrix[i] = make([]int, defaults.Cols)
		for j := range matrix[i] {
			matrix[i][j] = rand.Intn(2)
		}
	}

	return matrix, newMatrix
}

func measureSequential(matrix, newMatrix [][]int) {
	tmeas.MeasureFuncTime("Среднее время одной итерации без concurency", func()
		seq.RunSequential(matrix, newMatrix)
	})
}

func measureParallelStep(matrix, newMatrix [][]int, numThreads int) {
	tmeas.MeasureFuncTime(
		fmt.Sprintf("Среднее время одной итерации с %d горутинами", numThre
		func() {
			conc.RunParallel(matrix, newMatrix, numThreads)
		},
	)
}

func measureParallel(matrix, newMatrix [][]int) {
	for _, numThreads := range defaults.Threads {
```

```
                measureParallelStep(matrix, newMatrix, numThreads)
        }
}

----------------------

//Имя Go файла: seq.go
//Содержимое Go файла:
package seq

import "github.com/BaldiSlayer/rprp/lab5/internal/defaults"

func getNewCellValue(cellValue int, liveNeighbors int) int {
        if cellValue == 1 {
                if liveNeighbors < 2 || liveNeighbors > 3 {
                        return 0
                }

                return 1
        }

        if liveNeighbors == 3 {
                return 1
        }

        return 0
}

func updateCell(matrix [][]int, x, y int) int {
        liveNeighbors := 0

        for _, d := range defaults.Directions {
                nx, ny := (x+d.DX+defaults.Rows)%defaults.Rows, (y+d.DY+defaults.Co
                liveNeighbors += matrix[nx][ny]
        }

        return getNewCellValue(matrix[x][y], liveNeighbors)
}

func updateMatrix(matrix, newMatrix [][]int) [][]int {
```

```
        for i := 0; i < defaults.Rows; i++ {
                for j := 0; j < defaults.Cols; j++ {
                        newMatrix[i][j] = updateCell(matrix, i, j)
                }
        }

        return newMatrix
}

func RunSequential(matrix, newMatrix [][]int) {
        for iter := 0; iter < defaults.Iterations; iter++ {
                newMatrix = updateMatrix(matrix, newMatrix)

                for i := range matrix {
                        copy(matrix[i], newMatrix[i])
                }
        }
}

----------------------

//Имя Go файла: models.go
//Содержимое Go файла:
package models

type ElemRequest struct {
        X      int
        Y      int
        Result chan int
}

----------------------

//Имя Go файла: conc.go
//Содержимое Go файла:
package conc

import (
        "github.com/BaldiSlayer/rprp/lab5/internal/defaults"
        "github.com/BaldiSlayer/rprp/lab5/internal/models"
```

```
        "sync"
        "sync/atomic"
)

var (
        barrier          = sync.NewCond(&sync.Mutex{})
        completedThreads = atomic.Int32{}
)

var requestPool = sync.Pool{
        New: func() interface{} {
                return &models.ElemRequest{
                        Result: make(chan int),
                }
        },
}

type RowWorker struct {
        threadChannels []chan models.ElemRequest
        matrix         [][]int
        newMatrix      [][]int
        threadID       int
        numThreads     int
}

func (r *RowWorker) Wait() {
        barrier.L.Lock()

        completedThreads.Add(1)

        if completedThreads.Load() == int32(r.numThreads) {
                completedThreads.Store(0)

                for i := range r.matrix {
                        r.matrix[i] = r.newMatrix[i]
                }

                barrier.Broadcast()
        } else {
                barrier.Wait()
```

```
        }

        barrier.L.Unlock()
}

func (r *RowWorker) Run(start, end int) {
        listenChan := r.threadChannels[r.threadID]

        go func() {
                for req := range listenChan {
                        req.Result <- r.matrix[req.Y][req.X]
                }
        }()

