Университет ИТМО
Кафедра ИПМ

Лабораторная работа №1
По предмету:
«ТЕОРИЯ РАСПРЕДЕЛЁННЫХ И ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ»
Вариант: 1

Работу выполнила:
Студентка Гулямова С. И.
Группа:
P4115

Санкт-Петербург
2018 г

Задание:

Вариант 1.

Вычисление суммарной площади фигур: треугольника, прямоугольника, круга. Формат входного файла: описание фигур размещается на каждой новой строке, данные в строке разделяются пробелом. Первым словом идет тип фигуры: (“triangle”, “box”, “circle”), дальше следуют характеристики фигуры в виде вещественных чисел (разделитель – ‘.’): для треугольника – 3 длины сторон, прямоугольник – ширина и высота, круг – радиус. В результате необходимо вывести ответ в виде суммы площадей всех представленных фигур.

Выполнение:

```
   func calculate(coresCount: UInt8, completionHandler: @escaping(_
result: [Figure]) -> Void) {
        self.coresCount = coresCount
        calculate() { [weak self] in
            completionHandler(self?.figures ?? [])
        }
    }

    private func calculate(completionHandler: @escaping() -> Void)
{
        areasSum = 0

        let finish = BlockOperation(block: {
            completionHandler()
        })
        figures = GeneratorService.shared.figures
        for figure in figures {
            let opertation = BlockOperation {
                figure.square()
            }
            finish.addDependency(opertation)
            operationQueue.addOperation(opertation)
        }
        operationQueue.addOperation(finish)
   }
```

Выше приведен листинг кода производящего расчеты.

Ниже показан листинг кода, изменявшего число одновременно выполняющихся потоков:

```
private(set) var coresCount: UInt8 = 0 {
        didSet {
            operationQueue.maxConcurrentOperationCount =
Int(coresCount > 0 ? coresCount : 1)
        }
    }
```

Figure - абстрактный класс, от которого наследуются классы каждого типа фигур.

```
class Figure: Codable {
    let type: Figures!
    var area = 0.0

    func square() -> Double {
        return 0.0
    }

    init(type: Figures) {
        self.type = type
    }

    required init(from decoder: Decoder) throws {
        let values = try decoder.container(keyedBy:
CodingKeys.self)
        self.type = try values.decode(Figures.self, forKey: .type)

    }
}
```

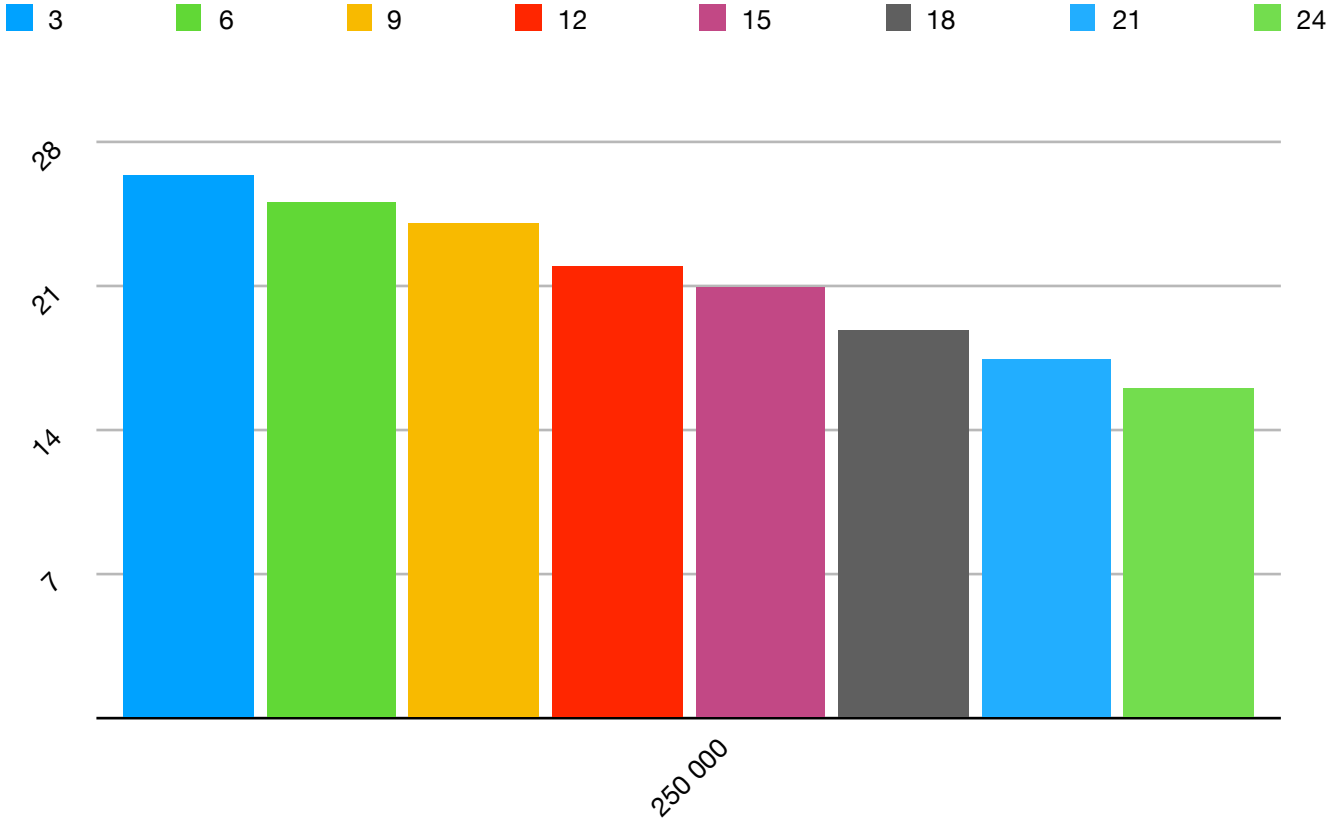
Генерация входных данных:

```
func generateFigures(count: UInt) -> [Figure] {
        if _figures == nil {
            _figures = [Figure]()
        }
        _figures?.removeAll()
        self.count.value = count
        for _ in 0..<count {
            _figures?.append(generateFigure())
            self.count.value = self.count.value + 1
        }
```

