Taller 02

Taller de Sistemas Operativos
Escuela de Ingeniería Informática

Jorge Rodríguez Antiquera

Jorge.rodrigueza@alumnos.uv.cl

Resumen. *En este informe se presenta la descripcion del problema, los datos y variables a utilizar, y el diseño de la solucion, en este caso se uso un diagrama de componentes para tener una mejor vista de la implementacion a desarrollar. Donde las tareas que se crearan deben realizarse de forma paralela e implementados con thread POSIX.*

1. Introducción

La computacion paralela [1] es donde se usan diversos recursos del computador para resolver un problema, la diferencia que se tiene con la computacion secuencial, es que las operaciones pueden ocurrir de forma simultanea para resolver el problema. Las ventajas de utilizar el paralelismo es resolver problemas en un tiempo acotado, en comparacion con la programacion secuencial. La ejecución del código es más rápida. Los resultados se obtienen en un tiempo menor. Permite la ejecucion de varias intrucciones en simultaneo.

En la programacion paralela se utilizan threads o hilos[2], esto quiere decir que al tener una cantidad de hilos especificos se pueden realizar varias tareas, ya que cada hilo estara a cargo de una tarea en especifico. Al tener procesos con diversas tareas independientes estas pueden terminar antes si estas tareas se encuentran en hilos diferentes, por lo que ahorraria tiempo de ejecucion. La creacion de hilo requiere menos recurso que un proceso y el tiempo de crearla es menor.

Los threads POSIX[3], es una modelo de ejecucion que no depende del lenguaje, sino que es un modelo de ejecucion para el paralelismo. En un programa se encuentran diversos controles de flujo de trabajo, donde en cada flujo de trabajo se encuentra en thread. La utilizacion de POSIX principalmente es por ser un estandar que facilita la creacion de apliaciones por su portabilidad y confiabilidad.

En el presente documento se podra observar la descripcion del problema, los modulos que se deberan en crear y que deben encontrarse en forma paralela para el desarrollo de la implementacion, de los datos y variables que se utilizaran al crear el código. Finalmente, se podra observar un diagrama de componentes que da la vista de la implementeacion, desde el llenado, las sumas parciales, los threads, hasta mostrar los resultados y el desempeño de estos.

2. Descripción del problema

El problema consiste en implementar un programa que conste con dos modulos. Uno que llene un arreglo con numeros enteros aleatorios de tipo uint32_t en forma paralela y otro que sume el contenido del arreglo en forma paralela. Se deben reaizar pruebas de desempeño que generen datos que permitan visualizar el comportamiento del tiempo de ejecución de ambos modulos dependiendo del tamaño del problema y de la cantidad de threads utilizados.

Los datos a utilizar seran uint32_t, esto quiere decir que se utilizaran numeros enteros sin signo que seran generados de forma aleatorias. Las variables a utilizar seran los parametros, estos son "**-N**" que tiene el tamaño de arreglo que el usuario ingresara, "**-t**" este es el numero de thread a utilizar, "**-l**" y "**-L**" son los limitires inferior y superior del rango aleatorio

3. Diseño de la solución

En la figura 1 se puede observar el diseño de la solucion con el cual se realizara la implementacion para la solucion del problema que fue mencionado previamente. Se visualiza una vista de procesos [4] que muestra la perspectiva que tendra el programador para la implementacion.

La metodologia que se utilizara es crear un modulo de llenado que por medio de los parametros ingresados del usuario se podra determinar el rango de los posibles valores que tomaran los numeros aleatorios que seran ingresados al arreglo, luego se realizara una suma serial y calcular su tiempo, luego al llegar a la seccion de los thread estos separan el trabajo de la suma en sumas parciales dependiendo de la cantidad de thread que hay en el parametro asignado, se sincronizan y se suman los resultados de las sumas parciales para finalmete calcular su tiempo. En la ultima parte se muestran los resultados con su eficiencia respectiva.

