Informde de laboratorio #4 – Pthreads

Alumno : Carpo Rodriguez David

Hw y Sw :

Para las pruebas ; se tuvo en cuenta un procesador Intel® Core™ i5-3230M CPU @ 2.60GHz × 4 , Memoria Ram de 8Gb DDR3. Tambien el sistema operativo Ubuntu 16.04 x64

- El codgo utilizado se encuentra en este repositorio: https://github.com/davidcrc/Paralelos

1. **Implementar el problema Matriz-Vector usando Ptthreads.**

   En la figura podemos apreciar los resultados de una prueba realizada con 4 threads de ejecucion y una matriz de $8x10^8$ x 8 , la cual nos arroja buenos resultados de tiempo.

```
david@T-C45:~/Escritorio/tarea3$ ./m 4 8000000 8

Se crearan 4 hilos
limite =  4, m = 8000000, n = 8
Thread 3 > tiempo transcurrido = 1.302080e-01 seconds
Thread 0 > tiempo transcurrido = 1.317239e-01 seconds
Thread 1 > tiempo transcurrido = 1.300080e-01 seconds
Thread 2 > tiempo transcurrido = 1.276379e-01 seconds
david@T-C45:~/Escritorio/tarea3$
```

Fig1 . Multiplicacion de matrices con pthreads

2. **Implementar el problema del Calculo de PI**

   - - Calculo de Pi : En la figura podemos apreciar los resultados de una prueba realizada con 2 threads y 10000 pasos para hallar el calculo de Pi en la formulada dada, , la cual nos arroja buenos resultados de tiempo. Con 10,000 pasos podemos obtener el mismo resultado calculado por threads y calculado de forma serial

```
david@T-C45:~/Escritorio/tarea3$ gcc -o m pi_pthread.c -lpthread
david@T-C45:~/Escritorio/tarea3$ ./m 2 10000

Se crearan 2 hilos
Con n = 10000 terminos,
Valor estimado-thr pi = 3.141492653590034
Valor estimado serial = 3.141492653590034
                   pi = 3.141592653589793

     Tiempo serial  = 1.120567e-04
```

Fig 2. Calculo de Pi

3. **Realizar pruebas y cambios de la tabla 4.1 del libro (Busy Waiting y Mutex)**

- Tiempo de ejecucin para hallar *Pi* usando 10^8 trminos : En la tabla 1 podemos apreciar que las pruebas nos indican que el calculo de Pi con *Mutex* tiene mejor rendimiento, mientras se utilice una mayor cantidad de hilos, dejando mas claro que *busy-waiting* tiene el problema de quedarse esperando a que un hilo se deje de utilizar el recurso para que pueda seguir trabajando.

Threads	Busy-Wait	Mutex
1	5.16E-01	5.20E-01
2	3.01E-01	3.07E-01
4	2.89E-01	2.88E-01
8	2.82E-01	2.96E-01
16	3.06E-01	2.52E-01
32	4.68E-01	1.62E-01
64	7.72E-01	4.08E-02

Tabla 1. Tiempos de ejecucion con Busy y Mutex

4. **Implementar la lista enlazada multithreading y replicar las tablas 4.3 y 4.4(lista enlaza)**

5. **Realizar experimentos y replicar los cuadros 4.5 (matrix-vector)**

En la tabla 2 podemos apreciar que la eficiencia al calcular una matrix de 8x10^8 x 8 aun es un poco alta , a comparacion de las otras dos , en la multiplicacion de 8000x8000 vemos que la eficiencia se mantiene en un promedio de las otras dos y en la ultima multiplicacion se ve claramente que la eficiencia mejora debido a que no presenta los problemas de la primera multiplicacion, donde tenemos un cache miss mas frecuente

Matrix Dimension						
	8, 000, 000 × 8		**8000 × 8000**		**8 × 8, 000, 000**	
Threads	**Time**	**Eff.**	**Time**	**Eff.**	**Time**	**Eff.**
1	3.32E-01	1.00	2.96E-01	1.00	2.52E-01	1.00
2	2.34E-01	0.71	2.25E-01	0.66	2.42E-01	0.52
4	1.27E-01	0.66	1.25E-01	0.59	1.30E-01	0.48

Tabla 2. Tienpos de ejecucion Multiplicacion de matrices con pthreads

6. **Implementar el problema presentado en la sección 4.11 del uso de strtok.**

Podemos ver como la tokenizacion es correcta, al utilizar la funcion *strtok_r()*, la cual tiene un argumento *&saveptr* que se utiliza para almacenar el estado de una llamada en lugar de utilizar una variable global

```
david@T-C45:~/Escritorio/tarea3$ ./m 2
Texto varias veces :
prueba de token uno
Thread 0 - texto = prueba de token uno
Thread 0 > string 1 = prueba
Thread 0 > string 2 = de
Thread 0 > string 3 = token
Thread 0 > string 4 = uno
pruena token dos
Thread 1 - texto = pruena token dos
Thread 1 > string 1 = pruena
Thread 1 > string 2 = token
Thread 1 > string 3 = dos
tokenizando tres
Thread 0 - texto = tokenizando tres
Thread 0 > string 1 = tokenizando
Thread 0 > string 2 = tres
```

Fig 3. Prueba utilizacion token_r con pthreads