Reporte Técnico: Taller 03

Taller de Sistemas Operativos
Escuela de Ingeniería Informática

Fernando Del Pino Machuca

Fernando.delpino@alumnos.uv.cl

Resumen. *En este reporte se documenta y plantea, la utilización del lenguaje "C++" para manejar el llenado de un arreglo de números enteros, los cuales procederán a ser sumados y mostrar la suma total por pantalla. Este proceso consiste en 2 módulos que deben ejecutarse en forma paralela mediante la implementación de la API OpenMP, donde además se realizaron comparaciones de tiempo de ejecución con respecto al taller anterior que presentaba las mismas características, aunque fue realizado en un proceso de manejo propio de Threads. El presente reporte se basa a grandes rasgos del modelamiento de una solución posible, para tener mayor claridad en los procesos siguientes relacionados con la codificación de la solución y su uso, para luego analizar los resultados obtenidos por parte de los tiempos de ejecución de las 2 implementaciones de manera comparativa.*

1. Introducción

1.1. Problema

Este reporte se basa en la implementación de un programa que llene un arreglo de números enteros donde estos se sumen, realizando estas tareas de forma paralela, proceso implementado con "Threads" (Hilos) POSIX y "OpenMP" . Se debe crear un programa que esté compuesto de dos módulos. Uno que llene un arreglo de números enteros aleatorios del tipo "uint32_t"(enteros sin signo de 32 bits) en forma paralela y otro que sume el contenido del arreglo también en forma paralela. Se deben hacer pruebas de desempeño que generen datos que permitan visualizar el comportamiento del tiempo de ejecución de ambos módulos dependiendo del tamaño del problema y de la cantidad de hilos utilizados. Para generar números aleatorios, se debe utilizar funciones que sean "Thread Safe" para que observe una mejora en el desempeño del programa. Junto a esto, el programa debe ser ejecutado con parámetros establecidos (Mostrados en la tabla 1), donde estos parámetros están especificados en la tabla 2.

Tabla 1. Forma de uso:

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

Tabla 2. Parámetros:

-N : Tamaño del arreglo

-t : Numero de Threads.

-l : Límite inferior del rango aleatorio

-L : Límite superior del rango aleatorio

[-h] : Muestra la ayuda de uso y termina

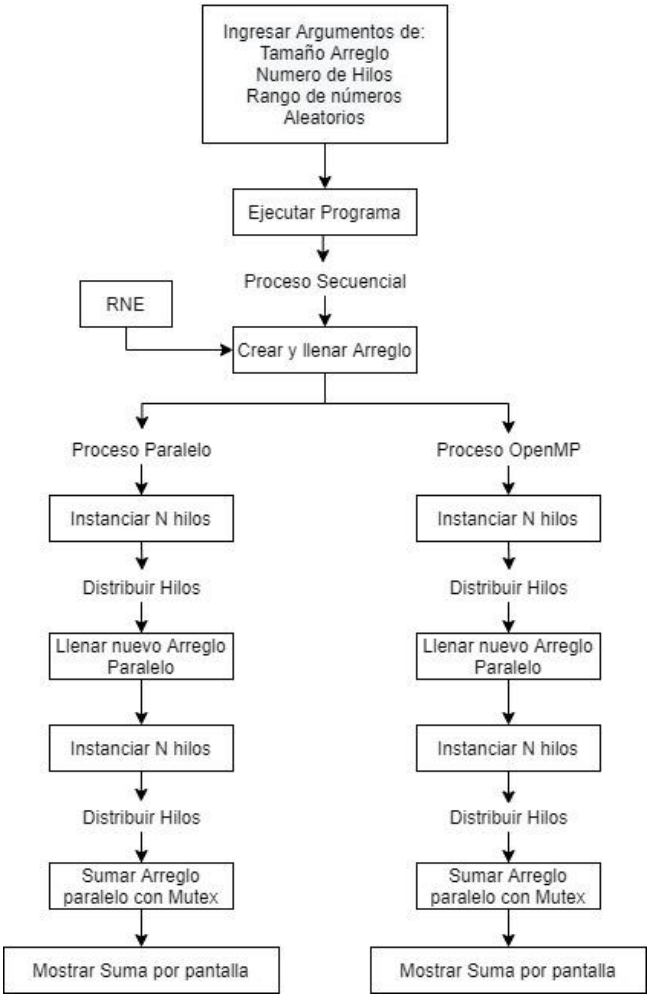
Cada parámetro deberá ser utilizado para especificar el funcionamiento del programa, especificando la cantidad de hilos a crear, el tamaño del arreglo y los límites de los números aleatorios que deben ser rellenados sobre el arreglo.

