Taller de Sistemas Operativos

Implementación y Análisis con OpenMP
"Taller 03"

Taller de Sistemas Operativos
Escuela de Ingeniería Informática
Cristian Valencia Avila
cristian.valenciaa@alumnos.uv.cl

***Resumen**: Se implementa un programa de dos módulos cuyo propósito es llenar un arreglo de números enteros para luego sumarlos, con la condición de que ambas tareas sean realizadas de forma paralela por medio de hilos(Threads-safe) cuyas tareas deben ser implementadas con OpenMP.*

1. Introducción

Ejecutar un programa que realice de forma paralela ciertas tareas, se puede lograr implementando un método llamado Threads, los cuales consisten en una unidad básica de ejecución de un SO. Cabe mencionar que cualquier programa el cual se ejecute consta de un thread mínimo.

En el Sistema Operativo los hilos son una secuencia de tareas que ocurren de manera simultánea y concurrente, para ser más preciso el SO reparte la CPU entre los threads que se ejecutan en todo momento, ya que un programa puede poseer varios threads, de manera que simplemente se "adueña" de la CPU y ejecuta el siguiente thread. Cada vez que ocurre esto el SO se encarga de guardar las características (registros y pila) que posee cada thread para que al momento de conmutar el mismo thread pueda restaurar su contexto inicial, esto nos da paso a llevar a cabo distintas funciones simultáneamente.

Se debe tener en cuenta que cada thread trabaja como si tuviese un microprocesador para el solo, es decir, se ejecuta de forma absolutamente independiente. Los threads que comparten recursos se conocen como proceso, esta característica posibilita que cualquiera de los thread pueda modificar dichos recursos, para lo cual se necesita algún tipo de sistema de sincronización. Estos procesos no terminan su ejecución mientras que uno de sus threads esté activo.

De manera general, un thread se define como una tarea que puede ser ejecutada al mismo momento que otra tarea. En la figura 1 se puede apreciar un thread en ejecución.

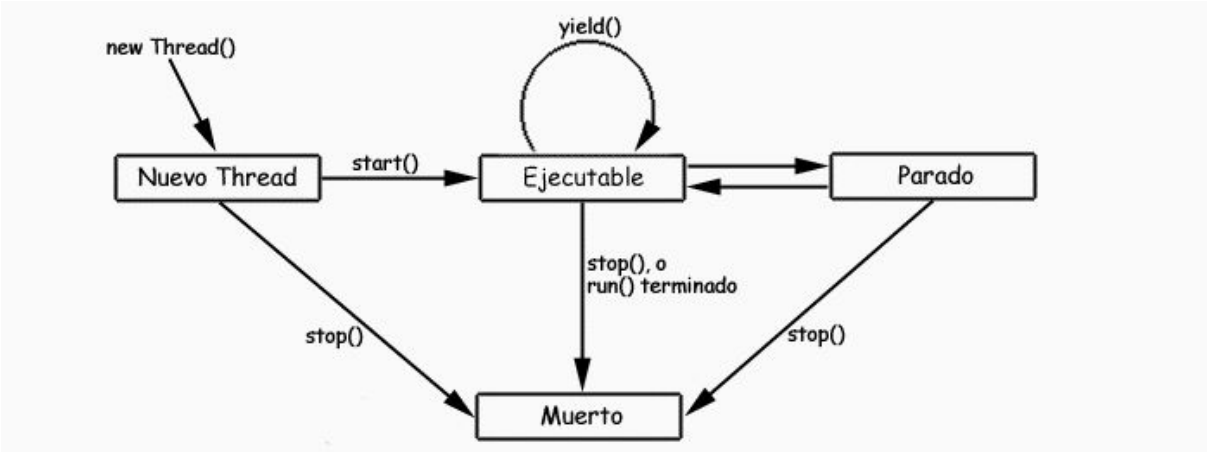


Figura 1: Esquema de un thread en ejecución.

En este taller la implementación se requiere el uso de OpenMP, el cual corresponde a una interfaz de programación de aplicaciones para la programación multiproceso de la memoria compartida, este también permite añadir concurrencia a programas que han sido implementadas en lenguaje C,C++. OpenMP está basado en un modelo fork-join. Esta implementación es requerida debido a que esta soporta el modelo de paralelismo de tareas.

