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

Estudio del Flujo de control Secuencia / Paralelo en Go

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

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

10 de noviembre de 2022

1. Introducción

En este trabajo se realizaron ensayos de los tiempos de ejecución para hallar el área de una función matemática, usando así la regla del trapecio.

En el cálculo, la regla trapezoidal también conocida como la regla del trapecio es una técnica de aproximación numérica de la integral definida.

$$\int_a^b f(x)dx \tag{1}$$

La regla trapezoidal funciona aproximando la región bajo el gráfico de la función $f(x)$ como un trapecio y calculando su área.

$$\int_a^b f(x)dx \approx (b-a)\cdot \tfrac{1}{2}(f(a)+f(b)). \tag{2}$$

La regla del trapecio puede ser vista como el resultado obtenido al promediar los valores de la función en los extremos del intervalo a y b. La regla del trapecio es un caso especial de la regla de Simpson, que es una regla de integración numérica más general. Incluso, la integral se puede aproximar aún mejor dividiendo el intervalo de integración aplicando la regla del trapecio a cada subintervalo, y sumando los resultados. En la práctica sería así:

Sea: $\{x_k\}$ una partición de $[a,b]$ tal que $a = x_0 < x_1 < \cdots < x_{N-1} < x_N = b$ y Δx_k es la longitud de k-ésimo subintervalo (es decir, $\Delta x_k = x_{k+1} - x_k$), entonces:

$$\int_a^b f(x)dx \approx \sum_{k=1}^{N} \frac{f(x_{k-1})+f(x_k)}{2}\Delta x_k \tag{3}$$

Cuando la partición tiene un número par de subintervalos, la regla del trapecio es exacta y podemos decir que todos los Δx_k tienen el mismo valor Δx, entonces la fórmula se puede simplificar para calcular la eficiencia factorizando Δx :

$$\int_a^b f(x)dx \approx \frac{\Delta x}{2}(f(x_0)+2f(x_1)+2f(x_2)+\cdots+2f(x_{N-1})+f(x_N)) \tag{4}$$

$$\int_a^b f(x)dx \approx \frac{\Delta x}{2}\sum_{k=1}^{N}(f(x_{k-1})+f(x_k)) \tag{5}$$

2. Ejecuciones en Sequencial

2.1. Código en Go

```
1  package main
2
3  import (
4    "fmt"
5    "math"
6    "strings"
7    "text/template"
8    "time"
9  )
```

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

```
11  func format(s string, v interface{}) string {
12    t, b := new(template.Template), new(strings.Builder)
13    template.Must(t.Parse(s)).Execute(b, v)
14    return b.String()
15  }
```

Esta función `format` me sirve para darle formato a mis `outputs` de la consola.

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

Esta es la parte más importante, y es dónde se aplica la regla del trapecio. Yo la hice un poco diferente a lo planteando en clase porque pienso que sin utilizar arrays puede llegar a ser más eficiente. En este caso, uso un bucle for que va desde 1 hasta n, y en cada iteración se va sumando el valor de la función en el punto x_i al valor de la suma. Y retorno el valor de la suma multiplicado por el valor de h.

```
26  f := func(x float64) float64 {
27        return ((math.Pow(x, 2) + 1) / 2)
28    }
29
30  n := 1
31    start := time.Now()
32    elapsed := time.Since(start).Nanoseconds()
33    result := TrapezoidRule(f, 5, 20, n)
34    area := format("Area: {{ . }}.", result)
35    fmt.Println(area)
36    time := format("Time: {{ . }} in ns.", elapsed)
37    fmt.Println(time)
38    fmt.Println()
```

Y como parte final agregamos el `main` los `prints` para que podamos ver los resultados. También podemos ir cambiando el valor de n el cual es el número de trapecios que se van a utilizar.

3. Paralelo

3.1. Código en Go

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

En este caso, importamos la librería `sync` para poder utilizar `waitgroup`.

```
9   var times time.Duration
10  var wg sync.WaitGroup
```

En esta parte se declaran las variables que se van a utilizar para hacer el cálculo en paralelo.

```
12  type Trapezoid struct {
13    f func(float64) float64
14    a float64
15    b float64
16    n int
17  }
```

En este caso he creado una clase para hacerlo más sencillo al momenento de ingresar los datos.

```
19  func (t *Trapezoid) TrapezoidRule() float64 {
20    h := (t.b - t.a) / float64(t.n)
21    sum := 0.5 * (t.f(t.a) + t.f(t.b))
22    for i := 1; i < t.n; i++ {
23      wg.Add(1)
24      sum += t.f(t.a + float64(i)*h)
25      wg.Done()
26    }
27    return sum * h
28  }
```