1.2. Análisis

Analizando el actual taller, se busca la implementación de un programa que llene un arreglo de números enteros, donde posteriormente, los números de este arreglo sean sumados de forma paralela, a raíz de la implementación de "OpenMP"[1]. OpenMP es una API(Interfaz de programación de aplicaciones) para realizar explícitamente paralelismo de múltiples hilos para el manejo de memoria compartida, en lenguajes como Fortran y C/C++. El uso de esta librería nos ayudará a comparar el trabajo del taller previamente realizado con respecto al manejo correcto de hilos, mostrándonos si la implementación propia del trabajo de llenado y suma fue correctamente implementado, o no cumplió con lo esperado. Esto será comprobado gracias a las medidas de desempeño, con respecto a los tiempos de procesamiento, a la hora de realizar el llenado en forma serial y paralela, por parte de la implementación anterior, como también el llenado y sumado de forma serial y paralela con el uso de la librería OpenMP. En resumidas cuentas, la codificación anterior de la solución y la actual, se deben basar en un proceso de llenado y suma concurrente, utilizando métodos de paralelización [2] para proporcionar mejoras de rendimiento de procesado en cuanto a los bucles y trabajos del código. Estos métodos de paralelización abarcan el paradigma de programación paralela, proceso que se basa en realizar varias tareas e instrucciones de forma simultánea, esto gracias al principio de "dividir y vencer", principio enfocado a tomar grandes tareas y dividirlas en pequeñas tareas, las cuales se entregan a distintos receptores de procesamiento que se encargaran de trabajar simultáneamente, agilizando el procedimiento de cómputo y ejecución. Para que este proceso de paralelización se lleve correctamente a cabo, se deben disponer de algoritmos de exclusión mutua, alias "mutex" (mutual exclusion), para evitar que más de uno de estos procesos ingrese a la sección crítica(fragmento del código o proceso donde se modifica el recurso compartido), ayudando así a mantener una consistencia de los datos a la hora de trabajar con información compartida y variables globales. Este proceso del manejo de hilos corresponde a lo establecido para mantener un desarrollo "Thread Safe"(seguridad en hilos) [3], el cual se basa en satisfacer que múltiples hilos puedan acceder a los mismos datos compartidos,

manteniendo la integridad de estos datos junto con minimizar el comportamiento inesperado a la hora de ejecución de estos hilos. El entendimiento del problema se ve reflejado en la Fig. 1.

Figura 1 Objetivo General del Problema.



Es posible ver cómo se debería comportar el arreglo de manera general y su llenado según se comprende. Es necesario destacar que “RNE” (Random Number Engine) es el encargado de generar números aleatorios, para llenar el arreglo vacío inicial, para que luego se divida en 2 procesos, un proceso que se realizará mediante la programación del taller previo, y el otro proceso mediante el manejo de hilos a través de OpenMP, donde ambos seguirán rellenando su respectivo arreglo a través de N hilos de llenado, para luego realizar el sumado en el módulo siguiente por los nuevos N Hilos, mostrando finalmente por pantalla la suma total de los 2 procesos. La cantidad de hilos creados es instanciada a la hora de ejecutar el programa, donde el módulo de llenado y suma deben tener a lo menos 1 hilo cada uno que los ejecute. Mientras más hilos creados haya, cada tarea de los módulos debe dividirse en pequeñas partes para cada hilo, donde habrá muchos hilos que guarden datos en el mismo arreglo, y muchos hilos que sumen los datos del arreglo. Su relación está basada con respecto a “N:N”,

que quiere decir que la cantidad de hilos que guarden datos, serán la misma cantidad de hilos que sumen los datos. También se comprende que, para analizar el desempeño del programa, se debe hacer estos módulos de manera serial para ir comparando su efectividad, tanto de la forma diseñada previamente, como también con el uso de OpenMP.