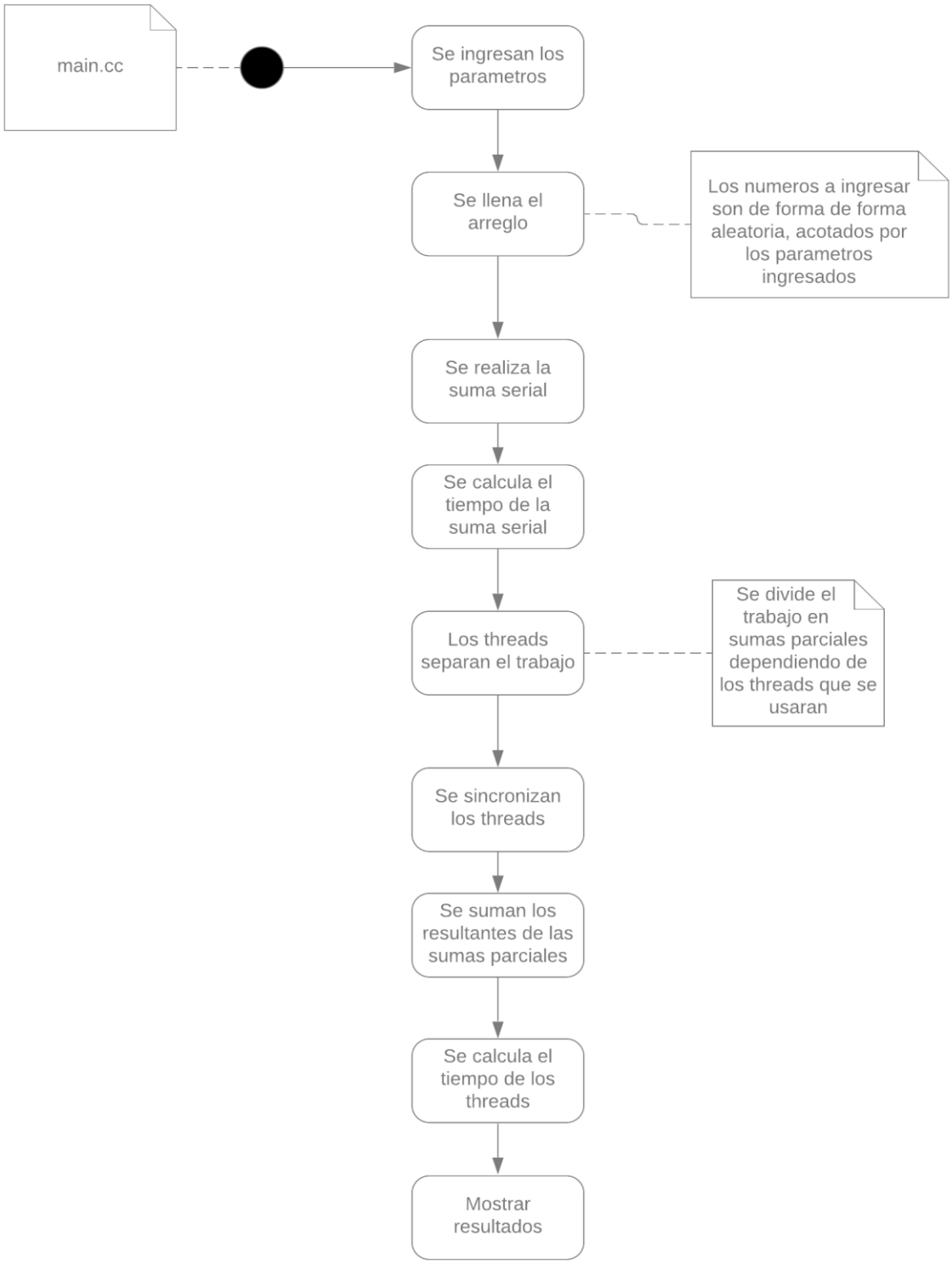

Figura 1. Diagrama de actividades.

