Reporte técnico: Taller 03

Taller de Sistemas Operativos

Escuela de Ingeniería Informática

Felipe Castro Aguilar

felipe.castroa@alumnos.uv.cl

Resumen-*Hoy en día el multiprocesamiento es una característica fundamental de la programación de sistemas, aunque existan procesos alternativos a la creación de hilos, estos presentan un gran número de ventajas en especial en los procesadores multi núcleo. Como objetivo se tiene la creación de un diseño de solución adecuado a la problemática planteada, especificando y analizando cada punto de este. . Es importante destacar que el objetivo de este taller es implementar un programa que llene un arreglo de números enteros y luego sume los datos. Cada tarea se debe realizar de forma paralela, implementadas con OpenMP. Con el objetivo de este taller en mente se generó un diseño, abarcando de manera general sin un detalle excesivo del código a implementar.*

1. Introducción

Para iniciar este taller debemos dominar y contextualizar las tecnologías que ocuparemos para el desarrollo de un diseño optimo que cumpla con la tarea de llenar un arreglo de números enteros y luego sumarlos. Ambas tareas se realizarán en forma paralela implementadas con OpenMP. Es importante destacar que el desarrollo de este reporte tiene la misma base y estructura que el informe Técnico de taller dos, ya que ambos trabajan y se desarrollan en el ámbito del paralelismo y tiempo de ejecución.

Para dar paso al diseño de solución debemos entender el contexto en el cual está sumergido este reporte, para ello analizaremos los siguientes puntos.

1.1. Paralelismo

Antes de entender que es multiprocesamiento, multithreading y thread debemos entender que paralelismo es una función que realiza el procesador para ejecutar varias tareas al mismo tiempo, en otras palabras, puede realizar varios cálculos simultáneamente, basado en el principio de dividir los problemas grandes para obtener varios problemas pequeños. Al hablar sobre paralelismo surge un concepto fundamental llamado concurrencia, el cual se refiere a la existencia de varias actividades ejecutándose simultáneamente y necesita sincronizarse para ejecutarse de manera conjunta.

Dentro del paralelismo existen una gran cantidad de términos, pero para este informe solo nos enfocaremos en los cuales son necesarios para entender el diseño que se presentará.

1.1.1. Multiprocesamiento

Como su nombre lo indica es un procesamiento el cual permite la ejecución de uno o más hilo de ejecución a la vez. Esto es debido a la existencia de múltiples CPU las cuales pueden ser utilizadas para ejecutar múltiples procesos o múltiple thread dentro de un mismo proceso.

Muchas son las ventajas las cuales pueden traer un multiprocesamiento, algunas son:

- Mayor productividad: Esto es debido a completarán mayor cantidad de tareas en un menor periodo de tiempo.
- Mayor confiabilidad: En la ausencia de un procesador el sistema se tornará más lento, pero no se caerá, ya que posee más de un CPU.
- Ahorro de dinero: Esto se apega al punto de mayor productividad ya que, si baja el tiempo para realizar la misma tarea, se ve reflejado en los insumos necesarios para el funcionamiento de la máquina.

1.1.2. Multithreading

Multithreading o multihilo es la capacidad de ejecutar eficientemente múltiples hilos de ejecución. Esta técnica junto con Multiprocesamiento, descrito en el punto anterior, son los principales métodos para mejorar el rendimiento de una máquina. Un punto fundamental de esta técnica es que se utiliza para dar solución a las instrucciones ejecutadas secuencialmente, ya que al crear más de un hilo de ejecución este puede realizar trabajos de manera paralela.

1.1.3. Thread

Un Thread o hilo, es la unidad básica de ejecución de un Sistema Operativo. Además, puede ser ejecutado al mismo tiempo que otra tarea. Todos los hilos de ejecución comparten el espacio de direccionamiento del proceso y permiten simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

Además, estos se diferencian de los procesos, ya que estos últimos son una instancia de ejecución del programa, en cambio los threads es la unidad de proceso más pequeña, otra diferencia es que los procesos se pueden dividir en varios hilos mientras que el hilo no se puede dividir.