Para el desarrollo de este taller se requiere utilizar el lenguaje de programación C++ 2014 o superior, para la creación de los thread bajo la implementación OpenMP, permitiendo efectuar de manera simultánea, en este caso, dos tareas explicadas más detalladamente en la siguiente sección.

2. Descripción del Problema

2.1 Planteamiento del Problema

Se solicita la implementación de un programa, el cual consta de dos módulos principales. Uno se encarga de llenar un arreglo y de forma paralela, el otro módulo se encarga de sumar el contenido del arreglo mencionado. En la Figura 2 se puede ver de manera general el proceso por el cual pasan ambos módulos.

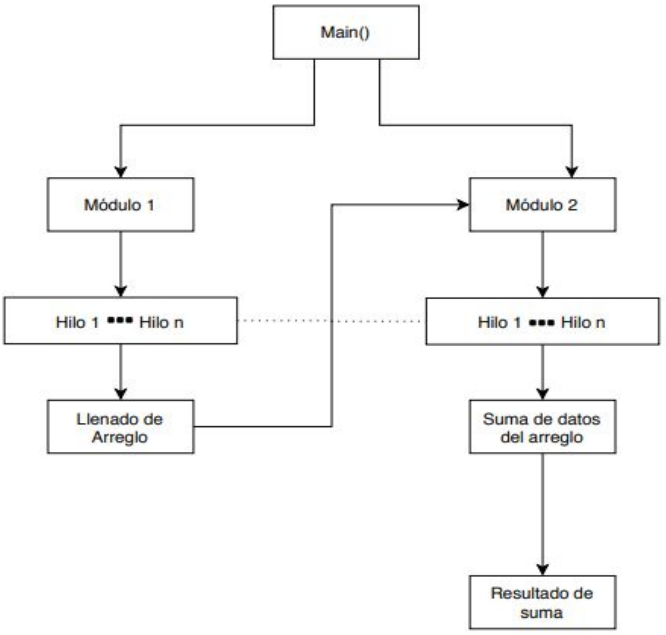


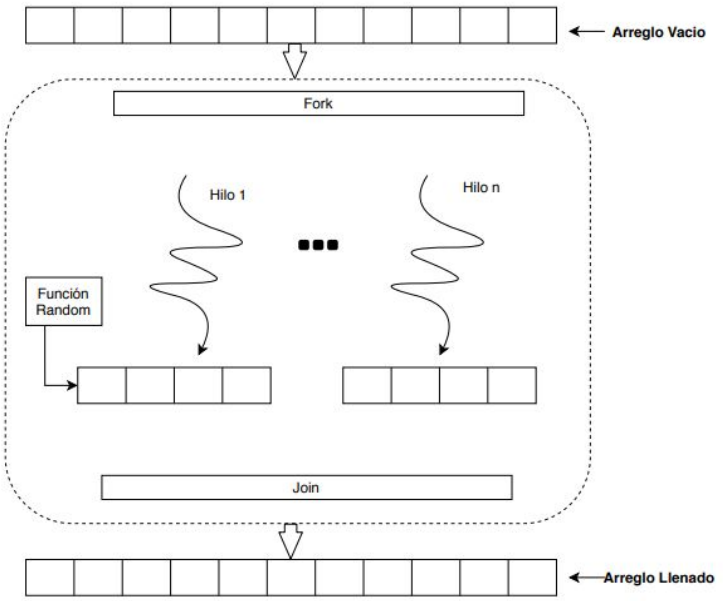
Figura 2: Vista general del problema

2.1.1 Módulo de llenado de arreglos

Este módulo se encarga de llenar los arreglos vacíos creados por el Main(), a través de alguna función que entregue números de forma randómica siendo del tipo `uint32_t`, donde cada hilo guardará los valores entregados por dicha función hasta el momento de realizar el llenado(Ver figura 3).

Como recomendación para generar los números aleatorios se requiere usar funciones que sean *thread safe* para observar una mejora de desempeño en el programa.

