Reporte Técnico: Taller 02

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, una propuesta de diseño de la solución en base al problema principal del taller, este problema consiste en 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, el llenado de números aleatorios del arreglo y el proceso de suma de los datos ingresados en el arreglo, mediante la implementación de Threads POSIX. Este proceso se debe manejar con el uso de secciones críticas para establecer la concordancia entre los datos y evitar errores de consistencia. 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.*

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. Se debe crear un programa que este 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

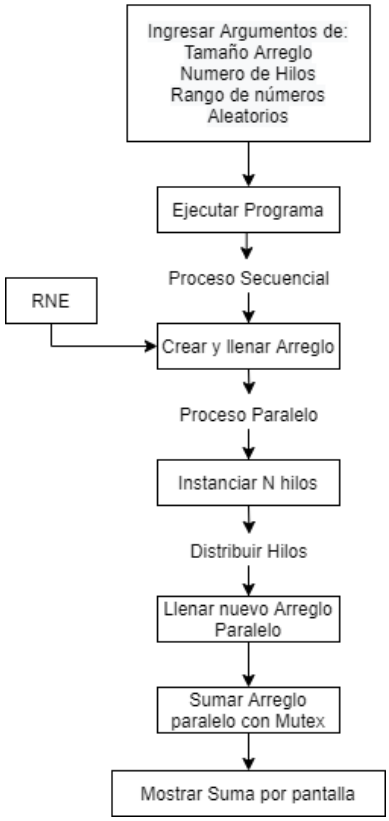
-t : Numero de Threads.
-l : Límite inferior del rango aleatorio
-L : Límite superior del rango aleatorio
[-h] : Muestra la ayuda de uso y termina

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 lo requerido, se busca la implementación de un programa que llene un arreglo de números enteros donde estos números sean sumados, mostrando el resultado total por pantalla. Para lograr la codificación de la solución se debe usar un proceso de llenado y suma concurrente, utilizando métodos de paralelización [1] 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 critica(fragmento del código o proceso donde se modifica el recurso compartido), ayudando a si 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) [2], 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 como 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 el arreglo paralelo que poseerán los N Hilos sea rellenado y luego sumado en el módulo siguiente por los nuevos N Hilos para mostrar la suma total por pantalla. 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.

En base a esto se formula la estructura de este primer reporte, abarcando el diseño de la posible solución al problema planteado, junto con su comprensión respectivo al módulo de llenado y el módulo de sumado, analizando el comportamiento para dar claridad al proceso de codificado a realizar en próximos avances.

Antes de profundizar en el procedimiento de diseño, es necesario destacar que todo el proceso de desarrollo se realizara 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++”.

2. Diseño

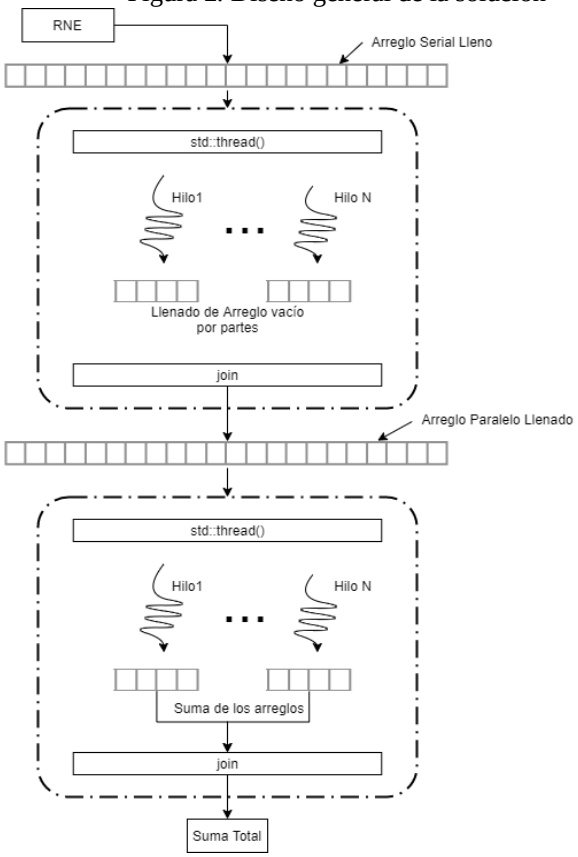
2.1. Metodología:

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 en entregas futuras.