Como ventaja los hilos tienen un tiempo de respuesta mejora notablemente, comparten recursos, por lo que se puede tener varios hilos de ejecución dentro del mismo espacio de direcciones. Otra ventaja es la utilización de múltiples CPU, esto permite que los hilos de un mismo proceso se ejecuten en diferentes CPU´s a la vez.

1.2. OpenMP

Luego de abarcar los temas de paralelismo, multiprocesamiento, multithreading y thread podemos continuar con el siguiente punto, el cual es el foco del desarrollo de este diseño y del código generado. OpenMP permite añadir

concurrencia a los programas escritos en C/C++. OpenMP se basa en el modelo fork-join, paradigma que proviene de los sistemas Unix, donde una tarea es dividida en ***k*** hilos(fork) para luego "reunir" sus resultados al final y unirlos en un solo resultado(join).

La API mencionada se divide en tres componentes, directivas de un compilador , rutinas de biblioteca y variables de entorno que influyan en el comportamiento en tiempo de ejecución.

Un punto fuerte de OpenMP es su modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas, como es el caso de este taller. Otro punto igual de importante es el procesamiento de memoria compartida en paralelo.

OpenMP es una API la cual nos permite añadir concurrencia a un código mediante paralelismo con memoria compartida. Su implementación se basa en la creación de hilos de ejecución paralelos compartiendo variables del proceso padre que los crea.

1.3. Problema

Se me solicitó crear un programa en el lenguaje de programación C++ 2014 o superior, el cual llene un arreglo de números enteros aleatorios del tipo uint32_t y luego los sume. El programa de be estar separado por dos módulos, el primero será el encargado de llenar un arreglo con números aleatorios en forma paralela y otro encargado de sumar el contenido de aquel arreglo en forma paralela, ambos módulos deben ser trabajados con OpenMP. Sumado a esto, se deben hacer pruebas de desempeño las cuales permitan visualizar el comportamiento del tiempo de ejecución de ambos módulos dependiendo del tamaño del problema la cantidad thread a utilizar.

El programa debe responder a una forma de uso (ver figura 1), estos parámetros se detallan en la siguiente tabla (ver tabla 1)

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

Figura 1 Forma de uso.

Tabla 1 Parámetros

PARÁMETROS	DESCRIPCIÓN
-N	Tamaño del arreglo.
-t	Número de thread.
-l	Límite inferior del rango aleatorio.
-L	Límite superior del rango aleatorio.
[-h]	Muestra la ayuda de uso y termina.

Como último requerimiento que presenta este taller fue la instalación de OpenMP en del equipo donde se desarrollará el código y en la maquina virtual a la cual se le implementará el código.

Para lograr abarcar todo el problema primero se realizó un correcto estudio del lenguaje de programación mencionado y seguido a esto profundizar en el buen uso de hilos.

2. Diseño

Luego de dar a conocer el marco teórico de este reporte técnico se procedió a desarrollar el diseño de la solución a implementar, para ello se enfoco en la solicitud inicial la cual nos pide crear un programa que este dividido en dos módulos los cuales cumplan funciones especificadas. Todo esto en el lenguaje de C++ con la implementación de OpenMP. A continuación, se muestra el diseño de cada módulo.

El diseño presentado se divide en dos partes donde la primera consta un desarrollo del taller dos, y la segunda del taller tres, el cual es el foco de este reporte técnico. El motivo de esta mención es la comparativa que se genera en una implementación a partir de thread creados y analizados en el taller dos en contra parte la utilización de una Api la cual se encarga de la creación de thread logrando paralelizar el código adjunto.

Por razones de utilidad el desarrollo del taller dos se presenta de manera resumida.

2.1. Taller 2

El taller dos presenta un problema similar al enfrentado en el taller numero tres donde se me solicitó crear un programa el cual llene un arreglo de números enteros aleatorios del tipo uint32_t y luego los sume. Ambas tareas separadas por módulos y cada tarea debe ser realizada en paralelo.

Para el primer modulo se realizó un diseño acorde a lo analizado y estudiado, generando el siguiente diagrama de secuencia.