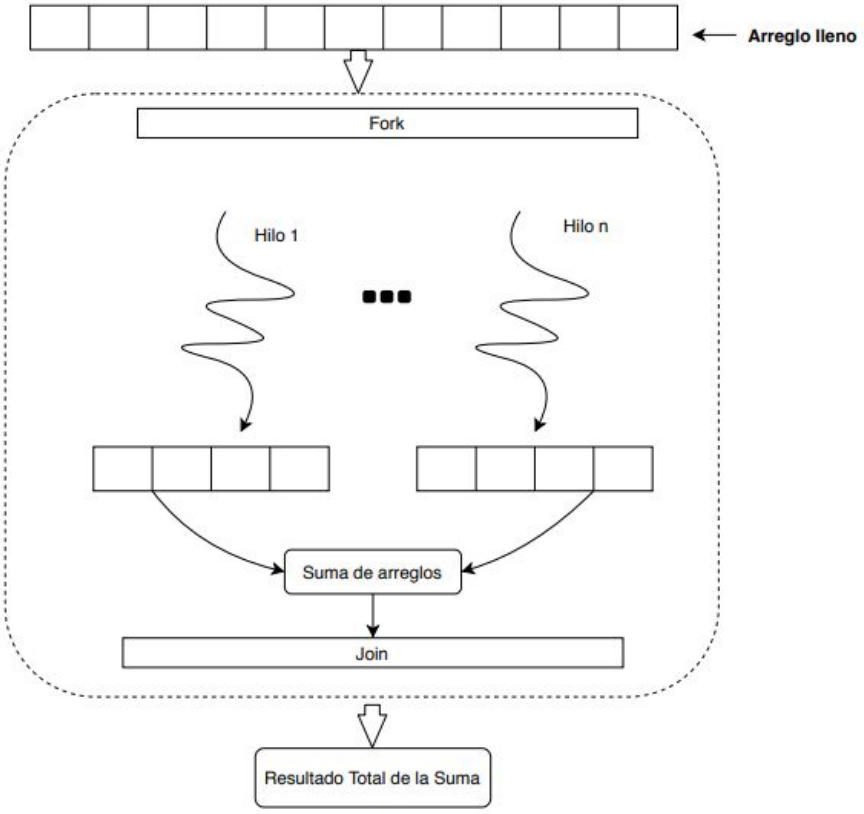
Figura 3: Esquema llenado de arreglo



2.1.2 Módulo de suma de arreglos

En este módulo se encarga de sumar los valores que contiene el arreglo llenado anteriormente, para el cual los hilos creados en el primer módulo realizan una división del arreglo para proseguir a realizar la suma de valores contenidos en el arreglo(ver Figura 4).

Figura 4: Esquema de Suma del arreglo.



2.2.1 Forma de uso

Para la ejecución del código se realiza de la siguiente manera:

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

Los parámetros utilizados se pueden ver en la tabla 1.

-N	Tamaño del arreglo.
-t	Número de Threads.

-l	Límite inferior rango aleatorio
-L	Lìmite superior rango aleatorio.
-h	Muestra la ayuda de uso.

Tabla 1: Descripción de los parámetros.

2.1.2 Ejemplo de la forma de uso

1) Creación de un arreglo de 100 posiciones, con 6 threads. Los números enteros aleatorios están en el rango [20,40], por lo que su línea de comando sería:

   ./ sumArray -N 100 -t 6 -l 20 -L 40

2) Muestra de la ayuda:

- **./sumArray -h**
- **./sumArray**

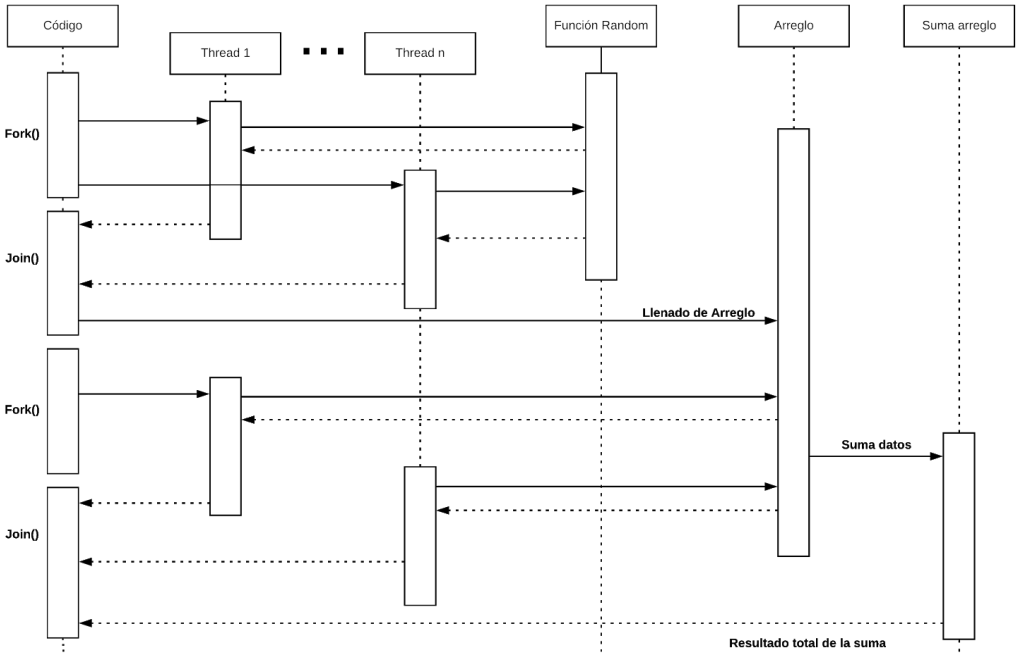
2.2 Tareas requeridas

Se requiere visualizar el comportamiento del tiempo de ejecución que toma cada módulo, esto dependiendo del tamaño de cada problema y de la cantidad de threads que se utilizaron, para esto se deben realizar pruebas de desempeño que generen cierto datos que nos permita esta visualización.

3. Diseño de Solución

El modelado de la solución se puede ver representado en la Figura 5, a través de un diagrama de secuencia se expresa el modelado de los procesos de ambos módulos, llenado y suma de los datos del arreglo.

Figura 5: Diagrama de Secuencia del proceso de los hilos.



4. Pruebas de Desempeño

Para comprender el comportamiento del código implementado y con ello visualizar el desempeño de este mismo, refiriéndose al tiempo en ejecución que lleva la realización de cada módulo. En primer lugar se evalúa el algoritmo implementado bajo distintas características para un arreglo de tamaño N, para esto se realiza una comparativa de los módulos con dos tipos de llenado: *Secuencial y Paralelo*, calculando su tiempo de ejecución y con ello su desempeño.

En primer lugar para comenzar se debe saber la forma de uso del código implementado con sus respectivos parámetros como se puede apreciar en la siguiente figura 6:

```
cristian@DESKTOP-VS0OV9G:/mnt/c/Users/Cristian/Desktop/taller03$ ./sumArray
Uso: ./sumArray ./sumArray -N <nro> -t <nro> -l <nro> -L <nro> [-h]
 Parámetros:
        -N   tamaño del arreglo.
        -t   número de threads.
        -l   limite inferior rango aleatorio.
        -L   limite superior rango aleatorio.
        -h   muestra la ayuda de uso y termina.
```

Figura 6: Forma de Uso.

4.1 Muestras del funcionamiento

Como se puede apreciar en las siguientes Figuras 7,8,9 y 10, son ejemplos de la ejecución del código con sus respectivos parámetros como se ve y explica en la forma de uso.

