

Computacion paralela Trabajo 01

ALUMNO: MACEDO PINTO LUIS MIGUEL COD: 196626

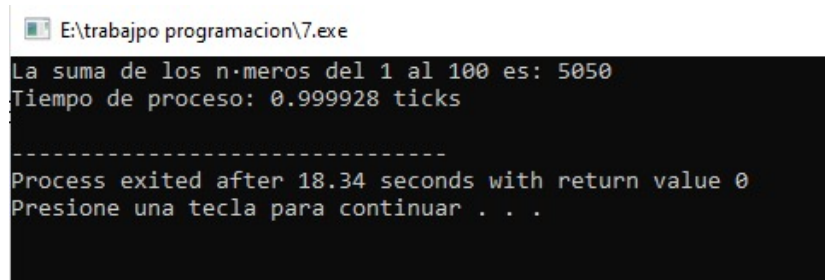
09 Mayo 2024

1 Realizar la impresion de la suma de los números del 1 al 100

1.1 ALGORITMO PARALELO

1.2 Proceso Paralelo: En un proceso paralelo, utilizamos múltiples hilos de ejecución para realizar tareas simultáneamente. Para sumar los números del 1 al 100 en paralelo, podríamos dividir el rango de números en partes más pequeñas y asignar cada parte a un hilo. Cada hilo calcularía la suma de su parte asignada, y luego combinaremos los resultados parciales para obtener la suma total.

1.3 Algoritmo: 1. Dividir el rango de números del 1 al 100 en partes iguales, por ejemplo, 4 partes de 25 números cada una. 2. Crear 4 hilos (uno por cada parte) y asignar a cada hilo una parte del rango de números. 3. Cada hilo calcula la suma de los números en su parte asignada. 4. Al finalizar, combinar los resultados parciales de todos los hilos para obtener la suma total.



```
E:\trabajo programacion\7.exe
La suma de los n-meros del 1 al 100 es: 5050
Tiempo de proceso: 0.999928 ticks

-----
Process exited after 18.34 seconds with return value 0
Presione una tecla para continuar . . .
```

Figure 1: Nos muestra la suma de numeros del 1 al 100

Listing 1: Código en C++

```
#include <iostream>
#include <omp.h>
using namespace std;
int main() {
    const int N = 100;
    int suma_total = 0;
    double inicio = omp_get_wtime(); // Tiempo de inicio
    #pragma omp parallel for reduction(+:suma_total)
    for (int i = 1; i <= N; ++i) {
        suma_total += i;
    }
    double fin = omp_get_wtime(); // Tiempo de finalizaci n

    cout << "La suma de los n meros del 1 al 100 es: " << suma_total << endl;

    double tick = omp_get_wtick();
    cout << "Tiempo de proceso: " << (fin - inicio) / tick << " ticks" << endl;

    return 0;
}
```

1.4 ALGORIMTO CONCURRENTES

Listing 2: Código en C++

```
#include <iostream>
```

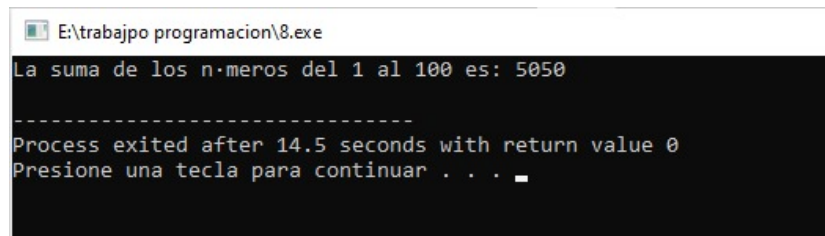


Figure 2: Impresión de manera concurrente del 1 al 100

```
#include <thread>
#include <mutex>
using namespace std;
mutex mtx; // Mutex para garantizar la exclusión mutua en la suma total

void sumar_rango(int inicio, int fin, int& suma_total) {
    int suma_parcial = 0;
    for (int i = inicio; i <= fin; ++i) {
        suma_parcial += i;
    }

    // Agregar la suma parcial a la suma total de manera segura con
    mtx.lock();
    suma_total += suma_parcial;
    mtx.unlock();
}

int main() {
    const int N = 100;
    int suma_total = 0;

    thread t1(sumar_rango, 1, N/2, ref(suma_total)); // Primer hilo
    thread t2(sumar_rango, N/2 + 1, N, ref(suma_total)); // Segundo

    t1.join();
    t2.join();

    cout << "La suma de los n-meros del 1 al 100 es: " << suma_total;
```

```
    return 0;  
}
```

1.5 Algoritmo Concurrente

1.6 Proceso Concurrente: En un proceso concurrente, las tareas se ejecutan de manera intercalada y no necesariamente en paralelo. Podríamos tener dos tareas que se ejecutan alternativamente, cada una sumando una parte del rango de números.

1.7 Algoritmo: 1.Dividir el rango de números del 1 al 100 en dos partes iguales. 2. Crear dos hilos, cada uno ejecutando una tarea para sumar los números en su parte asignada. 3. Alternar entre la ejecución de las dos tareas hasta que ambas hayan terminado. 4. Al finalizar, combinar los resultados de ambas tareas para obtener la suma total

RESUMEN

En resumen, en un proceso paralelo utilizamos múltiples hilos para realizar tareas simultáneamente, mientras que en un proceso concurrente las tareas se ejecutan alternativamente y pueden compartir recursos. Ambos enfoques tienen sus propias ventajas y desventajas dependiendo de la situación y los recursos disponibles.

1.8 LINK: <https://github.com/LUISMIGUELMACEDOPINTO/PRO>