En base a esto se formula la estructura de este reporte, abarcando el diseño de la solución al problema planteado, junto con su comprensión respecto al módulo de llenado y el módulo de sumado, para luego abarcar los resultados obtenidos a raíz de la implementación de los diseños planteados anteriormente, concluyendo con las observaciones y análisis de todo lo realizado.

Antes de profundizar en el procedimiento de diseño, es necesario destacar que todo el proceso de desarrollo se realizará a través de la máquina virtual instanciada en estudios previos, la cual se encuentra establecida con sistema operativo Ubuntu (distribución de Linux), a la cual se le instalo los paquetes de las aplicaciones respectivas para la compilación y ejecución de programas basados en lenguaje "C++", además de comprobar que el sistema actualmente poseía las librerías utilizadas para el desarrollo con OpenMP mediante el siguiente comando:

- "apt show libomp-dev"

Mostrando así, la versión del paquete de dependencias de OpenMP y comprobando que se poseían las librerías a utilizar por parte del programa.

2. Diseño

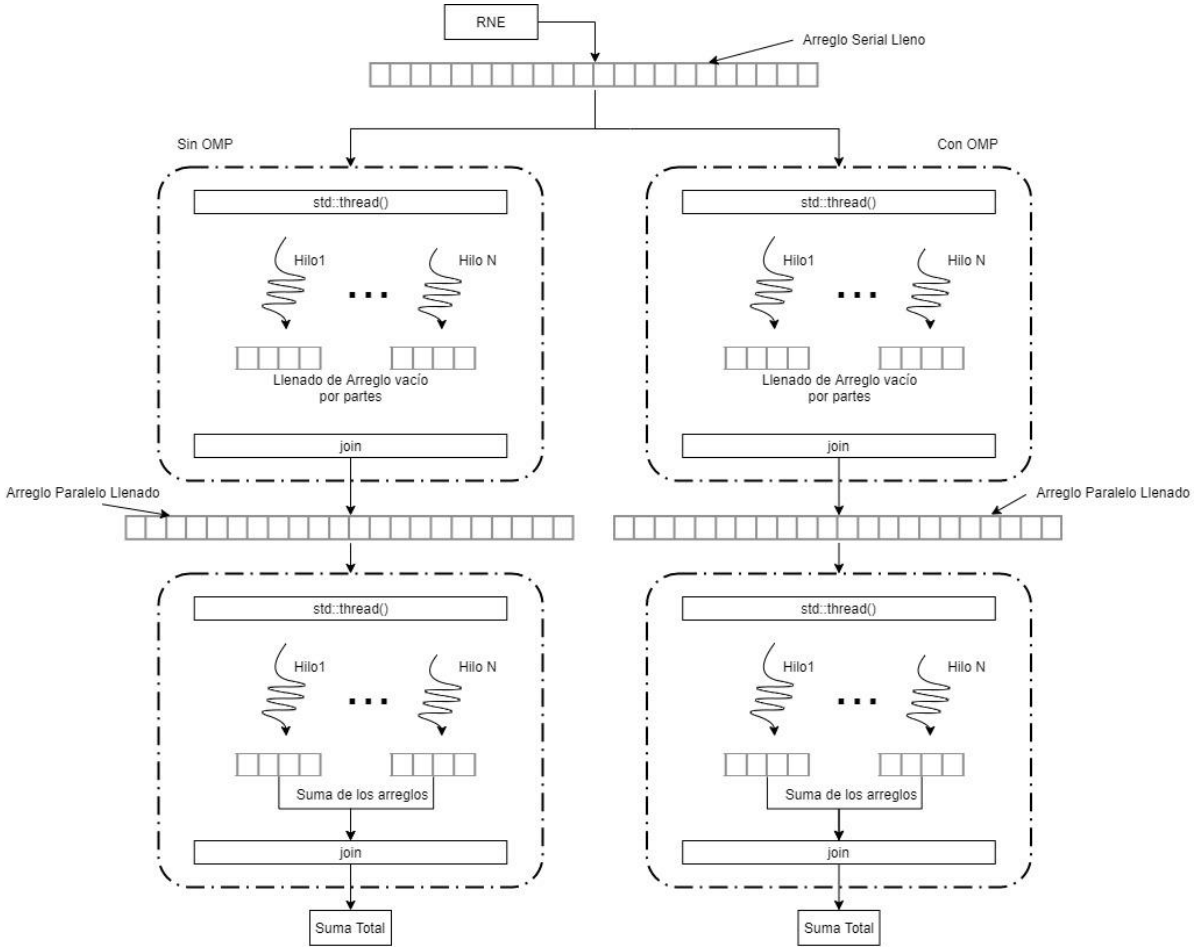
2.1. Metodología:

La metodología para abarcar las tareas propuestas por cada módulo fue analizar cada requerimiento, diseñar un modelo general del funcionamiento del programa, donde se vea reflejado un orden de procesos a realizar antes, para lograr el llenado y suma de los arreglos. Una vez analizado el funcionamiento general del programa, se debe crear un modelo de diseño específico para cada módulo del taller, eso quiere decir, que se debe idear un diseño de comportamiento para cada módulo, ya que cada módulo presenta una estructura y modo de ejecutar cada tarea de manera distinta, por lo que una vez especificado, se procederá a utilizar como ejemplo para su implementación en código.

2.2. Descripción General:

Para analizar el funcionamiento del programa, se debe comprender de manera general cómo se realiza cada proceso, proceso tal para el funcionamiento de lo realizado en el taller anterior, como también para el funcionamiento con el uso de OpenMP, donde de manera general los 2 se realizan según el diseño presentado en la Fig.2, bajo el análisis del manejo de hilos para los 2 módulos.

Figura 2. Diseño general de la solución

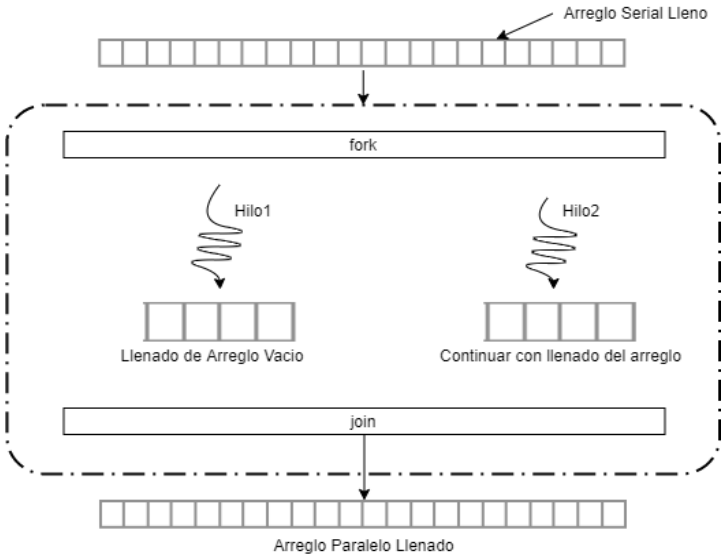


Como se puede observar, se realiza de la misma manera que el taller previo, donde su mayor diferencia viene desde la implementación con OpenMP(OMP), pero viéndolo desde el punto de vista de diseño, deberían realizar las mismas tareas, referentes a exigir argumentos de entrada en tiempo de ejecución, para definir el tamaño del arreglo, número de hilos a crear y el rango de los números aleatorios que se pueden presentar (este rango ingresa como límite inferior y límite superior). Para esto se puede ver que es necesario comprobar que los parámetros entregados son correctos. Una vez ejecutado el programa este debe comenzar secuencialmente a crear el arreglo vacío con el tamaño establecido, rellenar un arreglo en serie de números aleatorios, para luego crear la cantidad respectiva de hilos, tanto de la manera implementada en el taller anterior, como con la nueva implementación de "#pragma omp parallel" [4] para realizarlo mediante OpenMP. En base a esto se debe disponer tanto para lo realizado anteriormente, como para lo nuevo, de un número de hilos entre módulo 1 y módulo 2, para que completen las tareas de llenado y suma del arreglo exigido.