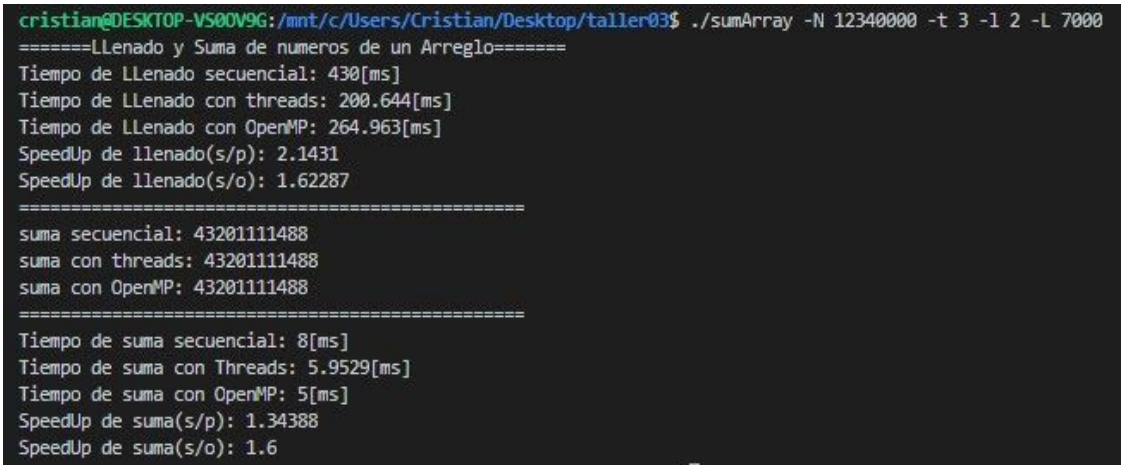


Figura 7: Ejemplo de una muestra de datos nº1

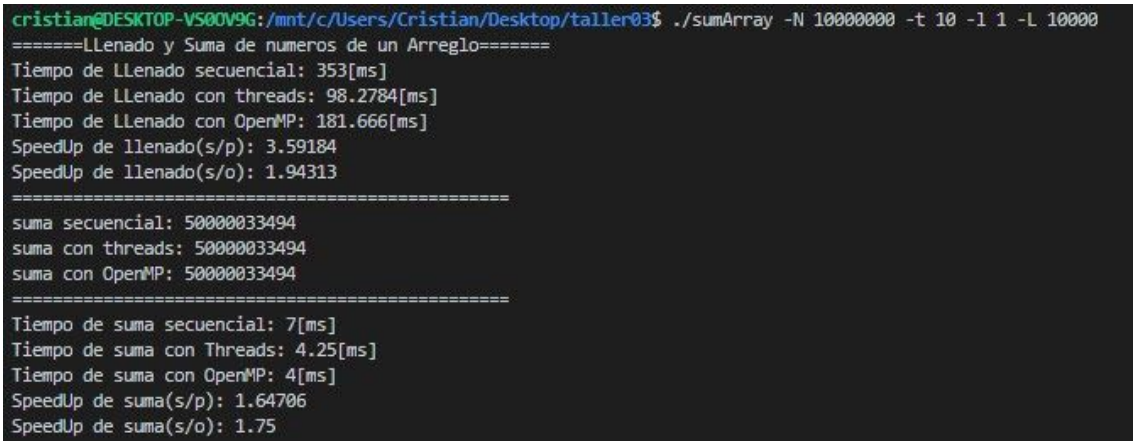


Figura 8: Ejemplo de una muestra de datos nº2

```
cristian@DESKTOP-VS0OV9G:/mnt/c/Users/Cristian/Desktop/taller03$ ./sumArray -N 46770000 -t 2 -l 10 -L 90000
=======LLenado y Suma de numeros de un Arreglo=======
Tiempo de LLenado secuencial: 1632[ms]
Tiempo de LLenado con threads: 935.861[ms]
Tiempo de LLenado con OpenMP: 1241.85[ms]
SpeedUp de llenado(s/p): 1.74385
SpeedUp de llenado(s/o): 1.31417
================================================
suma secuencial: 2105104459546
suma con threads: 2105104459546
suma con OpenMP: 2105104459546
================================================
Tiempo de suma secuencial: 34[ms]
Tiempo de suma con Threads: 21.4108[ms]
Tiempo de suma con OpenMP: 20[ms]
SpeedUp de suma(s/p): 1.58798
SpeedUp de suma(s/o): 1.7
```

Figura 9: Ejemplo de una muestra de datos nº3

```
cristian@DESKTOP-VS0OV9G:/mnt/c/Users/Cristian/Desktop/taller03$ ./sumArray -N 10000000 -t 1 -l 1 -L 10000
=======LLenado y Suma de numeros de un Arreglo=======
Tiempo de LLenado secuencial: 345[ms]
Tiempo de LLenado con threads: 343.281[ms]
Tiempo de LLenado con OpenMP: 335.609[ms]
SpeedUp de llenado(s/p): 1.00501
SpeedUp de llenado(s/o): 1.02798
================================================
suma secuencial: 50013199605
suma con threads: 50013199605
suma con OpenMP: 50013199605
================================================
Tiempo de suma secuencial: 8[ms]
Tiempo de suma con Threads: 7.3953[ms]
Tiempo de suma con OpenMP: 6[ms]
SpeedUp de suma(s/p): 1.08177
SpeedUp de suma(s/o): 1.33333
```

Figura 10: Ejemplo de una muestra de datos nº4

Como se pudo apreciar en la figuras mostradas anteriormente son ejemplos de la ejecución del código con distintos parámetros, además los datos que se pueden ver serán analizados más adelante en un serie de pruebas para comprender su comportamiento y desempeño.