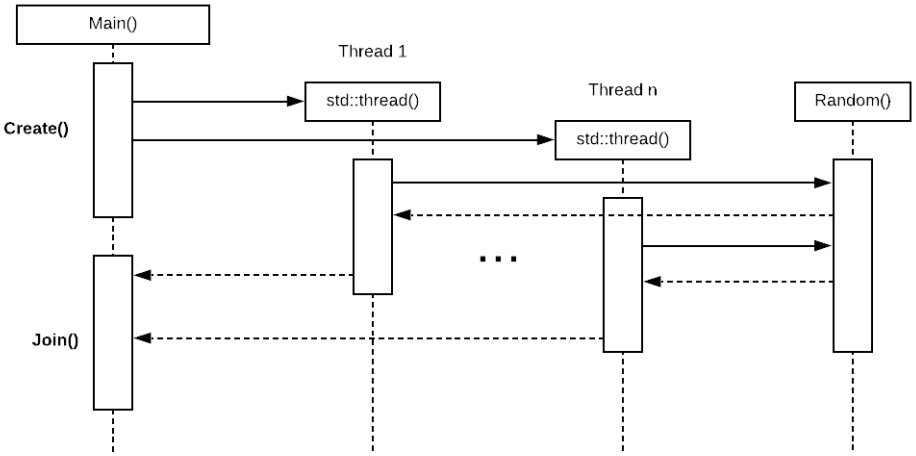

Figura 2 Diagrama de secuencia del proceso de llenado.

La Figura 2 muestra el flujo con el que opera la primera parte del módulo, donde cada thread creado mediante parámetro “dividirá” el arreglo en partes equitativas y cada uno de ellos se encarga de llenar el arreglo principal con los valores aleatorios correspondientes. El arreglo principal tendrá el tamaño de la cantidad de datos a crear y los valores aleatorios estarán acotados dependiendo de los parámetros a ingresar. Cada thread se encargará de llenar una sección del arreglo principal logrando con lo solicitado de llenar un arreglo en forma paralela.

Para la segunda parte del taller dos se encuentra la suma, donde nuevamente se crearán tantos hilos como en un inicio y cada uno de ellos “dividió” el arreglo inicial con los valores aleatorios y sumara cada una de su subdivisiones para finalmente sumar las sumas obtenidas por cada thread. La Figura 3 muestra lo anteriormente mencionado.

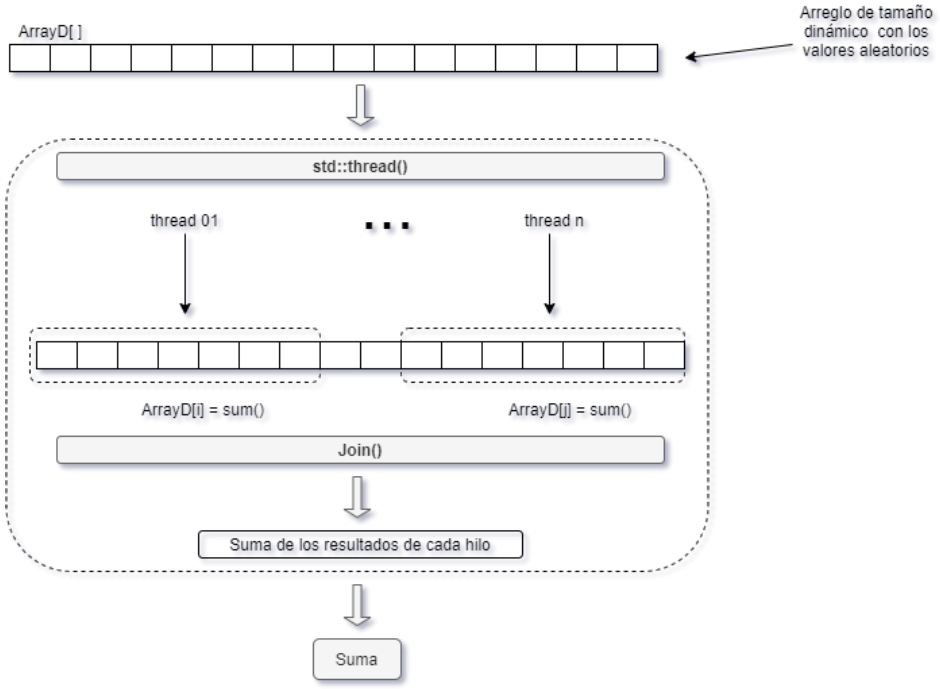

Figura 4 Diagrama de secuencia del proceso de suma.

