Threads y OpenMp

Taller de Sistemas Operativos
Escuela de Ingeniería Informática

Raúl Valenzuela Cinto

Raul.valenzuela@alumnos.uv.cl

Resumen. *Este informe tecnico consta de la creación de un programa con la Api OpenMP, el cual consta de dos modulos, el primero deberá crear y rellenar un arreglo con números randomicos en forma paralela y el segundo modulo deberá tomar este arreglo y sumarlo de forma paralela, además se deberá realizar pruebas de desempeño para visualizar el desempeño del programa.*

1 Introducción

Los Threads (hilos) son considerados una agrupación en un programa junto un conjunto de registros del procesador que utilizan además de una pila de maquina, estos Threads son simpliemente tareas que se pueden ejecutarse al mismo tiempo que otra tarea, si estos se ejecutan al mismo tiempo y comparten los mismos recursos se pueden llamar un proceso, este proceso seguirá funcionando ya hayan muchos threads o al menos uno siga activo en ejecución.

Debido a esto el paralelismo que constan en varios calculos que pueden realizarse simulteaneamente esto ayuda a dividir problemas grandes para obtener varios pequeños, que luego serán solucionados en paralelo. Existen 2 tipos de paralelismo: Single program y Multiple Data las cuales son tecnicas para empleadas para que se logre el paralelismo.

OpenMP es una interfaz de programación (API) la cual sirve para programación que contenga mutiprocesos, esta se basa en en el modelo fork-join, donde tareas que son muy pesadas se dividen en N hilos con menor peso, para después obtener ese resultado y unirlos en uno solo. Este soporta paralelismo de tareas, lo que servirá para este taller en si.

Para completar este taller se utilizarán todo lo antes mencionado además de utilizar un lenguaje de programación C++ que permite agrupación de instrucciónes y la separación de un programa en modulos que admiten una compilación independiente, permitiendo así cumplir con el objetivo del taller el cual consistirá en realizar un programa con dos modulos, uno que llene un arreglo de numeros enteros aleatorios de forma paralela y otro en el que sume los numeros del arreglo de forma paralela.Además se tiene que tener en cuenta que para la realización de pruebas de desempeño lo mejor es comparar cosas, por lo que en este caso también se hará los mismos pasos anteriores pero de forma secuencial para luego compararlos entre si con la forma paralela y así obtener mejores resultados de desempeño.

2 Descripción

Se deberá implementar un programa que este compuesto por 2 modulos, uno en el cual se llenará un arreglo de números aleatorios enteros de tipo uint32_t de forma paralela, el otro modulo tendrá que sumar el contenido del arreglo de forma paralela.

Además se deberán hacer pruebas de desempeño las cuales generen datos que permitan visualizar el comportamiento del tiempo de ejecución en ambos modulos, todo dependiendo del tamaño del problema y la cantidad de threads utilizados.

Al terminar los modulos la forma de ejecutar el programa se dará a cabo de la siguiente Tabla1.

./sumArray -N <nro> -t <nro> -l <nro> -L <nro> [-h]

Tabla1 Forma de uso.

Y sus parametros estan explicados en la Tabla2.

-N : tamaño del arreglo.
-t : número de threads.
-l : limite inferior rango aleatorio.
-L : límite superior rango aleatorio.
[-h] : muestra la ayuda de uso y termina.

Tabla 2 Parametros.

En la Tabla 3 Se puede vizualizar un ejemplo de utilización de los datos.

./sumArray -N 1000000 -t 4 -l 10 -L 50

Tabla 3 Ejemplo de utilización.

3 Diseño de la solución

Primero que nada para entender el problema a resolver, se debe tener en cuenta que este consiste OpenMp para la realización de un codigo en c++, el cual contendrá dos modulos,antes de empezar con los modulos se deberá tener en cuenta que el codigo tiene que solicitar ciertos argumentos de entrada, para definir el tamaño del arreglo,el numero de hilos a crear y el rango de los numeros aleatorios sindo definidos por un limitite inferior y uno superior, luego se creará un arreglo de tamaño dinamico, este será llenado con numeros aleatorios de forma paralela, luego tendremos el modulo de suma, que consistirá en tomar los datos guardados y sumarlos de forma paralela,todo esto implementado con la API de OpenMP, siendo guardados en una variable para hacer pruebas de desempeño proximamente.

En la *Figura 1* se puede analizar la forma del modulo de etapa de llenado en la cual se llena el arreglo con numeros random usando OpenMP.

