TECNOLÓGICO DE MONTERREY®

TC2037 – IMPLEMENTACIÓN DE MÉTODOS COMPUTACIONALES
GRUPO 604

Actividad Integradora 5.2
Programación Paralela y Concurrente

Brenda Elena Saucedo González – A00829855
José Ángel Rentería Campos – A00832436
Diego Alberto Baños Lopez – A01275100

3 de junio de 2022

Reporte de los Resultados Obtenidos

En la presente actividad se desarrollaron 2 versiones para una misma situación problema, en donde se busca calcular la suma de todos los números primos menores a un número *n*. Las versiones desarrolladas fueron de manera secuencial y paralela, en donde se busca que en esta última, el tiempo de ejecución sea menor que la versión secuencial.

Versión Secuencial

Actividad 5.2 > Secuencial > main.go

```go
26  var n float64
27
28  //Checa si los numeros son primos
29  func check_prime(n float64) bool {
30      if n <= 1 {
31          return false
32      }
33      for i := 2.0; i <= math.Sqrt(n); i += 1.0 {
34          if math.Mod(n, i) == 0 {
35              return false
36          }
37      }
38      return true
39  }
40
41  //Funcion para realizar la parte secuencial de la actividad
42  func sec_prime(limit int) int {
43      var sum int
44      for i := 2; i < limit; i++ {
45          if check_prime(float64(i)) == true {
46              sum += int(i)
47          }
48      }
49      return sum
50  }
51
52  func main() {
53      //n sera el numero a analizar
54      n := 5000000
55      //imprimimos el resultado
56      fmt.Println(sec_prime(n))
```

PROBLEMS (1) OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] go run "c:\Users\nenas\OneDrive\Documents\GitHub\Equipo-Racket\Actividad 5.2\Secuencial\main.go"
838596693108

[Done] exited with code=0 in 51.827 seconds
```

Versión Paralela

Actividad 5.2 > Paralelo > main.go

```go
//Funcion para sumar los numeros primos
func pll_prime(begining, ending, step int, ch chan int) {
    var sum int
    for i := begining; i < ending; i += step {
        if check_prime(float64(i)) == true {
            mutex.Lock()
            sum += i
            mutex.Unlock()
        }
    }
    ch <- sum
}
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] go run "c:\Users\nenas\OneDrive\Documents\GitHub\Equipo-Racket\Actividad 5.2\Paralelo\main.go"
838596693108

[Done] exited with code=0 in 10.513 seconds
```

Cálculo del Speedup

$$S_p = \frac{T_1}{T_p}$$

En donde, utilizando como referencia los parámetros obtenidos por uno de los miembros del equipo, se obtuvo:

- p = 6
- T_1 = 62.389
- T_p = 15.573

$$S_p = \frac{T_1}{T_p} = \frac{62.389}{15.573} = 4.00622873$$

Conclusión

La versión secuencial de la situación problema obtuvo un tiempo de ejecución de aproximadamente un minuto, en cambio, la versión paralela demostró tener un tiempo de ejecución mucho menor a la versión anterior (secuencial), en donde se pudo analizar que dicho tiempo se redujo hasta un 75% del tiempo de la versión secuencial. Esto demuestra la importancia de la programación paralela en programas exigentes o con un tiempo de ejecución tardado, disminuyendo considerablemente el tiempo de ejecución del programa al utilizar de manera más eficiente la capacidad de procesamiento de las computadoras utilizadas.