La razón de porque se optó por entender y analizar el diseño del taller dos es la siguiente, de acuerdo con lo solicitado en el presente taller, debo realizar una implementación paralela en un código para realizar la misma tarea anteriormente planteada, pero con la diferencia de que en este caso se debe ocupar OpenMP. Para ello se debe analizar tanto el diseño como el desarrollo del taller dos para lograr trazar las directrices del diseño de solución para OpenMP.

Ahora bien, como el nuevo problema se divide en dos módulos, se segmentará la siguiente etapa de diseño de solución en dos etapas las cuales cada una representa un módulo.

2.2. Diseño modulo uno

Como primer módulo se nos solicita que generemos un arreglo de números enteros aleatorios del tipo uint32_t en forma paralela. Como ya se menciono en el punto 1.3 el arreglo debe ser llenado a través de parámetros de entrada los cuales ya fueron especificados, esto establece que cada thread que creemos conoce el inicio y fin de donde debe almacenar el numero aleatorio.

Ya que el OpenMP se basa en el modelo fork-join() el diseño implementado en el taller dos sirve como base para el diseño del módulo uno.

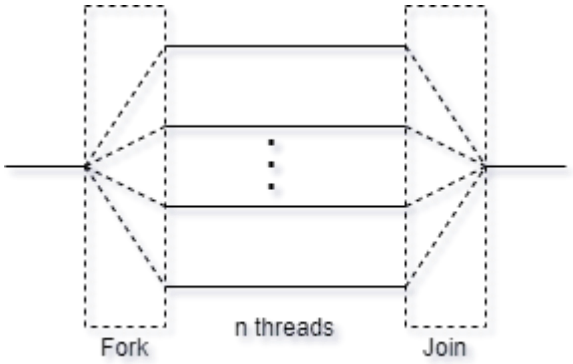

Figura 5 Modelo fork-join().

La figura 5 muestra el modelo fork-join() de la "n" cantidad de hilos a crear. Cada hilo estará encargado de "dividir" y luego llenar el arreglo con números aleatorios dentro de un rango .

Los números aleatorios se almacenaron en un arreglo de tamaño dinámico el cual se encargará de almacenar los números aleatorios. El arreglo dinámico tendrá un largo dependiendo del parámetro de entrada. Ya que se solicita que sea un llenado en paralelo se crearan una "n" cantidad de thread, la cual es entregada en la ejecución del programa (ver figura 1), los cuales trabajaran de manera paralela para llenar el arreglo a través de una función random().