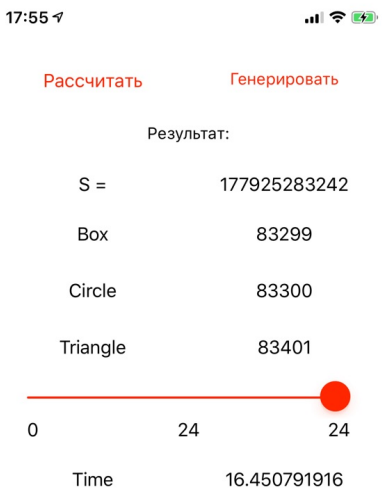
```
        return _figures!
    }

    private func generateFigure() -> Figure {
        return FigureConfigurator.shared.randomFigure()
    }

final class FigureConfigurator {

    public static let shared = FigureConfigurator()

    private init() {

    }

    func randomFigure() -> Figure {
        return figure(type: Figures.randomFigure())
    }

    func figure(type: Figures) -> Figure {
        switch type {
        case .box:
            return configBox()
        case .circle:
            return configCircle()
        case.triangle:
            return configTriangle()
        }
    }

    private func configBox() -> BoxFigure {
        return BoxFigure(UInt8.random(), UInt8.random())
    }

    private func configCircle() -> CircleFigure {
        return CircleFigure(UInt8.random())
    }

    private func configTriangle() -> TriangleFigure {
        return TriangleFigure(UInt8.random(), UInt8.random(),
UInt8.random())
    }
}
```

Результаты:

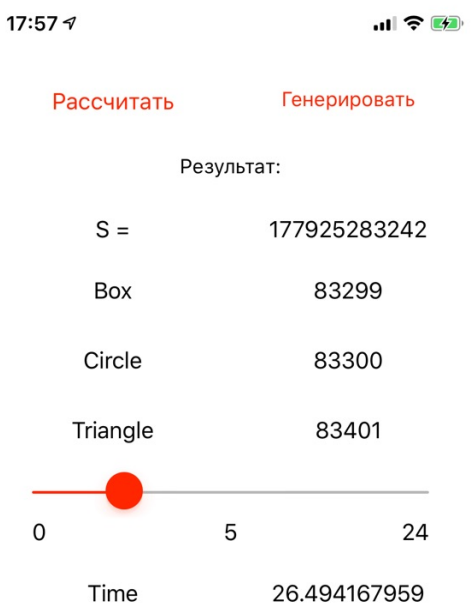
В таблице приведено время выполнения работы программы в секундах.



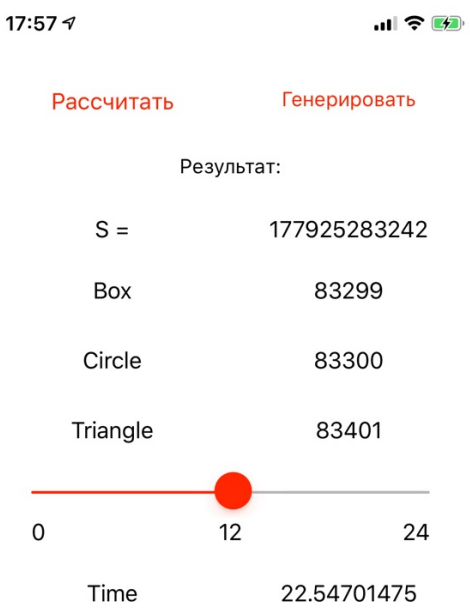
Результаты запуска приложения:



При 25 потоках и 250 000 фигур



При 3 потоках и 250 000 фигурах



При 12 потоках и 250 000 фигурах

Вывод:
Так как сложность выполняемой операции незначительная, сделать однозначные выводы о эффективности разделения на потоки довольно сложно. Расходы на создание и управление потоками не оправдывают себе и занимают большую часть от времени работы приложения. Тем не менее, на большем числе потоков можно наблюдать некоторое увеличение скорости работы приложения.