Facultad de Ingenierías y Matemáticas
Carrera Profesional de Ingeniería de Software
Computación Distribuida y Paralela

Metodo del trapecio Secuencial/Paralelo

Docente:
Richart Smith Escobedo Quispe
r.escobedo@ulasalle.edu.pe

Desarrollado por:
José Alfredo Pinto Villamar
jpintov@ulasalle.edu.pe

15 de noviembre de 2022

1. Introducción

Como se vio en la tarea anterior el lenguaje Go es un lenguaje de programación concurrente, esto quiere decir que es capaz de ejecutar varias tareas al mismo tiempo, en este caso se va a estudiar el flujo de control secuencial y paralelo en Go.

2. Ejecuciones en Sequencial

2.1. Código en Go

```
1  package main
2
3  import (
4      "fmt"
5      "math"
6      "time"
7  )
```

Aquí simplemente se importan las librerías necesarias para el desarrollo del código.

```
9  func TrapezoidRule(f func(float64) float64, a, b float64, n int) float64 {
10     h := (b - a) / float64(n)
11     sum := 0.5 * (f(a) + f(b))
12     for i := 1; i < n; i++ {
13         sum += f(a + float64(i)*h)
14     }
15     return sum * h
16 }
```

En esta parte del código se declara la función `TrapezoidRule`, esta función recibe como parámetros una función, un valor inicial, un valor final y un número de iteraciones.

```
18 func main() {
19
20     f := func(x float64) float64 {
21         return ((math.Pow(x, 2) + 1) / 2)
22     }
23
24     n := 1000
25
26     start := time.Now()
27     fmt.Println(TrapezoidRule(f, 5, 20, n))
28     elapsed := time.Since(start).Nanoseconds()
29     fmt.Println(elapsed)
30 }
```

Es aquí donde invoco a la función `TrapezoidRule` y le paso como parámetros la función que se van a ejecutar, el valor inicial, el valor final y el número de iteraciones. También se calcula el tiempo que tarda en ejecutarse la función.

3. Paralelo

3.1. Código en Go

```
1  package main
2
3  import (
4      "fmt"
5      "math"
6      "time"
7  )
```

En la siguiente parte del código se importan las librerías necesarias para el desarrollo del código.

```
9     func worker(jobs chan int, results chan float64) {
10        f := func(x float64) float64 {
11            return ((math.Pow(x, 2) + 1) / 2)
12        }
13
14        for n := range jobs {
15            results <- TrapezoidRule(f, 5, 20, n)
16        }
17    }
```

En la siguiente función `worker` se crea un canal de entrada y otro de salida, en este caso el canal de entrada es `jobs` y el canal de salida es `results`. Aquí se llama a la función $f(x) = \frac{x^2+1}{2}$ y se le asigna el valor de n que se recibe por el canal de entrada. Y se envía el resultado por el canal de salida.

```
19    func TrapezoidRule(f func(float64) float64, a, b float64, n int) float64 {
20        h := (b - a) / float64(n)
21        sum := 0.5 * (f(a) + f(b))
22        for i := 1; i < n; i++ {
23            sum += f(a + float64(i)*h)
24        }
25        return sum * h
26    }
```

En esta parte se declara la función `TrapezoidRule`, esta función recibe como parámetros una función, un valor inicial, un valor final y un número de iteraciones.

```
28    func main() {
29
30        n := 1000
31
32        jobs := make(chan int, n)
33        results := make(chan float64, n)
34
35        go worker(jobs, results)
36        go worker(jobs, results)
37
38        start := time.Now()
39        for i := 0; i < n; i++ {
40            jobs <- i
41        }
42        elapsed := time.Since(start).Seconds()
43        fmt.Println(elapsed)
44
45        close(jobs)
46
47        // for i := 0; i < n; i++ {
48        //  fmt.Println(<-results)
49        // }
50    }
```

Y como parte final, se crea la clase main donde se asignan workers, se envían los valores de n por el canal de entrada y se recibe el resultado por el canal de salida. También se declara el numero de iteraciones y los resultados que saldrán por los canales. También se calcula el tiempo en que se tardan en ejecutar cada una de las funciones. Inclusive se puede crear más workers para que el trabajo sea más rápido/eficiente. Al final se cierra `jobs` y si se desea se pueden imprimir los resultados.