        for iter := 0; iter < defaults.Iterations; iter++ {
                for i := start; i < end; i++ {
                        for j := 0; j < defaults.Cols; j++ {
                                r.newMatrix[i][j] = r.updateCellParallel(
                                        r.matrix,
                                        i, j,
                                        defaults.Rows,
                                        defaults.Cols,
                                        r.numThreads,
                                )
                        }
                }

                r.Wait()
        }

        close(r.threadChannels[r.threadID])
}

func RunParallel(matrix, newMatrix [][]int, numThreads int) {
        var wg sync.WaitGroup

        threadChannels := make([]chan models.ElemRequest, numThreads)
        for i := range threadChannels {
                threadChannels[i] = make(chan models.ElemRequest, 1)
        }
```

```
        wg.Add(numThreads)

        for t := 0; t < numThreads; t++ {
                startRow := t * defaults.Rows / numThreads
                endRow := (t + 1) * defaults.Rows / numThreads
                threadID := t

                go func() {
                        defer wg.Done()

                        worker := RowWorker{
                                threadChannels: threadChannels,
                                matrix:         matrix,
                                newMatrix:      newMatrix,
                                threadID:       threadID,
                                numThreads:     numThreads,
                        }

                        worker.Run(startRow, endRow)
                }()
        }

        wg.Wait()
}

func getLiveness(threadChan chan models.ElemRequest, nx, ny int) int {
        res := make(chan int, 1)

        threadChan <- models.ElemRequest{
                X:      nx,
                Y:      ny,
                Result: res,
        }

        return <-res
}

func getLivenessWithSp(threadChan chan models.ElemRequest, nx, ny int) int {
        req := requestPool.Get().(*models.ElemRequest)
```

```
        defer requestPool.Put(req)

        req.X = nx
        req.Y = ny

        threadChan <- *req

        return <-req.Result
}

func (r *RowWorker) updateCellParallel(matrix [][]int, x, y, rowCount, colCount int
        liveNeighbors := 0

        for _, d := range defaults.Directions {
                // уже и так умрет от перенаселения
                if liveNeighbors > 3 {
                        break
                }

                nx, ny := (x+d.DX+rowCount)%rowCount, (y+d.DY+colCount)%colCount

                ownerThread := nx * numThreads / rowCount

                if ownerThread != x*numThreads/rowCount {
                        liveNeighbors += getLivenessWithSp(r.threadChannels[ownerT

                        continue
                }

                liveNeighbors += matrix[nx][ny]
        }

        // считаем значение
        if matrix[x][y] == 1 {
                if liveNeighbors < 2 || liveNeighbors > 3 {
                        return 0
                }

                return 1
        }
```

```
        if liveNeighbors == 3 {
                return 1
        }

        return 0
}
```

```
//Имя Go файла: defaults.go
//Содержимое Go файла:
package defaults

const (
        Rows       = 1000
        Cols       = 1000
        Iterations = 512
)

var (
        Directions = [...]struct{ DX, DY int }{
                {-1, -1}, {-1, 0}, {-1, 1},
                {0, -1}, {0, 1},
                {1, -1}, {1, 0}, {1, 1},
        }

        Threads = [...]int{2, 4, 8, 16}
)
```

```
//Имя Go файла: tmeas.go
//Содержимое Go файла:
package tmeas

import (
        "fmt"
        "github.com/BaldiSlayer/rprp/lab5/internal/defaults"
        "time"
```

```
)

func MeasureFuncTime(logString string, f func()) {
        start := time.Now()

        f()

        sequentialDuration := time.Since(start)
        fmt.Printf("%s %v\n", logString, sequentialDuration/time.Duration(defaults
}
```

Вторая часть

```
//Имя Go файла: main.go
//Содержимое Go файла:
package main

import "github.com/BaldiSlayer/rprp/lab52/internal/app"

func main() {
        s := app.New()

        s.Run()
}
```

```
//Имя Go файла: app.go
//Содержимое Go файла:
package app

import (
        "log"
        "math/rand"
        "sync"