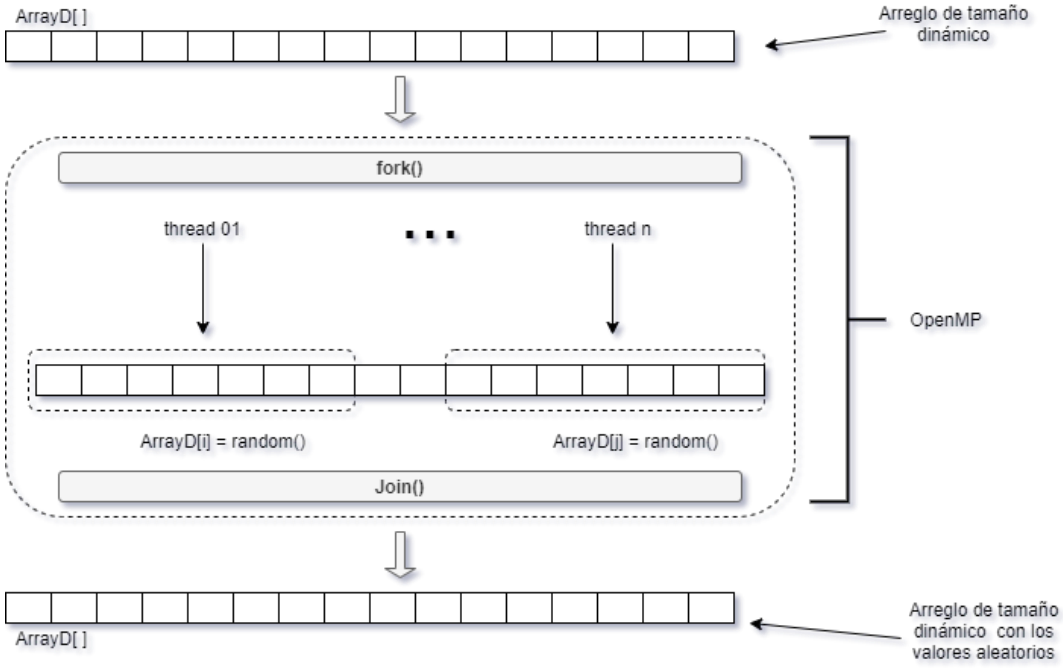

Figura 6 Diagrama general del módulo uno.

Como se muestra en la Figura 6 el diseño general del módulo uno se establece que a través del modelo frok-join() se creen n cantidad de threads para realizar el llenado de números aleatorios ya definidos. La API se encarga de paralelizar el código el cual esta asignado para llenar el arreglo con datos aleatorios obtenidos previamente.

2.3. Diseño modulo dos

Para este módulo se solicitó sumar los valores del arreglo obtenidos anteriormente mediante el correcto uso de OpenMP. Para esto se diseña un flujo similar al anterior donde a través de los parámetros de entrada se volvió a "dividir" el arreglo para que los threads se encarguen de sumar la "división" correspondiente y finalmente sumar dichas sumas, para obtener el resultado esperado.

En este caso se diferencia del caso del taller dos ya que no presenta un consolidado de datos fuera de la creación y aplicación de hilos. En este caso la suma se va acumulando gracias a una función de OpenMp, reduction() la cual se encarga de sumar todas las sumas de las sumas correspondientes a cada uno de los threads solicitados.