5. Resultados

Como se mencionó anteriormente, después de trabajar y realizar un análisis de la serie de pruebas al código solicitado en este taller se logró obtener los siguientes resultados:

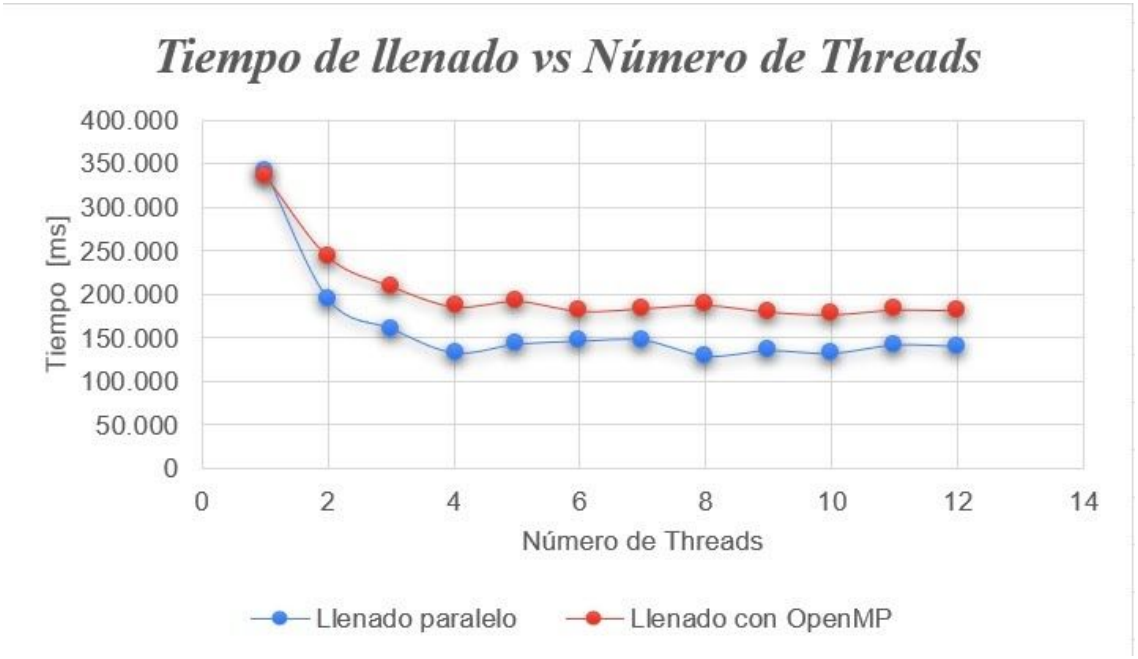


Figura 11: Gráfico de tiempo de llenado vs número de threads.

En la Figura 11 se pueden ver los datos que se han obtenido después de calcular el promedio de una serie de pruebas que se llevaron a cabo en la máquina virtual tanto como en el computador anfitrión, donde nos muestra la comparativa del tiempo de llenado con threads y con la implementación con OpenMP, como se aprecia claramente en la ejecución con threads en el proceso de llenado presenta un desempeño mejor a comparación de la implementación OpenMP, ya que a medida que aumenta la cantidad de hilos el tiempo del proceso disminuye más en el llenado con threads.

Cabe mencionar que los parámetros ocupados para estas pruebas son los siguientes:

- -N 10000000
- -t (Hilos que irán variando)
- -l 1
- -L 10000

Los parámetros mencionados se ocuparon tanto para el módulo de llenado como para el de suma.