Atención a la la línea 23, donde se agrega un `waitgroup` para que el programa espere a que se termine de ejecutar el bucle for. Y en la línea 25 acaba, y está listo para volver a llamar función.

```
30
31  func main() {
32    f := func(x float64) float64 {
33      return ((x*x + 1) / 2)
34    }
35    t := Trapezoid{f, 5, 20, 10}
36    start := time.Now()
37    result := t.TrapezoidRule()
38    elapsed := time.Since(start).Nanoseconds()
39    defer fmt.Println(elapsed)
40    area := fmt.Sprintf("Area: %v", result)
41    fmt.Println(area)
42    wg.Wait()
```

Y es aquí donde se llama a la función `TrapezoidRule` y se imprime el resultado. Tenindo en cuenta que también se llamará a la función `Wait`, para ejecutar los procesos en paralelo.

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.

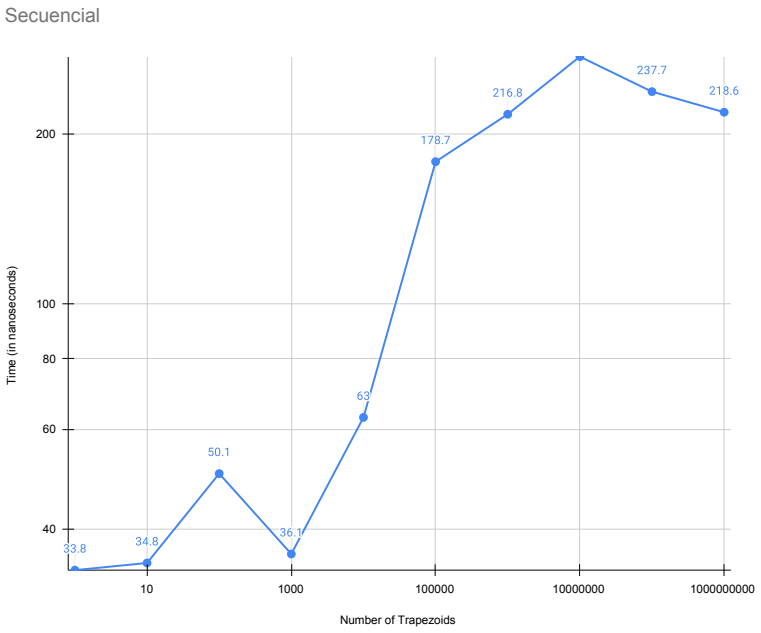

Figura 1: Tiempo de ejecución de la regla del trapecio en programación secuencial.

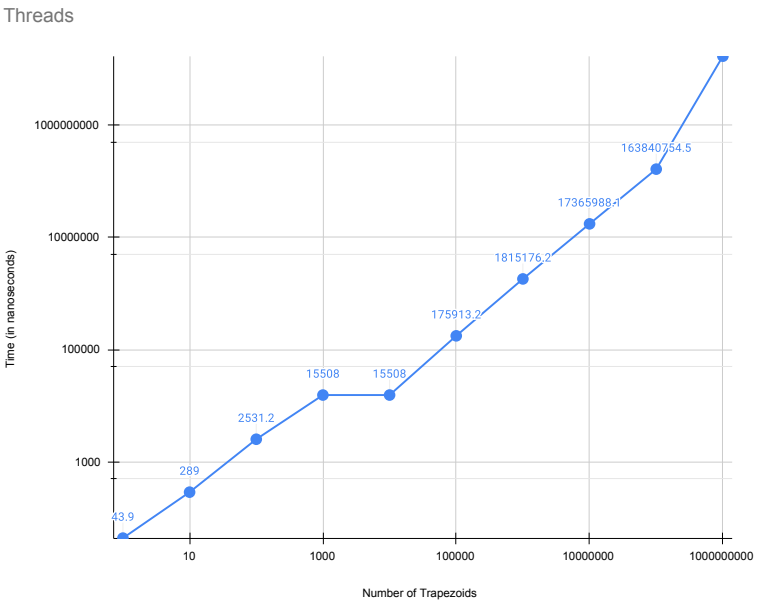

Figura 2: Tiempo de ejecución de la regla del trapecio en programación paralela.

5. Repositorio

- https://github.com/pintovillamar/computacion-distribuida-y-paralela/tree/main/tarea01-golang