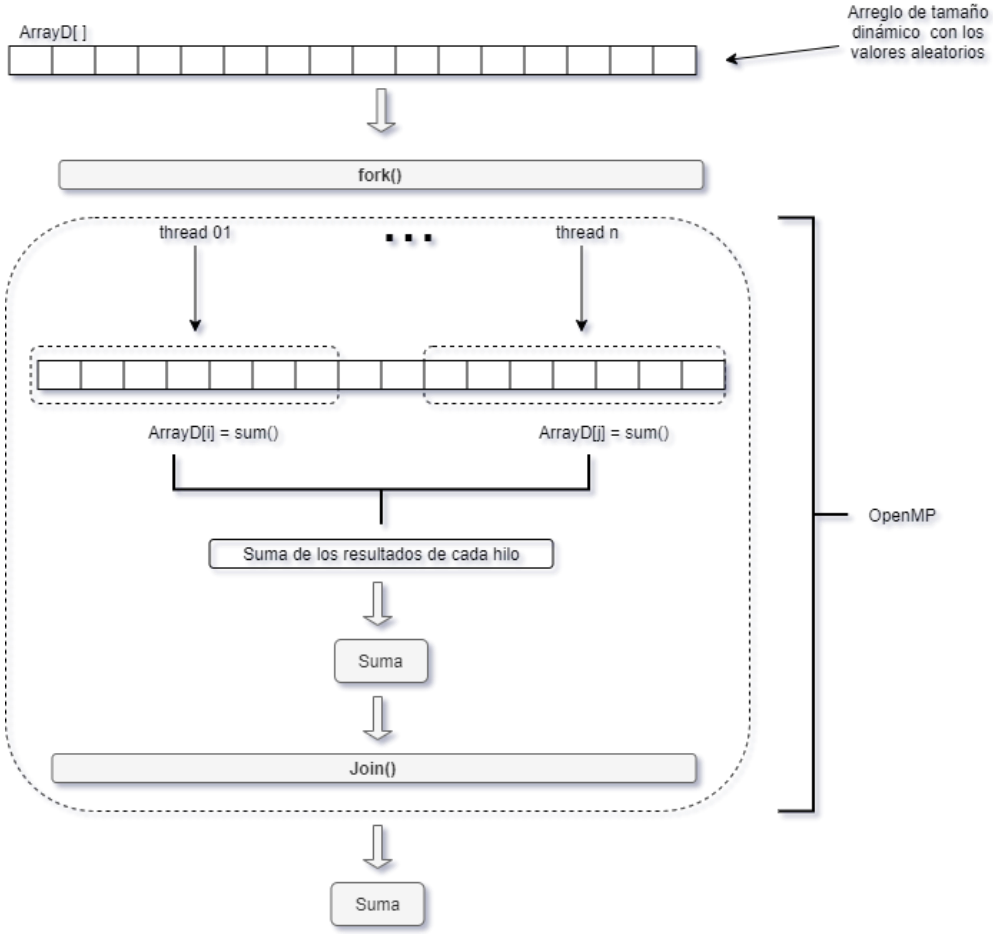

Figura 7 Diagrama general del módulo dos.

En la Figura 7 se muestra el diseño del proceso a realizar el cual entrega como salida el valor solicitado en este taller.

Es importante destacar que ambos módulos trabajan con OpenMP y que cada módulo coparte el mismo arreglo base para poder trabajar con los datos sin generar un incongruencia en los datos.

3. Resultados

Una vez analizado, comprendido, trabajado y desarrollado el código solicitado en este taller y a través de múltiples pruebas tanto en mi equipo y en la máquina virtual instalada generé los siguientes resultados:

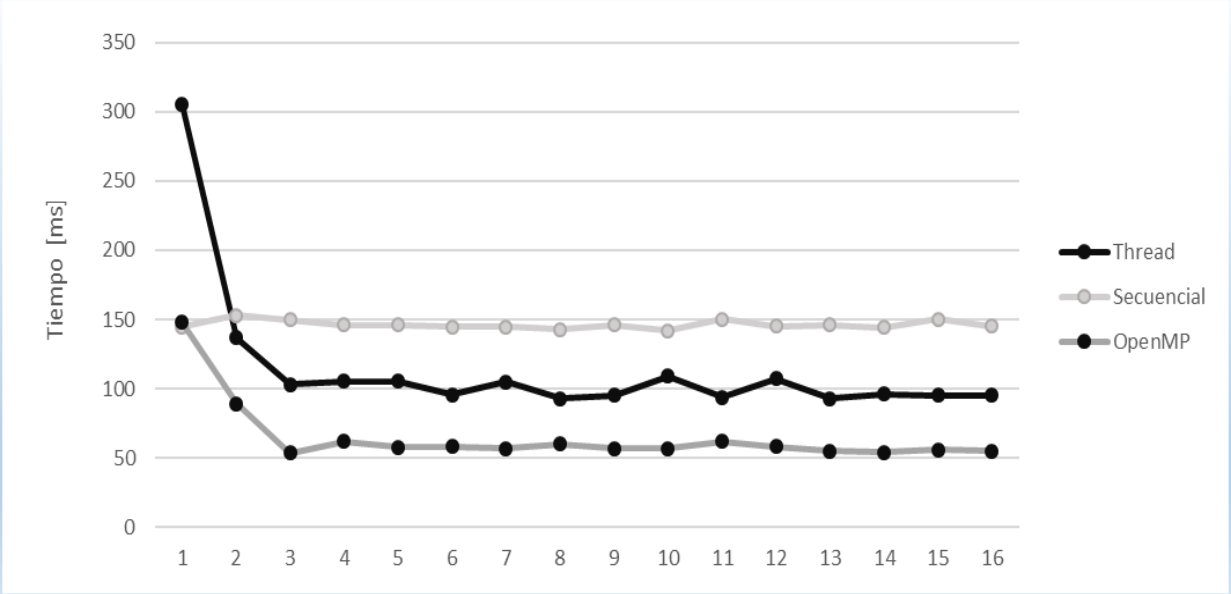

Figura 8 Grafico Tiempo de llenado vs cantidad de threards

La Figura 8 se muestra los datos obtenidos de las pruebas realizadas en la máquina virtual donde claramente la ejecución con OpenMP presente un mejor rendimiento a medida que la cantidad de thread a utilizar. Esto se explica, ya que al ocupar hilos estamos dividiendo el trabajo en partes "similares" y esto mejora los tiempos de ejecución.