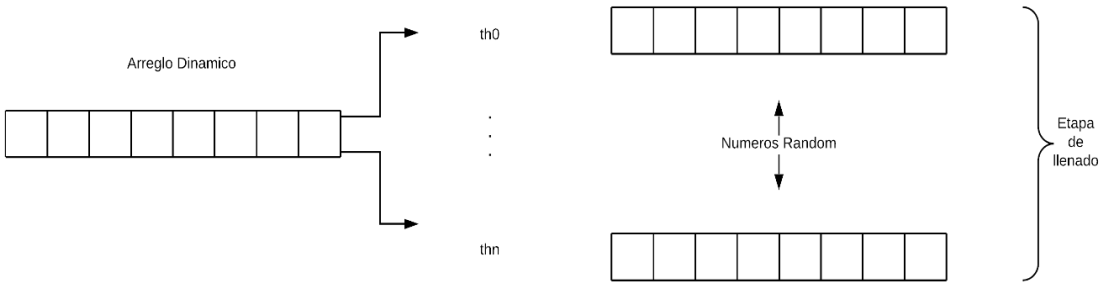

Figura 1:Modulo de llenado

En la *Figura 2* se puede ver la forma que toma los datos de suma con OpenMP y este gracias a su forma fork-join suma todo y lo deja almacenado.

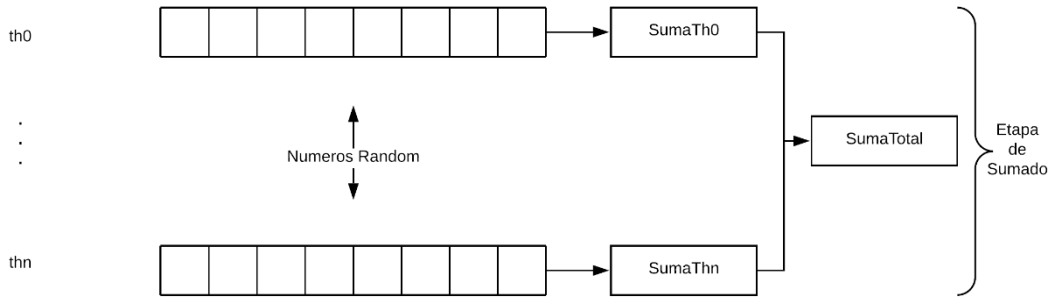

Figura 2: Modulo de sumado

4 Pruebas

Primeramente se tiene que ver si los parametros ingresados son correctos, si no lo son esto debe mostrar la forma de uso como lo muestra la *Figura 3*

```
ignaciov@tallerdesistemas:~/taller3/TSSOO-taller03$ ./sumArray -r
./sumArray: invalid option -- 'r'
Uso: ./sumArray -N <nro> -t <nro> -l <nro> -L <nro> [-h] Parametros:
        -N   tamaño del arreglo.
        -t   número de threads
        -l   limite inferior rango aleatorio
        -L   limite superior rango aleatorio
        -h   muestra esta ayuda y termina

ignaciov@tallerdesistemas:~/taller3/TSSOO-taller03$
```

Figura 3: Muestra forma de uso.

Como se puede ver en la *Figura 4* tanto como en la *Figura 5* y *Figura 6* se muestra la forma de ejecución del codigo con distintos parametros, además de las distintas muestras de resultados que serán analizados más adelante, ya que se harán mas pruebas de las que se muestran a continuación para una mejor visualización en resultados.

```
ignaciov@tallerdesistemas:~/taller3/TSSOO-taller03$ ./sumArray -N 10000000 -t 2 -l 1 -L 1000
Elementos: 10000000
Threads  : 2
limite inferior: 1
limite Superior: 1000
==========Suma Total================
Suma en Secuencial: 5003165297
Suma en Paralelo: 5003165297
Suma en Omp: 5003165297
==========Tiempos de Llenado=========
TiempodeLLenadoSecuencial: 316.868[ms]
TiempodeLLenadoParalelo: 155.766[ms]
TiempodeLLenadoOpenMp: 227.153[ms]
SpeedUp Etapa de Llenado Thr vs Serial: 2.03426
SpeedUp Etapa de Llenado Omp vs Serial : 1.39496
==========Tiempos de Sumado==========
TiempoDeSumadoSecuencial: 6.70256[ms]
TiempoDeSumadoParalelo: 7.33914[ms]
TiempoDeSumadoOpenMp: 5.40809[ms]
SpeedUp Etapa de Suma Thr vs Serial: 0.913262
SpeedUp Etapa de Suma Omp vs Serial: 1.23936
ignaciov@tallerdesistemas:~/taller3/TSSOO-taller03$
```