2.2. Descripción General:

Para lograr un correcto entendimiento del funcionamiento del programa, es necesario, que se analice el proceso de forma general, además de entender como es el flujo de los datos entre modulo 1 y 2. Este modelo general se ve reflejado en la Fig. 2.

Figura 2. Diseño general de la solución

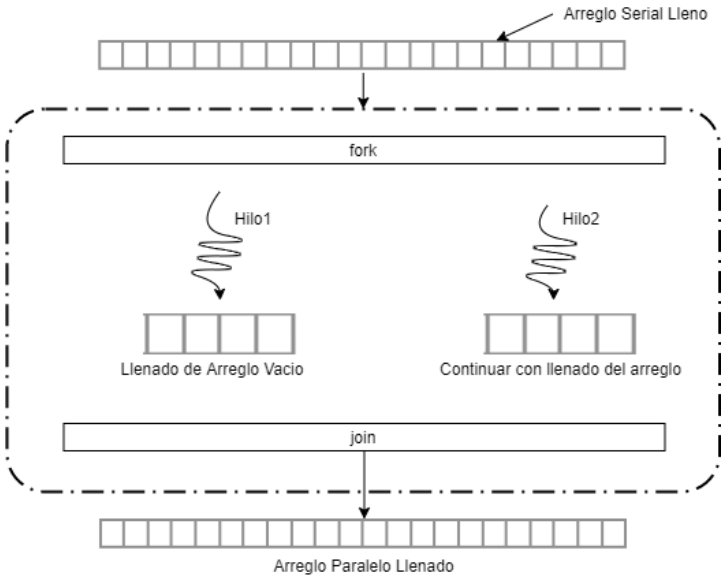


Como se puede observar, se exige que los argumentos de entrada a la hora de ejecutar el programa entreguen los parámetros para definir el tamaño del arreglo, numero de hilos a crear y el rango de los números aleatorios que se pueden presentar (este rango ingresa como límite inferior y límite superior). 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, para luego crear la cantidad respectiva de hilos que ayudaran a los diferentes módulos. En base a esto se debe disponer de un numero equilibrado de hilos entre modulo 1 y modulo 2, para que completen las tareas de llenado y suma del arreglo exigido. Una vez realizado este proceso se debe determinar cuántos hilos se usarán para el módulo 1 y asignarles las tareas de llenado del arreglo, por las partes que se estimen convenientes. Este llenado se debe hacer mediante el encargado de generar números aleatorios en base a los limites propuestos a la hora de ejecutar el programa. Tras realizar el llenado del arreglo, se debe ejecutar el módulo de suma del arreglo compuesto de la otra mitad de hilos instanciados. Se sugiere que como mínimo se creen 2 hilos (Hilo 1 e Hilo 2), para que cada uno realice un módulo y así entregar la información respectiva a la suma total de los datos.

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.



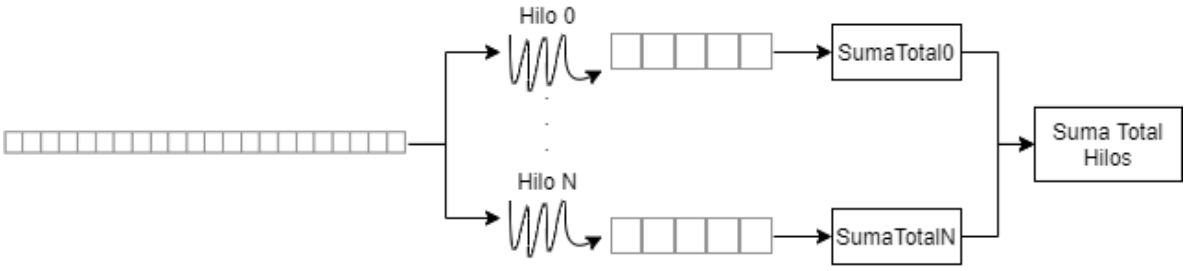
En primer lugar, se rellena un arreglo serial con números aleatorios , para que luego, el arreglo paralelo se separe en pequeños arreglos para cada hilo instanciado, una vez realizado esto, cada hilo debe rellenar cada fragmento del arreglo vacío, con los números aleatorios dentro del arreglo serial anterior. Una vez realizado

esto por cada hilo, se debe reagrupar todos los pedazos del arreglo en uno solo para ser usado luego por el siguiente modulo.

2.4. Módulo de Suma:

Tras rellenar el arreglo, 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.