2.3. Módulo de Llenado:

Una vez realizado la estructuración del diseño general de la solución, se puede diseñar el módulo respectivo a la etapa de llenado del arreglo. El diseño de este módulo se puede ver estructurado en Fig. 3.

Figura 3. Diseño modulo llenado del arreglo.

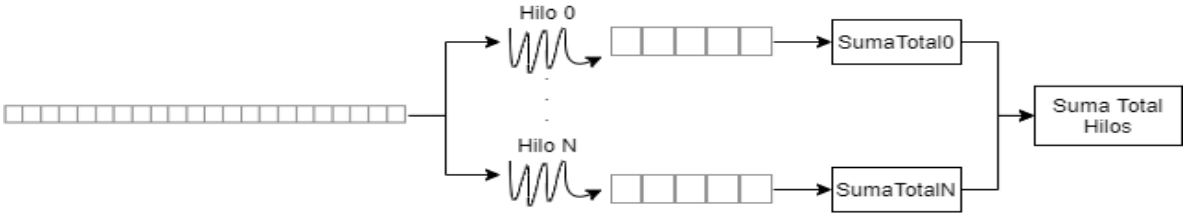


En primer lugar, se estimó que este proceso, tanto para lo desarrollado anteriormente, como para lo realizado mediante OMP(OpenMP), sería implementado de la misma manera, debido a que, a ambos se les entregará un arreglo llenado de manera serial, con números aleatorios, para que a través de la utilización de Threads, se realizará un proceso de llenado paralelo en cuanto a creación de hilos, llenados por parte de un arreglo local para cada hilo, donde finalmente se consolidarán los datos en un arreglo de elementos, llenados paralelamente por 2 distintos procesos. Cabe destacar que es muy posible que el tiempo de llenado se vea afectado por factores como no conocer técnicamente el funcionamiento interno de OMP, sino más bien, se entiende como debería comportarse, al ser una librería que posee sus propias funciones y características, esta puede implementar mayores tiempos de ejecución como también menores, al presentarse a simple vista como un módulo de caja negra que simplemente se le ingresan los parámetros para ejecutar sus tareas de paralelización.

2.4. Módulo de Suma:

Tras rellenar el arreglo, a través de los 2 procesos, se debe instanciar nuevamente cada hilo especificado, para que realice el proceso de suma del contenido del arreglo. El módulo de suma se ve representado en la Fig. 4.

Figura 4. Diseño modulo suma del arreglo.



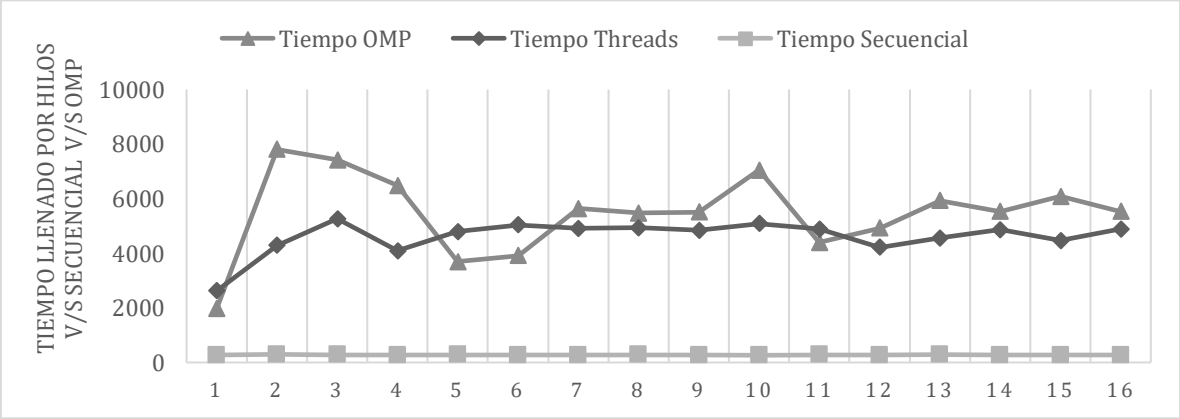
El diseño se basa en que cada hilo instanciado, tome una parte del arreglo paralelo rellenado, considerado que cada hilo posee pedazos del arreglo original, continuando con el respectivo proceso de suma de cada elemento de estos sub-arreglos. Finalmente, cada una de estas sumas, serán nuevamente sumadas para obtener la suma total de cada hilo.