4. Resultados

Las siguientes gráficas son muestra de las pruebas que se hicieron. Se realizó 10 pruebas en cada ciclo, es decir, en 1 trapecio, se realizaron 10 pruebas para sacar el promedio de tiempo. Y así sucesivamente.

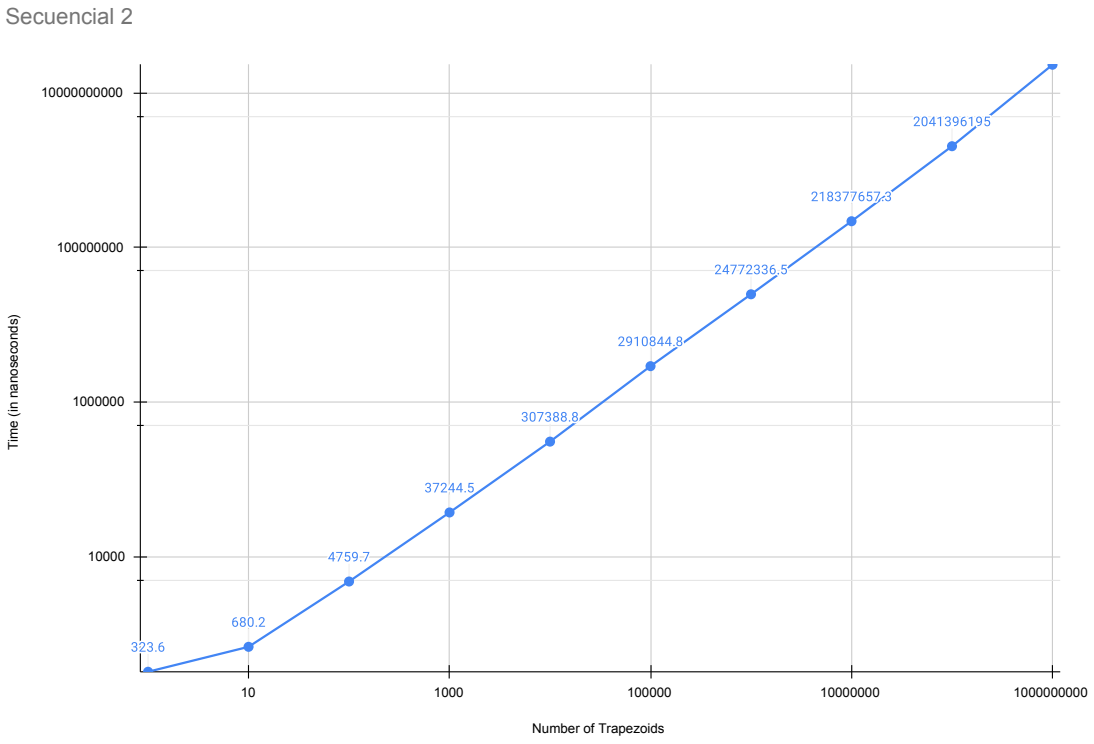

Figura 1: Tiempo de ejecución de la regla del trapecio en programación secuencial.