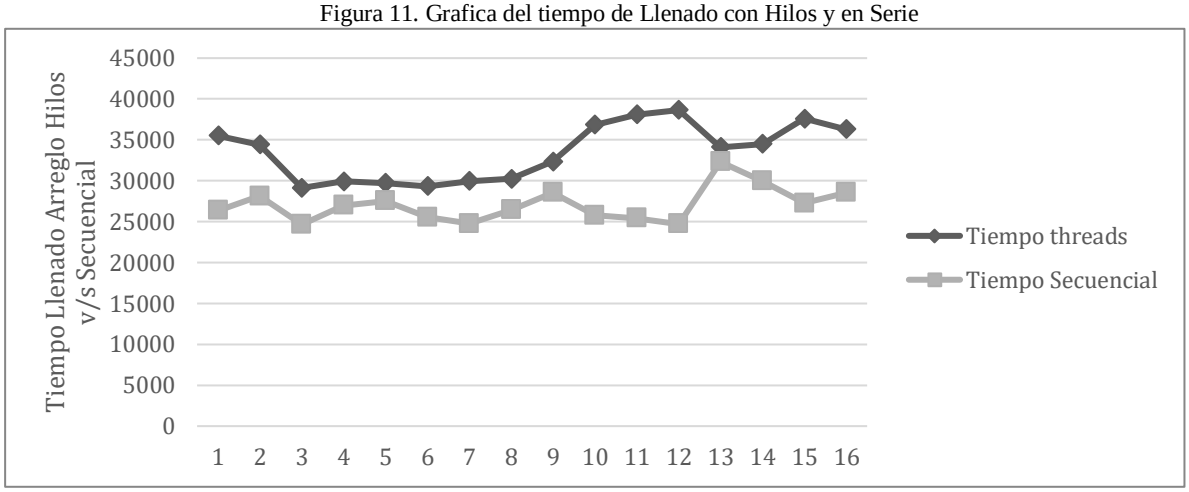
El diseño se basa en que cada hilo instanciado, tome una parte del arreglo paralelo rellenado, considerado que cada hilo posee pedazos del arreglo original, continuando con el respectivo proceso de suma de cada elemento de estos sub-arreglos. Finalmente, cada una de estas sumas, serán nuevamente sumadas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

3. Resultados

Una vez se implemento la codificación de los diseños planteados anteriormente, se analizaron los respectivos resultados con varios procesos iterativos, donde se probo varias veces la ejecución del programa con distintas directrices. A primeras se analizo el comportamiento del programa con distintos rangos de elementos para incluir dentro de los arreglos, donde se estimo conveniente trabajar con arreglos de 100000000 elementos, donde este se aproximaba al limite de elementos con los que podía trabajar la maquina virtual sin que tuviera un volcado de memoria, como también sin que esta cortara la ejecución del programa debido a la alta cantidad de procesamiento que tomaba (Caso que paso 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).

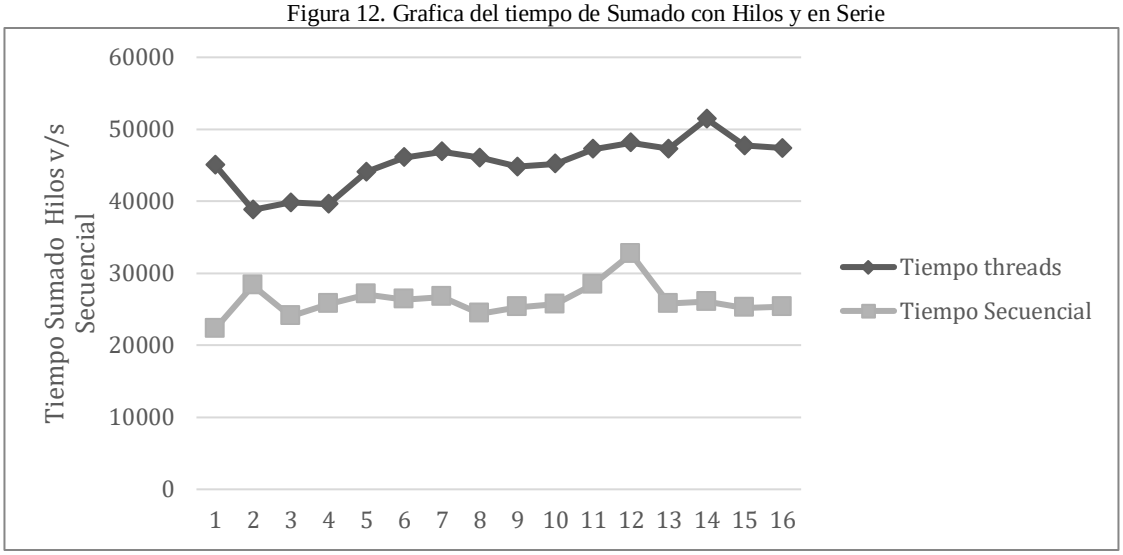
A raíz de una serie de pruebas, se promediaron los datos con respecto a los tiempos de ejecución de cada tarea dependiendo de la cantidad de hilos utilizados, los cuales fueron de 1 a 16 hilos, en base a esto la Fig.11 muestra los promedios de cada prueba realizada, con respecto al tiempo de llenado de los arreglos de manera secuencial y en paralelo.