4. Resultados

A continuación, se presenta la ejecucion del codigo como se muestra en la Figura 2, donde se visualiza los parametros que él usuario debe ingresar.

```
jorge@jorge:~/TSSO-taller02$ ./sumArray
Uso: ./sumArray -N <nro> -t <nro> -l <nro> -L <nro> [-h] Descripción:
        -N   Tamaño del arreglo
        -t   Cantidad de threads a utilizar
        -l   Limite inferior rango aleatorio
        -L   Límite superior rango aleatorio
        [-h]   Muestra esta ayuda y termina

jorge@jorge:~/TSSO-taller02$
```

Figura 2

En la figura 3 se muestra una prueba realizada con un arreglo de 10000000 posiciones, 4 hilos y los numeros enteros estan en el rango de [10,50].

```
jorge@jorge:~/TSSO-taller02$  ./sumArray -N 10000000 -t 4 -l 10 -L 50
Tamaño del arreglo: 10000000
Threads  : 4
Limit inferior: 10
Limit superior : 50
====Serial====
sumaSerial  : 299980321
Tiempo total serial   :4

====Threads====
sumaThreads : 299980321
Tiempo total threads :3

SpeedUp    :1.33333
Eficiencia :0.333333
```

Figura 3

En la Figura 4 se muestra la ejecucion del programa en un solo core, pero se logra visualizar una mejora en los tiempos de speedUp y Eficiencia con la Figura 3.

```
jorge@jorge:~/TSSO-taller02$ taskset -c 0 ./sumArray -N 10000000 -t 4 -l 10 -L 50
Tamaño del arreglo: 10000000
Threads  : 4
Limit inferior: 10
Limit superior : 50
====Serial====
sumaSerial  : 300031150
Tiempo total serial   :4

====Threads====
sumaThreads : 300031150
Tiempo total threads :5

SpeedUp    :0.8
Eficiencia :0.2
```

Figura 4

En la Figura 5 se muestra la ejecucion del programa con dos cores, doonde el SpeedUp es 1.33333 por lo que hay una ejecucion paralela. .

```
jorge@jorge:~/TSSO-taller02$ taskset -c 0,1 ./sumArray -N 10000000 -t 4 -l 50 -L 500
Tamaño del arreglo: 10000000
Threads  : 4
Limit inferior: 50
Limit superior : 500
====Serial====
sumaSerial  : 2749643645
Tiempo total serial   :4

====Threads====
sumaThreads : 2749643645
Tiempo total threads :3

SpeedUp    :1.33333
Eficiencia :0.333333
```

Figura 5

En la figura 6 se muestra la ejecución del programa utilizando el core 0, pero con hyper-threading. Se observa una mejora en SpeedUp y en Eficiencia.

```
jorge@jorge:~/TSSO-taller02$ taskset -c 0,2 ./sumArray -N 10000000 -t 4 -l 10 -L 50
Tamaño del arreglo: 10000000
Threads  : 4
Limit inferior: 10
Limit superior : 50
====Serial====
sumaSerial  : 300020592
Tiempo total serial   :4

====Threads====
sumaThreads : 300020592
Tiempo total threads :4

SpeedUp    :1
Eficiencia :0.25
```

Figura 6

5. Conclusiones

Se puede concluir que el diseño de alto nivel que se muestra en el documento permite visualizar la estructura que tendra la implementacion para satisfacer los requerimientos del problema, de como se debe trabajar en el. Tambien la importancia que tiene la utilizacion de los threads para hacer programacion paralela, en este caso para el llenado del arreglo y la suma de estos por medio de los diversos hilos que el usuario ingresará la cantidad de estos. Con los resultados mostrados se puede contemplar que una mejora en la Eficiencia, SpeedUp y en los tiempos al utilizar Threads con dos cores o usar Hyper-threading.

Referencias

[1] Github, http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html.

[2] Wikipedia, https://es.wikipedia.org/wiki/Hilo_(inform%C3%A1tica)#Funcionalidad_de_los_hilos.

[3] Utsm, http://profesores.elo.utfsm.cl/~agv/elo330/2s08/lectures/POSIX_Threads.html.

[4] Lucidchart, https://www.lucidchart.com/pages/es/tutorial-diagrama-de-actividades-uml.