3. Resultados

Una vez se implementó la codificación de los diseños planteados, se analizó los respectivos resultados en comparación de los tiempos de ejecución de los módulos de manera secuencial y paralela versus la implementación con OMP. A primeras se analizó el comportamiento del programa con distintos rangos de elementos para incluir dentro de los arreglos, donde se estimó conveniente trabajar con arreglos de 100000000 elementos, donde este se aproximaba al límite de elementos con los que podía trabajar la máquina virtual sin que tuviera un volcado de memoria o comenzará a entregar datos erróneos con respecto a las capacidades que poseía esta máquina virtual, como también sin que esta cortara la ejecución del programa debido a la alta cantidad de procesamiento que tomaba (Caso que pasó más de alguna vez cuando se le ingresaron muchos más elementos al arreglo y el llenado por parte de hilos no se concluyó o inicio).

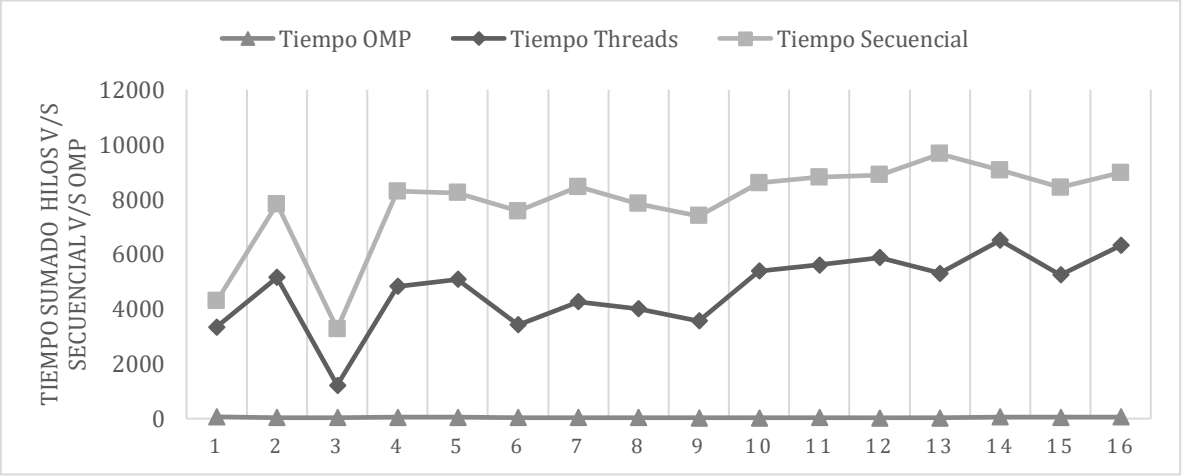
A raíz de una serie de pruebas, se promediaron los datos con respecto a los tiempos de ejecución de cada tarea dependiendo de la cantidad de hilos utilizados, los cuales fueron de 1 a 16 hilos, en base a esto la Fig.5 muestra los promedios de cada prueba realizada, con respecto al tiempo de llenado de los arreglos de manera secuencial, en paralelo y mediante OMP.