Figura 11. Grafica del tiempo de Llenado con Hilos y en Serie



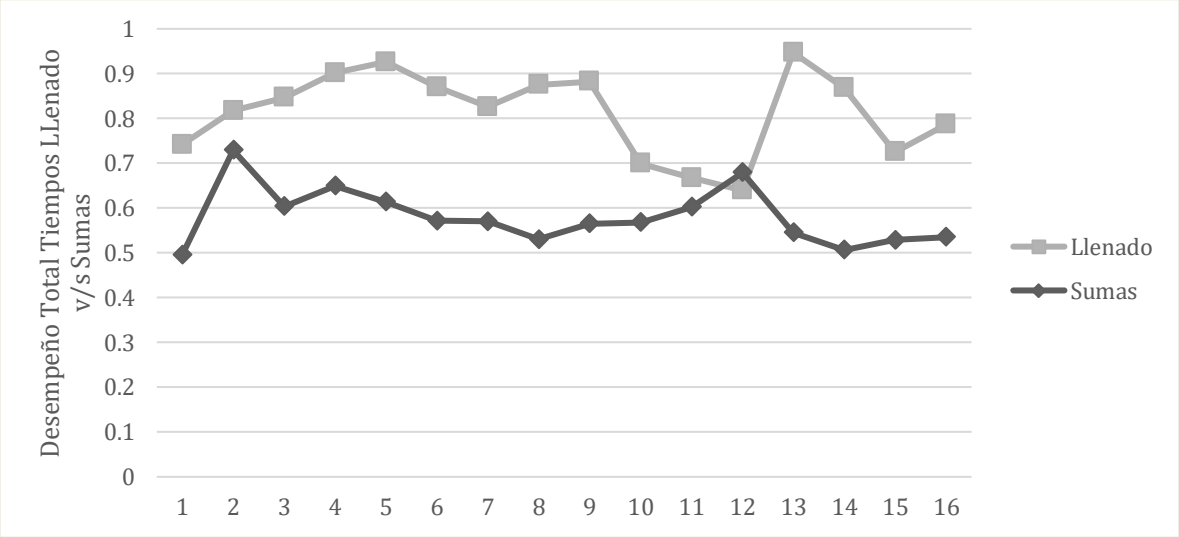
Se puede observar que el tiempo de llenado con hilos tomo más tiempo a grandes rasgos, en varios casos de las pruebas se presentó que el llenado en paralelo era mucho mejor que el llenado en serie, pero en promedio no fue así, se analiza a raíz de esto, que es posible que hayan factores como el manejo de los números aleatorios que afecten el desempeño del llenado, pero se descartó esta situación debido a que para estas pruebas, esa situación había sido reemplazada llenando 2 veces los arreglos, es decir, se llenó primero un arreglo de manera serial con números aleatorios dentro de los rangos establecidos, para que después cuando se solicitara el llenado a la función de llenado, la suma serial y en paralelo llenarían otro arreglo a raíz del arreglo previamente generado, cumpliendo con la necesidad de llenar un arreglo con números aleatorios de igual manera. Otro caso que pudo afectar es que no conocemos como maneja la creación y coordinación de ejecución de los hilos cada procesador, por lo que de manera específica también pudo ser una razón que pudo afectar el desempeño de ejecución de este módulo. Se estima conveniente para estudios futuros el analizar y posiblemente mejorar la implementación del código como también otros métodos de ejecución de las pruebas que pudieron afectar el manejo de estos hilos por parte de la máquina virtual, como añadiendo más características a la maquina o estableciendo otra con mayores especificaciones. Concluyendo esta figura, se puede comprender que gran parte del desempeño de los hilos se optimizo cuando se manejaron los arreglos con 2 hilos, es decir, el tiempo mejoro considerablemente con el uso de mas de 1 solo hilo y así con el tercero, para mantenerse estable con el llenado hasta la creación y manejo del octavo hilo, donde una vez creado el noveno hilo y los siguientes, el trabajo que le tomo a la máquina virtual para llenar los arreglos tomo mucho mas trabajo de procesamiento, esto puede deberse muy posiblemente a que nuestro equipo posee 8 procesadores lógicos (Hilos), donde utilizaron su máximo desempeño hasta que se quiso crear mas de los hilos que se recomiendan, donde este proceso se volvió mas lento y engorroso para manejar la cantidad de elementos que se le entregaron. En base a esto también se analiza que gran parte de los tiempos de llenado en forma serial, se vieron afectados en ciertos momentos aumentando de gran manera los tiempos de llenado y ejecución de esto, un dato importante que puede deberse a algunos elementos que quizás no consideremos dentro del desempeño y tareas que realiza la máquina virtual.