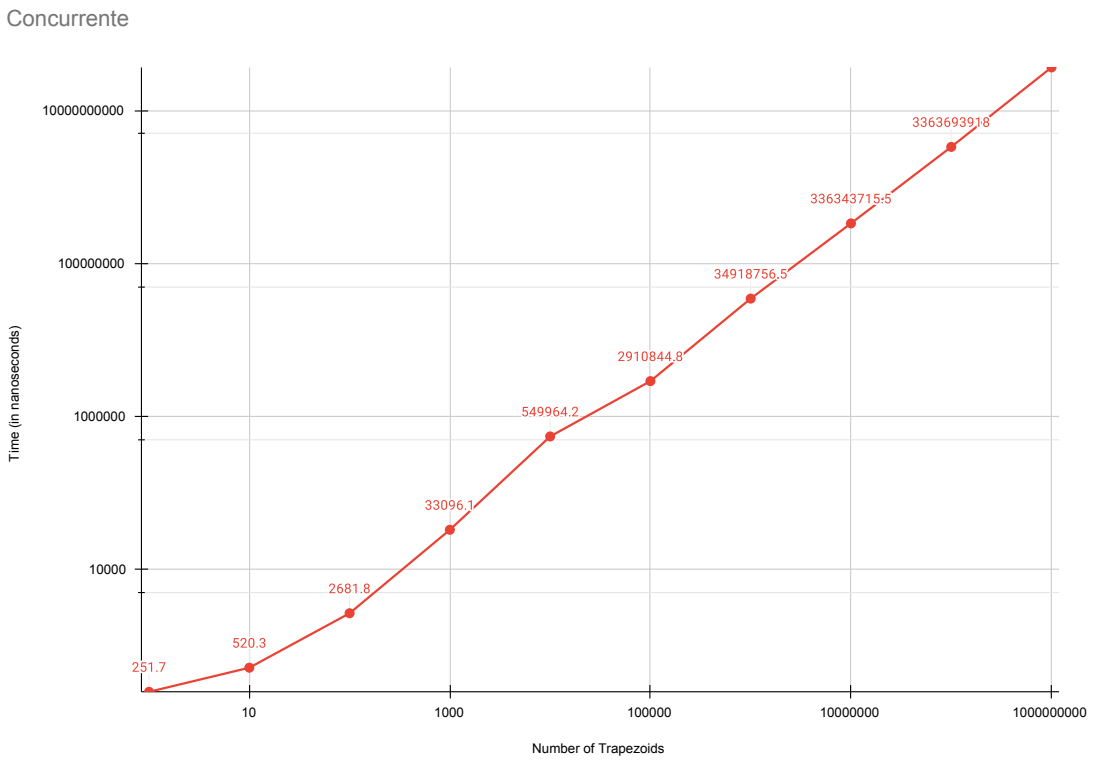

Figura 2: Tiempo de ejecución de la regla del trapecio en programación concurrente.

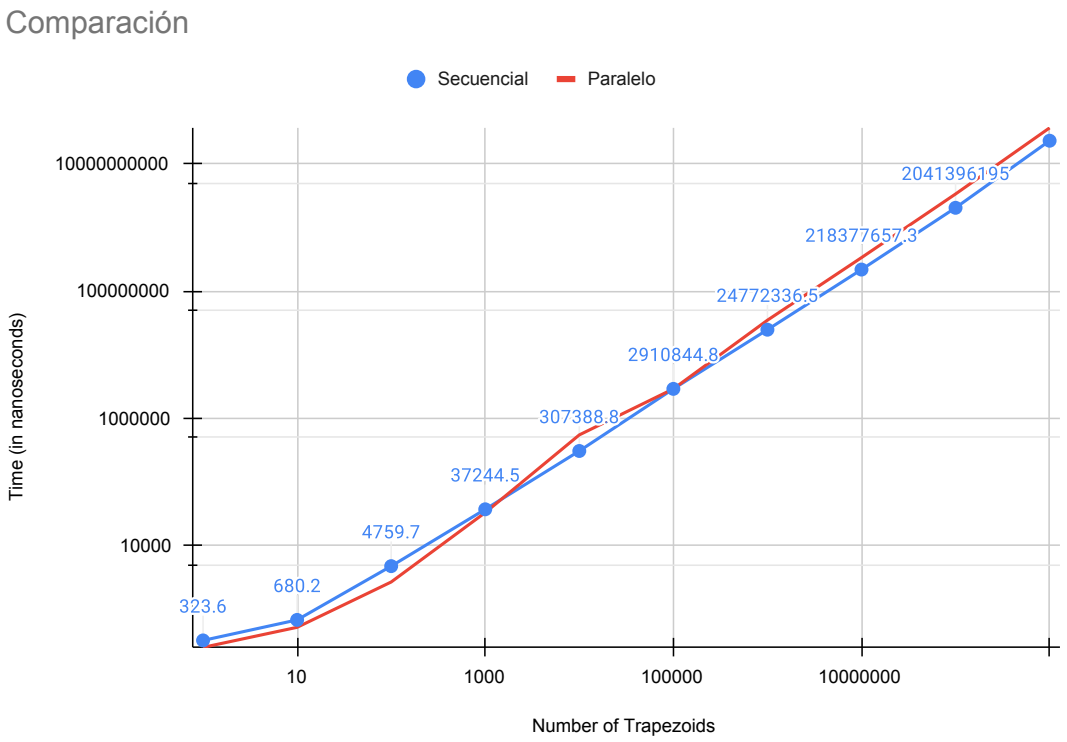

Figura 3: Comparación entre la programación secuencial y concurrente.

5. Repositorio

- https://github.com/pintovillamar/computacion-distribuida-y-paralela/tree/main/tarea02-golang