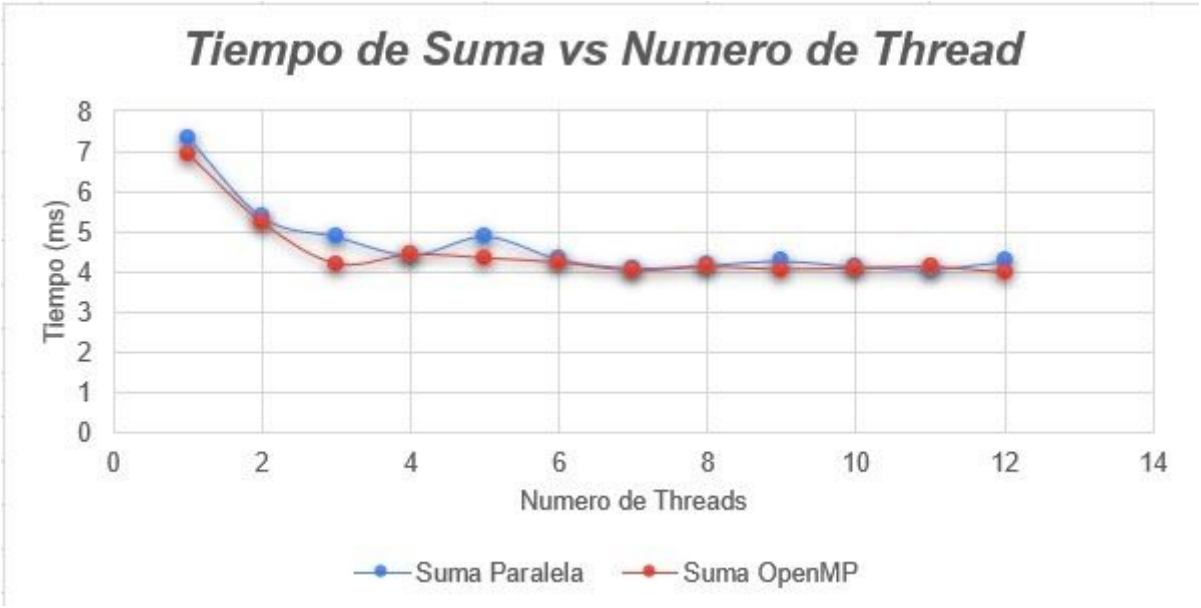


Figura 12: Gráfico de tiempo de suma vs número de threads.

En la figura 12 se pueden ver los datos que sean obtenido del mismo número de pruebas realizadas para el primer módulo, pero en este caso se realiza la comparativa de los tiempos de las sumas realizadas con threads y la suma realizada con la implementación de OpenMP.

Como se puede apreciar en el gráfico en un inicio las sumas para ambos casos es prácticamente la mismas, pero en a medida que va en aumento el número de hilos, hasta 3 aproximadamente en el caso con OpenMP, se logra ver un mayor desempeño por parte de este último, pero a medida que sigue en aumento el número de threads las curvas avanzan de manera “constante”, ya no existiendo una diferencia notable para cada caso.

Cabe destacar que cada valor obtenido puede variar dependiendo de los recursos que se otorguen para la ejecución, por esto mismo el motivo de la realización de una serie de pruebas, para lograr tener una vista lo más clara posible del desempeño explicado anteriormente.

6. Conclusión

Como se hizo referencia en un inicio y los objetivos que estaban propuestos para el taller presentado, en primer lugar se logró realizar de manera correcta la creación de ambos módulos, el llenado y la suma de datos de un arreglo dinámico, donde se logra visualizar el desempeño con respecto al tiempo tomado en la ejecución de cada módulo, además de comprender el funcionamiento y la utilidad que proporciona el uso de hilos, gracias a distintas pruebas llevadas a cabo.

Cabe destacar de lo que se logra percibir después de un número reiterado de pruebas que a medida que el número de datos aumenta diferencia entre el uso de threads y la implementación con OpenMP, deja de variar de manera notable, por lo que en ciertos casos el uso de dicha implementación puede ser "mejor" si se habla de desempeño, pero no siempre será este el caso.

Como comentario al realizar las pruebas en la máquina virtual y en el computador anfitrión, es que el rendimiento al momento de la ejecución en ambos casos los resultados varían debido a que no se poseen los mismos recursos, pero lo que no quiere decir una mejor implementación no pueda corregir o por lo menos reducirlo.