Estructura de Datos y Algoritmos 2

Semestre 2023-1

Profesor: Jorge Alberto Solano Galvez

Práctica 11. Introducción a OpenMP

Grupo 06

Integrantes:
Aguilar Martinez Erick Yair

Objetivo.

Conocer y aprender a utilizar algunas de las directivas de OpenMP utilizadas para realizar programas paralelos.

Actividades.

• Revisar el ambiente necesario para trabajar con OpenMP y lenguaje C.

• Realizar ejemplos del funcionamiento de las primeras directivas de OpenMP en lenguaje C.

• Realizar un primer programa que resuelva un problema de forma paralela utilizando las primeras directivas de OpenMP.

Instrucciones:

Ejercicio 1

Dado el siguiente programa que resuelve la integral definida:

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

Ejercicio 2

Para obtener el tiempo de ejecución OpenMP tiene la función

double omp_get_wtime()

Modifica el ejercicio 1 de esta práctica para medir el tiempo de ejecución del programa serial.

Ejercicio 3

Utilizando los siguientes elementos de OpenMP:

Constructor	Función	Clausula
#pragma omp parallel	omp_get_thread_num()	
	omp_get_wtime()	

Realiza una versión paralela del programa que permite calcular la integral definida del ejercicio 1.

Ejercicio 4

Modifica el ejercicio 3 de esta práctica para medir el tiempo de ejecución del programa paralelo.

Ejercicio 5

Dado el siguiente programa que utiliza el constructor **barrier**:

Implementar las funciones necesarias para que el programa ejecute correctamente y muestre la funcionalidad del constructor.

Ejercicio 6

Dado el siguiente programa que utiliza el constructor **critical**:

Implementar las funciones necesarias para que el programa ejecute correctamente y muestre la funcionalidad del constructor.

Ejercicio 7

Dado el siguiente programa que utiliza el constructor **atomic**:

Implementar las funciones necesarias para que el programa ejecute correctamente y muestre la funcionalidad del constructor.

Resultados obtenidos.

Ejercicio 1.

```
[edaII06alu01@samba practica_11]$ gcc -fopenmp ejercicio_1.c
[edaII06alu01@samba practica_11]$ ./a.out
Pi = 3.141593
```

Ejercicio 2

```
[edaII06alu01@samba practica_11]$ ./a.out
Time = 0.029087
Pi = 3.141593
[edaII06alu01@samba practica_11]$
```

Ejercicio 3.

```
[edaII06alu01@samba practica_11]$ gcc -fopenmp ejercicio_3.c
[edaII06alu01@samba practica_11]$ ./a.out
Time: 0.029210 Pi:3.141593
[edaII06alu01@samba practica_11]$
```

Ejercicio 4.

```
[edaII06alu01@samba practica_11]$ gcc -fopenmp ejercicio_4.c
[edaII06alu01@samba practica_11]$ ./a.out
Time:0.003867
Pi:3.141594
```

Ejercicio 5.

```
[edaII06alu01@samba practica_11]$ gcc -fopenmp ejercicio_5.c
[edaII06alu01@samba practica_11]$ ./a.out
A:239 B:915000007
A:430 B:760000007
A:610 B:535000007
A:783 B:240000007
[edaII06alu01@samba practica_11]$
```

Ejercicio 6.

```
[edaII06alu01@samba practica_11]$ gcc -fopenmp ejercicio_6.c
[edaII06alu01@samba practica_11]$ ./a.out
res = 62.814960
```

Ejercicio 7.

```
[edaII06alu01@samba practica_11]$ gcc -fopenmp ejercicio_7.c
[edaII06alu01@samba practica_11]$ ./a.out
A:26256.000000
[edaII06alu01@samba practica_11]$
```

Conclusiones.

Paralelizar es una técnica que nos permite reducir los tiempos de ejecución que tendrá nuestro programa sin embargo, no siempre resulta ser así porque existen dos principales limitantes:

1) No todo se puede paralelizar y aun cuando se pudiera lo complejidad a nivel lógico aumenta porque pensar de manera secuencial es más sencillo que de manera paralela.
2) Aumentar los hilos de ejecución no siempre resulta en una mejora sustancial,esto se nota más cuando el cómputo es pequeño.

Si vamos a paralelizar un algoritmo es porque resulta ser nuestro último recurso y hemos determinado que no hay una forma eficiente de disminuir la complejidad de un algoritmo.