Instituto Tecnológico y de Estudios Superiores de Monterrey

Jesús Ramírez Delgado

A01274723

Implementacion de Metidos Computacionales

Actividad Integradora 3: Programación Paralela

En esta actividad integradora 3 lo que se pedía era hacer una programación paralela en C++, el caso a resolver es:

Escribe dos versiones de un programa que calcule la suma de todos los n´umeros primos menores a 5,000,000 (cinco millones):

• La primera versión debe ser una implementación convencional que realice el cómputo de manera secuencial.

• La segunda versión debe realizar el cómputo de manera paralela a través de los mecanismos provistos por el lenguaje siendo utilizado (por ejemplo, places o la funci´on pmap). Debes procurar paralelizar el código aprovechando todos los n´ucleos disponibles en tu sistema.

Y finalmente se debe calcular el SpeedUp con la formula;

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

Para resolver esta situación problema utilice C++ con librerías clave para poder desarrollar el código.

```cpp
#include <iostream>
#include <cmath>
#include <chrono>
#include <vector>
#include <thread>
```

1. Primero se definió 2 funciones afuera de la función **main** que ayudaran a desarrollar el código.
    a. Función para comprobar si es un numero primo.

```cpp
bool esPrimo(int n) {
    if (n < 2)
        return false;
    for (int i = 2; i <= std::sqrt(n); ++i) {
        if (n % i == 0)
            return false;
    }
    return true;
}
```

b. Función para calcular la suma de números primos en un rango.

```
21  void calcularSumaPrimos(int inicio, int fin, long long& suma) {
22      for (int i = inicio; i < fin; ++i) {
23          if (esPrimo(i))
24              suma += i;
25      }
26  }
```

2. Ahora dentro de la función **main** primero definimos 2 variables que nos ayudaran a implementar las soluciones para cada tipo de ejecución.

```
28  int main() {
29      const int limite = 5000000;
30      long long suma = 0;
```

Limite y **suma** que sirven para establecer el límite requerido por el problema y un contador respectivamente.

3. Ahora corresponde la implementación de la forma secuencial

```
32      // Versión secuencial
33
34      auto start_secuencial = std::chrono::high_resolution_clock::now();
35
36      for (int i = 2; i < limite; ++i) {
37          if (esPrimo(i))
38              suma += i;
39      }
40
41      auto end_secuencial = std::chrono::high_resolution_clock::now();
42      std::chrono::duration<double> duration_secuencial = end_secuencial - start_secuencial;
43      double tiempo_secuencial = duration_secuencial.count();
44
45      cout << "Suma de primos (versión secuencial): " << suma << endl;
46      cout << "Tiempo de ejecución (secuencial): " << tiempo_secuencial << " segundos" << endl;
```

En este bloque de código lo que realiza es:

a. Capturar el momento en que inicia la ejecución de inicio.
b. Por medio de un ciclo for recorre todos los números a partir del 2 hasta el límite, en este caso 500000. Y si **i** es primo se suma al contador.
c. Captura cuando finaliza la ejecución.
d. Calcula el tiempo de ejecución restando los tiempos anteriores.

e. Obtiene el tiempo de ejecución.

f. Imprime la suma y el tiempo de ejecución.

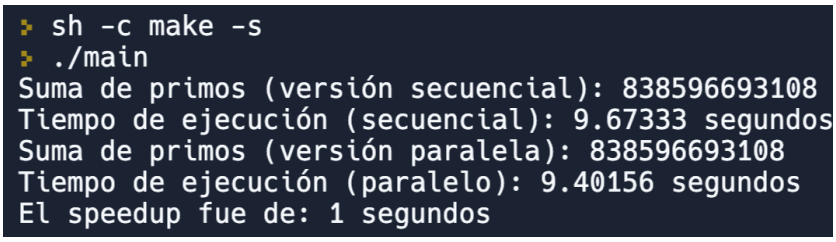
4. Posteriormente se realizó la función para la solución paralela, siguiente los mismos pasos pero en este caso con la ayuda de <threads> paralelizamos la ejecución.

```
48      // Versión paralela
49
50      const int num_hilos = std::thread::hardware_concurrency();
51      std::vector<std::thread> hilos(num_hilos);
52      std::vector<long long> sumas_parciales(num_hilos, 0);
53      long long sumaParalela = 0;
54
55      auto start_paralelo = std::chrono::high_resolution_clock::now();
56
57      int intervalo = limite / num_hilos;
58      for (int i = 0; i < num_hilos; ++i) {
59          int inicio = i * intervalo;
60          int fin = (i == num_hilos - 1) ? limite : (i + 1) * intervalo;
61          hilos[i] = std::thread(calcularSumaPrimos, inicio, fin, std::ref(sumas_parciales[i]));
62      }
63
64      for (int i = 0; i < num_hilos; ++i) {
65          hilos[i].join();
66          sumaParalela += sumas_parciales[i];
67      }
68
69      auto end_paralelo = std::chrono::high_resolution_clock::now();
70      std::chrono::duration<double> duration_paralelo = end_paralelo - start_paralelo;
71      double tiempo_paralelo = duration_paralelo.count();
72
73      cout << "Suma de primos (versión paralela): " << sumaParalela << endl;
74      cout << "Tiempo de ejecución (paralelo): " << tiempo_paralelo << " segundos" << endl;
```

5. Finalmente calculamos el SpeedUp con la formula previa.

```
76      // SpeedUp
77
78      long long speedUp;
79      speedUp = tiempo_secuencial / tiempo_paralelo;
80
81      cout << "El speedup fue de: " << speedUp << " segundos" << endl;
82
83      return 0;
84  }
```

El resultado de la ejecución fue:



Se puede observar que el resultado es el mismo en ambas ejecuciones como es de esperarse, sin embargo, el tiempo de ejecución fue diferente, siendo la solución paralela más rápido.

Esta actividad fue gratificante sobre todo porque fue en C++ un lenguaje con el que me siento cómodo. En este caso se demuestra que la paralelización es efectiva al momento de dividir la carga de trabajo para ejecutar un programa y puede significar mejorar significantes en optimización. Considero que la paralización es una herramienta poderosa y sobre todo en el hardware actual donde los procesadores cada vez tienen más núcleos e hilos.