La gráfica muestra una comparativa de los tiempos de llenado mediante hilos implementados por OpenMP, de manera secuencial y la utilización threads, los cuales fueron desarrollados para el taller dos. Para cada valor se hicieron 5 pruebas donde se promedió para obtener los datos presentados en la tabla. Es evidente que el método secuencial se mantenga "constante", ya que la cantidad de datos y el rango de ellos, para todas las pruebas fueron la misma.

También es importante entender que la curva de threads con OpenMP se aplana debido a la capacidad física del computador donde se implementa dicho código y la calidad del código realizado.

Además del caso de ejecución con OpenMP vs método secuencial se realizó otra prueba para el ámbito de llenado donde se compara la ejecución de la ejecución de threads realizada en el taller dos vs la ejecución con OpenMp.

En la Figura 8 se aprecia como la curva presentada en la ejecución con threads(taller dos) disminuye su tiempo drásticamente pero luego se mantiene "estable". En cambio, la ejecución con OpenMP no presenta una gran disminución de tiempo a medida que aumentan los hilos, pero si es mejor en términos de tiempo de ejecución en todo momento.

Como comentario para esta etapa puedo decir que el tiempo al ocupar un solo hilo y el tiempo secuencial deben ser similares, pero comparando el valor inicial de la ejecución secuencial con el valor inicial de la aplicación thread, se muestra con una diferencia de 150 [ms], esto es debido, luego de un análisis, al como se llenaron los arreglos correspondientes, ya que se ocuparon vectores y arreglos para este módulo.

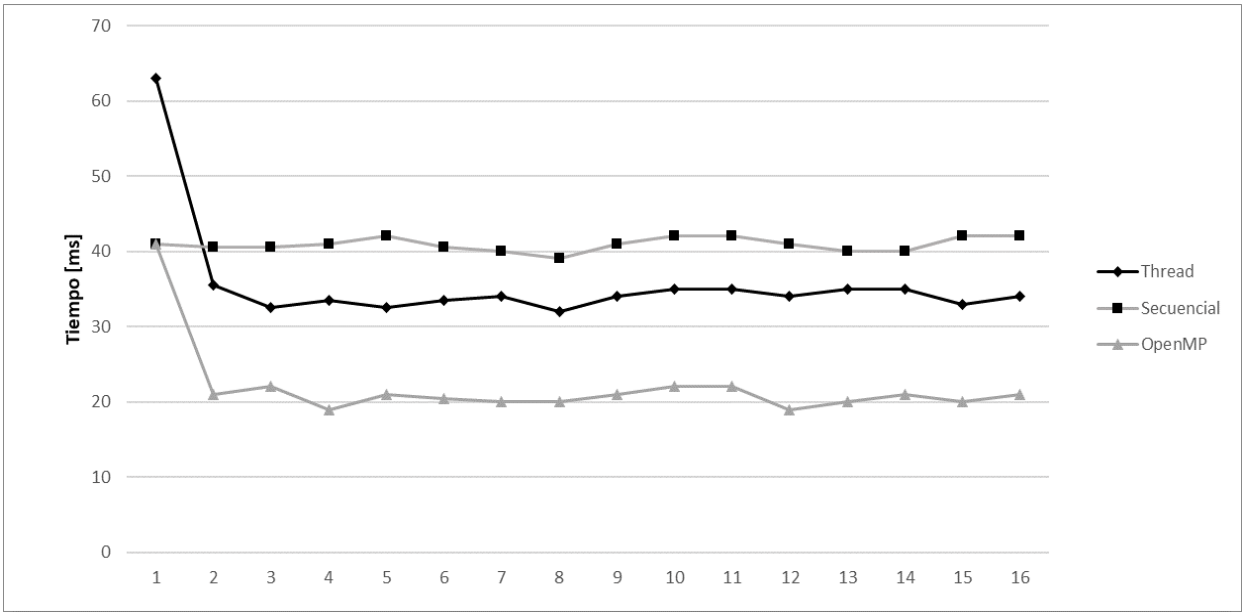

Figura 9 Grafico Tiempo de Suma vs cantidad de threards.