        "github.com/BaldiSlayer/rprp/lab52/internal/defaults"
        "github.com/BaldiSlayer/rprp/lab52/internal/llist"
)
```

```
type App struct{}

func New() *App {
        return &App{}
}

func Filling(list *llist.LinkedList) {
        wg := sync.WaitGroup{}

        wg.Add(defaults.NumThreads)

        for i := 0; i < defaults.NumThreads; i++ {
                go func() {
                        defer wg.Done()

                        for j := 0; j < defaults.NumValuesPerThread; j++ {
                                list.AddIfNotExists(rand.Intn(defaults.ValueMaxRanc
                        }
                }()
        }

        wg.Wait()
}

func (app *App) Run() {
        list := llist.New()

        Filling(list)

        log.Println(list)

        if list.CheckForDuplicates() {
                log.Println("Дубликаты найдены в списке!")

                return
        }

        log.Println("Дубликатов не найдено!!!")
}
```

```
----------------------

//Имя Go файла: defaults.go
//Содержимое Go файла:
package defaults

const (
        // NumThreads - количество потоков
        NumThreads = 10
        // NumValuesPerThread сколько значений добавляет каждый из потоков
        NumValuesPerThread = 10
        // ValueMaxRand - максимальное число, которое может быть в list
        ValueMaxRand = 10000
)

----------------------

//Имя Go файла: llist.go
//Содержимое Go файла:
package llist

import (
        "fmt"
        "strings"
        "sync"
)

type Node struct {
        value int
        next  *Node
}

type LinkedList struct {
        head *Node
        mu   sync.RWMutex
}

func New() *LinkedList {
        return &LinkedList{}
```

```
}

func (list *LinkedList) AddIfNotExists(value int) {
        list.mu.RLock()

        current := list.head
        for current != nil {
                if current.value == value {
                        list.mu.RUnlock()

                        return
                }
                current = current.next
        }

        list.mu.RUnlock()

        list.mu.Lock()
        defer list.mu.Unlock()

        current = list.head
        for current != nil {
                if current.value == value {
                        return
                }

                current = current.next
        }

        newNode := &Node{value: value}
        if list.head == nil {
                list.head = newNode

                return
        }

        current = list.head
        for current.next != nil {
                current = current.next
        }
```

```
        current.next = newNode
}

func (list *LinkedList) CheckForDuplicates() bool {
        list.mu.RLock()
        defer list.mu.RUnlock()

        seen := make(map[int]bool)

        current := list.head
        for current != nil {
                if seen[current.value] {
                        return true
                }

                seen[current.value] = true
                current = current.next
        }

        return false
}

func (list *LinkedList) String() string {
        list.mu.RLock()
        defer list.mu.RUnlock()

        var sb strings.Builder

        current := list.head
        for ; current != nil; current = current.next {
                sb.WriteString(fmt.Sprintf("%d ", current.value))
        }

        return sb.String()
}

----------------------
```

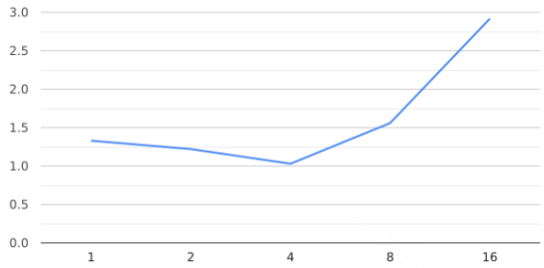

Рис. 1: График

3 Результат

Характеристики устройства

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

Запускалось на матрице 1000x1000

```
Среднее время одной итерации без concurency 18.540191ms
Среднее время одной итерации с 2 горутинами 15.520603ms
Среднее время одной итерации с 4 горутинами 10.328018ms
Среднее время одной итерации с 8 горутинами 9.848462ms
Среднее время одной итерации с 16 горутинами 11.355249ms
```