Informe práctica 3

Paralelización función SAXPY iterativa

Autores: KEVIN ALONSO RESTREPO GARCÍA , CARLOS ANDRES GÓMEZ MARTÍNEZ

1. Introducción

→ El código a paralelizar es un programa que implementa la función Saxpy, cuyo objetivo es: "Para dos vectores ***x*** y ***y*** de tamaño ***n*** y un valor escalar ***alpha***, se calcula la siguiente sumatoria escalar de vectores: ***y=y+alpha x***".

→ Para el desarrollo de este laboratorio utilizamos C como lenguaje base para los objetivos del curso, y como editor de texto usamos Visual Studio Code. En el caso de realizar gráficas, hicimos uso de Python, en especial de su librería matplotlib haciendo uso de Colab que es un servicio cloud, basado en los Notebooks de Jupyter.

2. Método de Paralelización

En primer lugar, para hacer efectiva la paralelización se siguieron los pasos que el profesor nos dio durante las clases de laboratorio, teniendo así, una noción básica pero muy completa de como crear los hilos y paralelizar la función *SAXPY*. Ahora, tomando como base los adelantos del profesor, mantuvimos una estructura (*figura 1*) para pasar los parámetros a cada uno de los hilos creados.

```
typedef struct _param{
   int init;
   int end;
   int it;
   int p;
}param_t;
```

figura 1: Parameter Structure

Para paralelizar la función utilizamos dos vectores, uno para guardar los n hilos que se crean, y otro vector para almacenar los parámetros de cada uno de los n hilos creados. Para asegurar que al momento de paralelizar, se tuviera la certeza de que todos los hilos inicien a la misma vez cada

una de las iteraciones, utilizamos un ciclo externo que nos va a controlar cada una de las iteraciones y así garantizar que no existan hilos que se puedan adelantar y en esa ejecución se generen resultados diferentes a los esperados; dentro de este ciclo para controlar las iteraciones, creamos otro ciclo para la creación de los n hilos, dentro del cual utilizamos una variable que contiene la cantidad de posiciones a procesar por cada hilo, para asignar valores a cada de uno de los parámetros de cada hilo. Una vez se tiene la estructura creada para el i-ésimo hilo, procedemos a crearlo y pasarle la función *compute* para que la ejecute.

En la función *compute* en primer lugar obtenemos los datos de la estructura correspondiente al hilo en cuestión, una vez hecho esto, utilizamos otro ciclo para realizar la operación *Saxpy* en el rango de datos asignado, a la vez, en una variable *acum* vamos guardando la suma de los resultados; y posteriormente cuando se hayan culminado todas las operaciones, calculamos el promedio de la iteración actual, este cálculo se considera como una región crítica, ya que puede generar un *Race Condition*; por ello, hacemos uso de un semáforo binario *mutex* para protegerla y así, evitar resultados incorrectos.

3. Resultados

Comando lscpu en el equipo que se realizaron las ejecuciones:

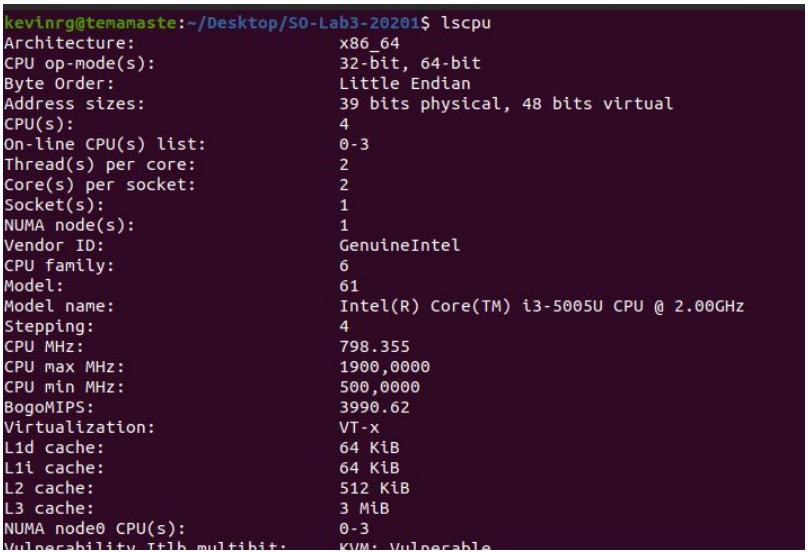

Figura 2: Características del equipo

Para la experimentación realizamos la ejecución de los casos de 1,2,4 y 8 hilos, un máximo de iteraciones de 50,100 y 1000, el tamaño del vector se mantuvo por defecto en 10000000 y la semilla también se mantuvo por defecto en 1. Para cada caso se realizaron repeticiones para tener un resultado promediado de los tiempos de ejecución y determinar la posible mejora en el rendimiento, debido a la paralelización de la función *Saxpy*.