Figura 4:Modo de uso

```
ignaciov@tallerdesistemas:~/taller3/TSSOO-taller03$ ./sumArray -N 10000000 -t 4 -l 1 -L 5000
Elementos: 10000000
Threads  : 4
limite inferior: 1
limite Superior: 5000
==========Suma Total================
Suma en Secuencial: 25002327777
Suma en Paralelo: 25002327777
Suma en Omp: 25002327777
==========Tiempos de Llenado=========
TiempodeLLenadoSecuencial: 341.324[ms]
TiempodeLLenadoParalelo: 157.942[ms]
TiempodeLLenadoOpenMp: 205.191[ms]
SpeedUp Etapa de Llenado Thr vs Serial: 2.16107
SpeedUp Etapa de Llenado Omp vs Serial : 1.66345
==========Tiempos de Sumado==========
TiempoDeSumadoSecuencial: 7.07423[ms]
TiempoDeSumadoParalelo: 5.59092[ms]
TiempoDeSumadoOpenMp: 5.23184[ms]
SpeedUp Etapa de Suma Thr vs Serial: 1.26531
SpeedUp Etapa de Suma Omp vs Serial: 1.35215
ignaciov@tallerdesistemas:~/taller3/TSSOO-taller03$
```

Figura 5: Modo de uso

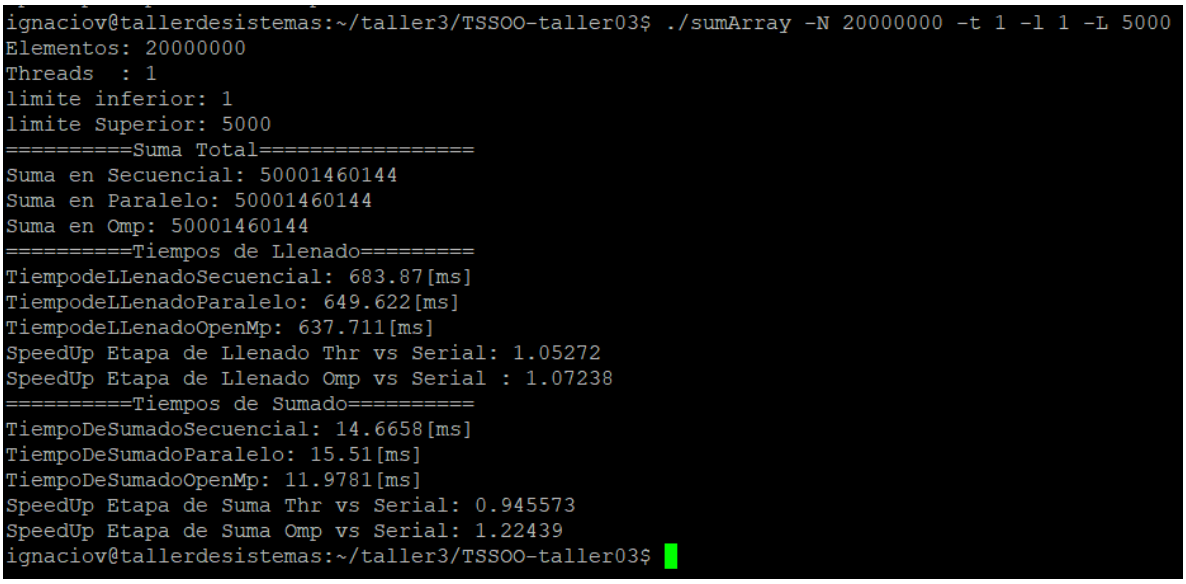

Figura 6: Modo de uso

5 Resultados

Luego de ciertas pruebas se pudo analizar que para el mismo numero de elementos y limites inferior, superior, al aumentar el numero de threads se puede ver la que mejora la eficencía del programa hasta cierto punto.

Forma de uso para las pruebas : ./sumArray -N 10000000 -t 1 -l 1 -L 5000 cambiando el numero de threads para ver el cambio, además no se hizo con datos muy grandes ya que al hacerlo en el sistema invitado los recursos son más limitados y el tiempo de ejecución y el de suma se veían afectados.

Como se puede ver en la *Figura 7* la comparación de el tiempo de llenado entre OpenMP y Threads para ver cual de estos es más eficiente para el trabajo.