Figura 5. Gráfica del tiempo de Llenado con Hilos y en Serie



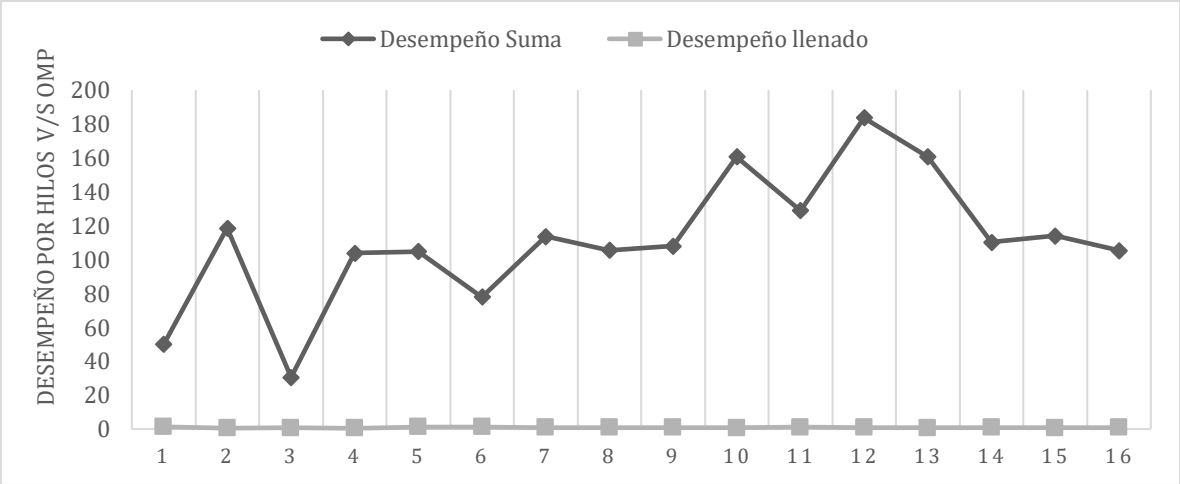
Se puede observar que el tiempo de llenado en secuencia tomo muy poco tiempo en comparación a los tiempos de llenado por hilos en paralelos y mediante OMP, esto puede deberse a varios factores, donde se realizaron pruebas en distintas máquinas virtuales para comprobar la veracidad y consistencia de estos datos, donde las soluciones obtenidas óptimas fueron las registradas. Esto nos dice que muy posiblemente el problema venga desde la implementación de código, que como ya se había mencionado en el taller anterior, el manejo de los hilos no se realizó de la mejor manera. Aunque, viéndolo desde otro punto de vista, se comprende que extrañamente el llenado de manera secuencial siempre fue la solución óptima, pero debido a esto, se puede entender que el llenado a través de un arreglo previamente llenado, pudo no ser la mejor decisión a la hora de implementación, debido a la cantidad de problemas presentados, ya que los resultados obtenidos no eran los esperados. Se analizará más en profundidad y realizarán procesos de mejoras para el estudio propio sobre estos problemas de implementación de código

Figura 6. Grafica del tiempo de Sumado con Hilos y en Serie



El grafico de la Fig.6, presenta una situación mucho mejor para el uso de OpenMP, ya que su desempeño fue mucho mayor que la suma con hilos y en secuencia. Si bien es cierto, esta vez se realizaron mejor las pruebas para el programa en el modulo de sumando, presentando que, en este taller, a diferencia del anterior, el modulo de sumado con Threads, tuvo una mejora de los tiempos de sumado de los arreglos en forma paralela en contraste con los tiempos de sumado en serie. Como era de esperarse, el desempeño de OMP fue mucho mayor que nuestra implementación del taller pasado, lo que nos hace cuestionarnos que se puede mejorar mucho mas la manera en que sumamos los elementos de los arreglos. Cabe destacar que, para estudios personales futuros, se concluye que el modulo de llenado pudo ser una de las mayores dificultades para intentar corresponder a las expectativas del taller, con respecto a su modo de implementación para lograr un correcto manejo de los tiempos de procesamiento, como se ejemplifica claramente en la teoría del uso de estas herramientas.