A continuación, se muestran los resultados para cada caso de la experimentación:

Iteraciones	Hilos	Exec_Tavg(ms)	SpeedUp
50	1	3121,8757	1
50	2	1583,6008	1,971377951
50	4	1512,4728	2,064087169
50	8	1573,8251	1,983623021

Tabla 1: Resultados con 50 iteraciones

Iteraciones	Hilos	Exec_Tavg(ms)	SpeedUp
100	1	6258,2187	1
100	2	3146,5644	1,988905328
100	4	3024,8643	2,068925439
100	8	3108,7935	2,013069926

Tabla 1: Resultados con 100 iteraciones

Iteraciones	Hilos	Exec_Tavg(ms)	SpeedUp
1000	1	62281,4656	1
1000	2	31430,1165	1,981585579
1000	4	30144,3462	2,066107694
1000	8	31438,0485	1,981085614

Tabla 1: Resultados con 1000 iteraciones

Para analizar de mejor manera los resultados, se realizó una gráfica donde se visualiza la variación del tiempo de ejecución respecto al número de hilos.

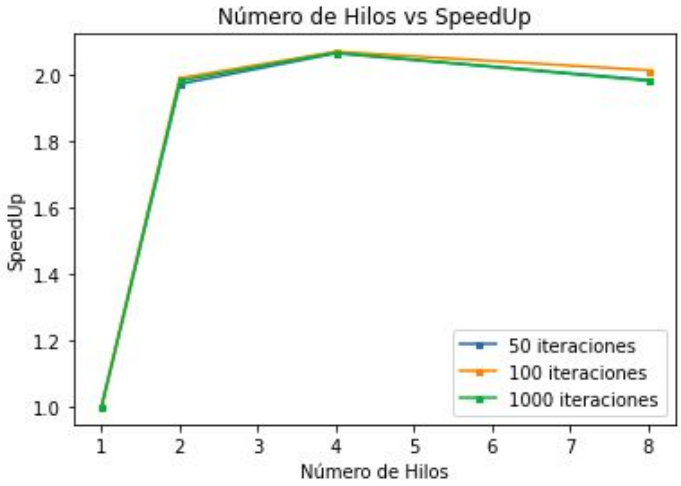

Figura 3: Número de hilos vs SpeedUp (Iteraciones)

La Figura 3 presenta un perfil de speedup para la paralelización de la función *Saxpy*, mostrando cómo cambia el SpeedUp dependiendo del número de hilos y del máximo de iteraciones ejecutadas.

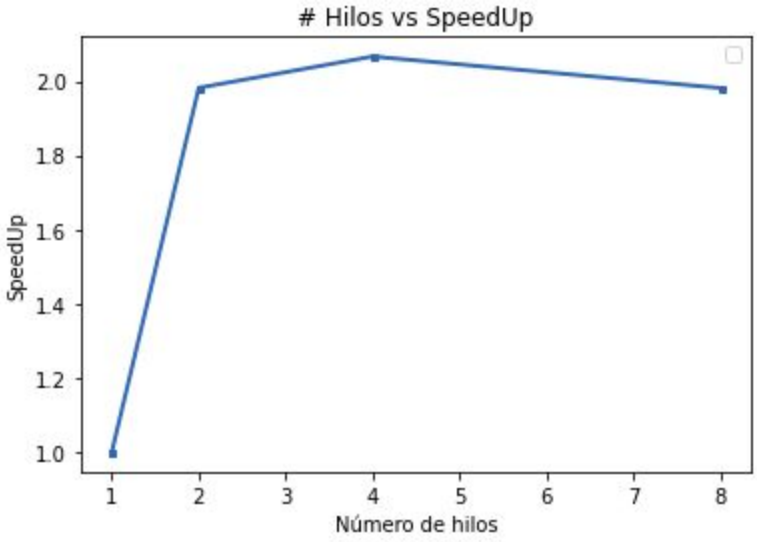

Figura 4: Número de hilos vs SpeedUp (1000 iteraciones)

En la figura 4 vemos los valores de SpeedUp mayores a 1 para cada número de hilos creados, lo que indica una mejora en el rendimiento lo cual es razonable ya que la carga de trabajo es alta y el proceso de división de carga y toda la previa necesaria para la paralelización, es más 'barata' que la ejecución secuencial.

4. Conclusiones

Para terminar, las actividades desarrolladas en este informe, fueron realizadas para explicar la manera en la que afrontamos el reto de paralelizar la función Saxpy y el agregado de calcular los promedios, de manera efectiva. Obteniendo resultados esperados, como lo es que el efecto de paralelizar aumenta la eficiencia de los procesos en comparación a una ejecución secuencial, y más cuando es una carga alta de trabajo (más iteraciones).

SI bien, en los resultados se puede apreciar que el SpeedUp obtuvo una mejora, al realizar la experimentación en un equipo con características regulares, no es tan notoria la diferencia.

El trabajo realizado, código, gráficas, informe se encuentra alojado en el repositorio de GitHub:

5. Referencias bibliográficas

Arpaci, R., & Arpaci Dusseau, A. (2015). Operating Systems: Three Easy Pieces (English Edition) (1.a edition) Arpaci-Dusseau Books. Recuperado de: http://pages.cs.wisc.edu/~remzi/OSTEP/

https://github.com/dannymrock/SO-Lab3-20201