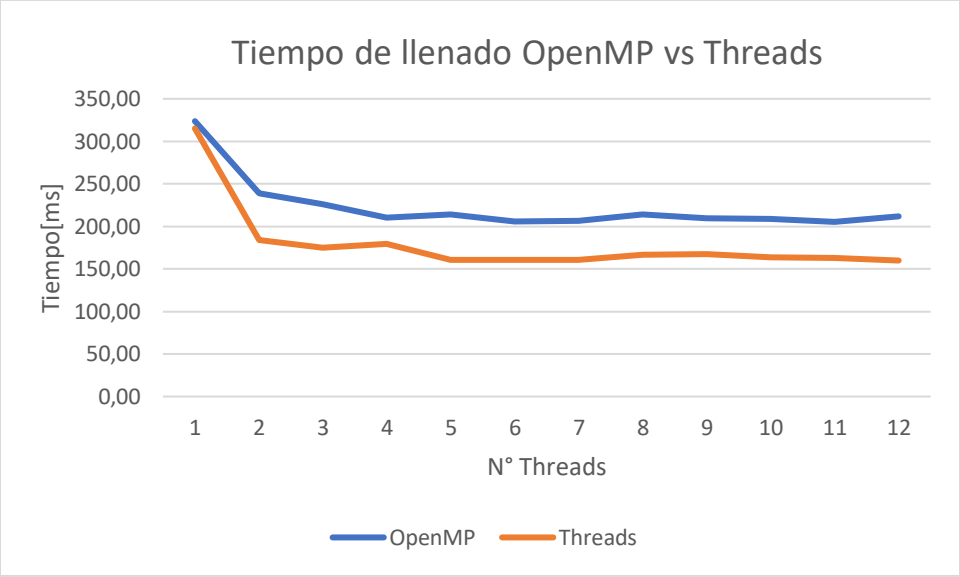

Figura 7:Gráfico de comparación llenado OpenMP vs Threads

Analizando los resultados en el grafico se puede evidenciar que para el llenado de la solución, el tiempo que demora paralelamente, es más eficiente los threads siempre que sea hasta 4 threads ya que después el cambio es minimo y OpenMP se ve por debajo en este caso.

En la *Figura 8* se puede vizualizar la comparativa entre los tiempos de suma que se obtienen sumando con OpenMP vs la forma con Threads.

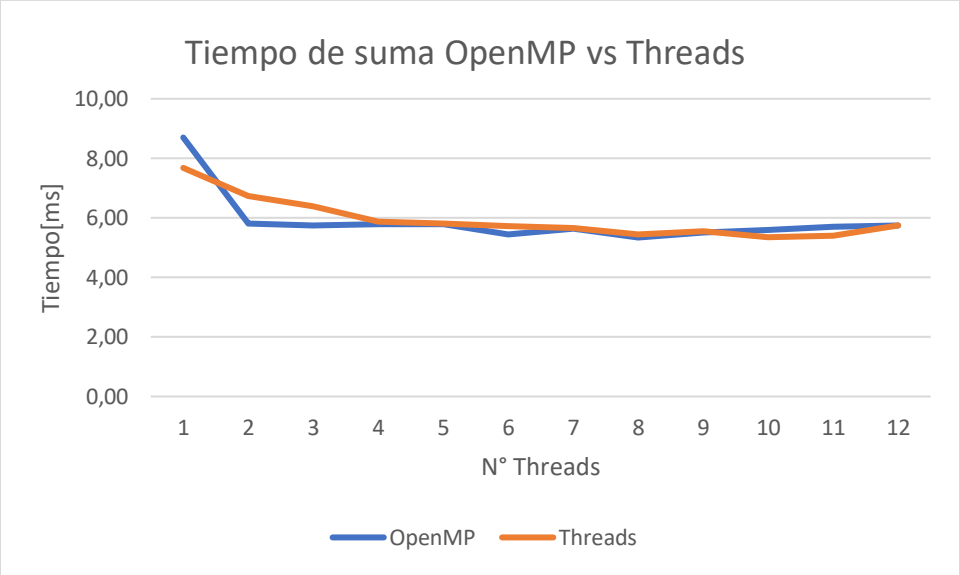

Figura 8:Gráfico de comparación suma OpenMP vs Threads

Analizando los resultados que se entregan en el gráfico se puede notar que en ambos casos son formas eficientes de sumar hasta los 3-4 threads ya que después no se vizualiza una eficiencia mayor si no que se mantiene.

6 Conclusión

Al termino de este taller se cumple con el objetivo de realizar los dos modulos de llenado y suma paralelamente implementados con OpenMp, y al haber analizado los resultados, se puede concluir que en este caso los threads hasta cierto punto pueden llegar a ser más efectivos de forma paralela respecto al OpenMp, pero no necesariamente en todos los casos ya que el OpenMp puede ser mejor para otros tipos de soluciones y ser mejores que los threads, por lo que para el taller en si solo se requiere usar entre 2-4 threads para una eficiencia satisfactoria ya que si se usan más no es notorio el cambio. Además se pudo comprender mejor que tanto el uso de threads como de OpenMp pueden llegar a ser igual de efectivos.