En la figura 9 se muestra el promedio de los datos para la ejecución de suma de los valores aleatorios. Se puede observar que cuando se ejecuta con un solo hilo el tiempo Secuencial y con la aplicación OpenMP es la misma y eso se explica debido que la tarea no es "dividida" si no se trabaja de la misma forma. Con cuanta más cantidad de hilos se utiliza mejor será el rendimiento, pero luego del cuarto hilo se estabiliza el tiempo. Como añadido a este gráfico, al igual que el anterior, se incluyo la grafica implementada en el taller dos(thread) y se puede notar como OpenMP genera una mejor ejecución e implementación de hilos. Como se aprecia en la gráfica, cuando se utiliza solo un hilo para la ejecución thread el tiempo es mayor que para la ejecución de manera secuencial, esto es debido a que el proceso para obtener la suma total tiene mayor trabajo que de manera secuencial.

Es destacable que las pruebas fueron con los mismos datos y rango de ellos.

Como último dato importante, es destacable aclarar que los hilos en ambas graficas solo afectan a al apartado paralelo, ya que la ejecución secuencial no se ve perjudicada ni beneficiada por ellos. El motivo de los gráficos es la comparación de ambos métodos y medirlos en términos de tiempo y eficiencia.

Los datos de mi equipo junto, el de la máquina virtual y los parámetros utilizados para realizar las pruebas se muestran en las siguientes tablas:

Tabla 2 Parámetros utilizados

PARÁMETROS	TAMAÑO
-N	100000000
-t	Variable
-l	1
-L	1000000

Tabla 3 Datos máquina virtual

TITULO	DESCRIPCIÓN
Memoria base	**2048 MB**
Procesadores	**3**
Aceleración	**VT-x/AMD-V, Paginación anidada, Paravirtualización KVM**

Tabla 4 Datos de mi equipo

TITULO	DESCRIPCIÓN
Procesador	**intel® Core ™ i5 -7400 CPU @ 3.00GHZ 3.00 GHZ**
RAM	**16,0 GB**
Sistema operativo	**64 bits**

Las tablas 2, 3 y 4 fueron mencionadas debido a que la ejecución puede variar dependiendo de la capacidad física de cada equipo. Este punto será explicado en mayor detalle en la conclusión.

4. Conclusión

De esta manera y mediante el desarrollo del código realizado gracias a un correcto diseño previamente realizado se corroboro la utilidad de OpenMP mejorando los tiempos de ejecución sin un gran desarrollo del código. Las gráficas presentadas respaldan el correcto uso de ellos ya que, incluso, mejora el tiempo obtenido en el taller dos en el cual se trabajó con hilos para mejorar el tiempo de ejecución de la misma tarea. Además, puedo agregar que el uso de OpenMP facilita la paralelización de código y genera un buen tiempo de desempeño en comparación a lo ya presentado, no obstante, esto no significa que no pueda existir otro método o código el cual reduzca aun más los tiempos de ejecución para el mismo código. Otro detalle que quiero hacer mención es la notoria aparición de outliers dentro de las pruebas con OpenMP, cada cinco pruebas aproximadamente. Como detalle agrego que para una gran cantidad de datos (800.000.000) puede que no sea factible el uso de los threads utilizados en el taller dos, debido que el proceso de sumar los datos (sumas de sumas) obtenidos por cada thread puede extender el tiempo de ejecución de este. Este caso me ocurrió al hacer las pruebas en mi equipo y luego en la máquina virtual disminuye esa cantidad máxima debido a que la maquina virtual no posee la misma capacidad física que el equipo hospedador. Este punto fue probado tanto con el código desarrollado en el taller dos y con la implementación de OpenMP y puedo llegar a entender que siempre existirá una limitante física a la hora de trabajar con hilos.

Finalmente, gracias a las pruebas realizadas y el estudio de los conceptos asociados se logró generar un código que cumple con los requerimientos iniciales, pero esto no significa que nos pueda ser optimizado en algún futuro.

5. Referencias

[1] The cplusplus website. [Online]. Avalaible: http://www.cplusplus.com