Figura 7. Gráfico de Desempeño entre implementación anterior v/s uso de OMP, con respecto al módulo de Llenado y Suma



Este último gráfico (Fig.7) muestra el desempeño de los promedios de tiempo, respectivos a los módulos de llenado y suma, respectivos al desempeño obtenido por parte de la comparación entre el llenado paralelo propio y el llenado paralelo mediante la implementación de OMP, donde también se grafica el desempeño realizado por la suma de los arreglos de manera paralela como en el taller anterior, en contraste con la suma de los elementos del arreglo paralelo mediante OMP. Para aclarar un poco este gráfico, el desempeño de llenado no fue 0, si no más bien, fue cercano a 0, lo que nos demuestra que en gran cantidad de los casos el desempeño de llenado fue mucho mejor con la implementación propia, que en vez de la implementación de OMP, esto también se ve reflejado en la Fig.5, donde como ya aclaramos, el proceso de llenado fue bastante engorroso para comparar los resultados, personalmente se consideró que el uso de números aleatorios también juega un papel importante, debido a que su implementación se puede considerar un tanto difícil de manejar y prever cómo se ha visto en otras asignaturas, al menos podría ser un factor clave a considerar por parte del manejo de esta actividad, pero también se considera que una correcta implementación con un mayor flujo de control y entendimiento posiblemente pudo haber logrado mejorar estos aspectos y lograr una mejor implementación. Como otro aspecto se analiza que en todos los casos de la implementación de OpenMP, con respecto al módulo de sumado, siempre fue mucho más eficiente el uso de esta librería con respecto al desempeño que entregó está en contraste con la utilización simple de Threads. En la siguiente tabla (Tabla 3), se resume cuáles fueron los parámetros ingresados a la hora de ejecutar el programa.

Tabla 3. Parámetros:

-N : 100000000
-t : 1-16
-l : 1
-L : 50000

4. Conclusiones.

En base a los diseños planteados, se comprendió que estos eran lo suficientemente completos como para realizar todo el proceso de codificación, debido a que este taller no requería mayor complejidad en el uso de la librería OpenMP, aunque si bien es cierto, el taller anterior jugó un papel importante a la hora de la comparación e implementación de resultados esperados. Debido a esto último, el proceso de entendimiento de los resultados y análisis de estos fue un tanto complejo, debido a que se presentaron varias trabas a la hora de implementar la solución correcta que satisfaga los resultados esperados, con respecto a cómo se deben comportar los hilos en llenado y suma de elementos de un arreglo. Aparte de esto factores como el tiempo, tanto para realizar las pruebas del programa como también los tiempos de desarrollo fueron un tanto extensos, sobre todo por la recopilación de los resultados, debido a las limitaciones que presentaba el procesamiento de las tareas en la máquina virtual. Como síntesis de todo, se realizó un proceso de desarrollo con respecto a un programa que manejara hilos mediante una previa implementación y una implementación que usará OpenMP para la creación y recopilación de las tareas que realizaba cada Thread, donde se concluye que este proceso se realizó, aunque no de la manera esperada, sin obtener los resultados que se esperaba que se reflejaran a la hora de realizar este taller. Para futuros estudios personales se retomará este mismo problema con el afán de entender y resolver las problemáticas referidas a los tiempos de resultados para entender y realizar el cómo lograr la correcta implementación y uso, tanto del proceso de llenado como el proceso de suma. Sin más que decir, se concluye este reporte de actividad.

5. Referencias.

[1] Procesamiento Paralelo, Introducción a OpenMP, Javier Iparraguirre, Universidad Tecnológica Nacional, Facultad Regional Bahía Blanca, pp.10-15.

[2] Introducción a los Sistemas Operativos, Concurrencia y paralelismos, Marisa Gil, pp.10-19.

[3] An Introduction to Programming with Threads, Andrew D. Birrell, 1989, pp.03-14.

[4] OpenMP 5.0 API Syntax Reference Guide, mayo 2019 , pp.01.