Figura 12. Grafica del tiempo de Sumado con Hilos y en Serie



El grafico de la Fig.12, presenta la misma situación anterior, donde el tiempo de sumado por parte de los hilos fue mucho más que el tiempo de sumado en serie. Aunque esta situación haya presentado este mismo inconveniente, este se considera y conoce que es netamente por parte de la implementación del código, debido a que, para este caso, era indispensable que el tiempo de sumado fuera mayor que el de llenado sobre todo que aumentaría mientras más hilos se utilizaran. Esto último es debido a que la implementación del código, para considerar que el sumado en paralelo tuviera la consistencia requerida, es decir, que la suma realizada por los hilos fuera exactamente igual que la suma total en serie, se consideró la implementación de la exclusión mutua (mutex), analizando su utilización como candados que permitan establecer la consistencia de los datos a la hora de que los hilos no alteraran la suma total. Es debido a esto que los tiempos de sumado iban a ser indudablemente mayores que los tiempos de sumado en serie, ya que la variable de suma total no se iba a disponer a otros hilos hasta que el proceso de suma fuera soltado por el hilo en ejecución. Si bien es cierto, el uso de exclusión mutua pudo haber sido reemplazado por otro mecanismo de codificación como el almacenamiento de las sumas dentro de otro vector, pero, esta idea de solución surgió una vez ya realizado este proceso de pruebas e implementación, a raíz del análisis y conversaciones con otros alumnos de la asignatura, consolidando finalmente la suma del contenido del vector de elementos. Esta última observación se estimará conveniente desarrollar e implementar en algún proceso posterior a la entrega de este taller, para comprobar su factibilidad y uso.

Figura 13. Grafico de Desempeño del Tiempo Total del Módulo de Llenado y Suma



Este último grafico (Fig.13) muestra el desempeño de los promedios de tiempo, respectivos a los módulos de llenado y suma, considerando que este desempeño se realizo dividiendo los tiempos promedio de llenado en forma serial por el promedio de llenado en forma paralela, como también la división de los tiempos promedios de la suma en serie por la suma en paralelo. Analizando el grafico, se entiende que el proceso de llenado tomo mucho mas tiempo que el proceso de sumas, donde en zonas características como la utilización de 12 hilos correspondió a un factor bastante extraño, debido a que fue un punto critico para ambos, desempeño de suma y desempeño de llenado. Comprendiendo un poco su actividad, el proceso de llenado correspondiente a la utilización de hilos, presento un mejor desempeño que el modulo de sumas, esto puede deberse tanto a su manera de implementarse como también a la forma en que rellenan un arreglo los hilos con respecto a como suman. Hay que considerar que la utilización de candados impacto en gran manera el desempeño del programa, donde nos demuestra que muy posiblemente no sea la mejor manera de implementarse. Podemos también comprender que el mejor desempeño que tuvieron los hilos a la hora de realizar el llenado fue cerca de los 13 hilos, donde analizando figuras anteriores, este casi fue mejor que el llenado del arreglo en serie. La suma presento un mejor desempeño cuando se utilizaron 2 y 12 hilos, aunque cuando se utilizaron 2 hubo mucha mejoría en cuanto al uso de 1 solo hilo, cuando se utilizaron 12 no lo hubo, si no que la suma en serie se comporto de manera que utilizo mas tiempo para su ejecución, dato a tener en cuenta que a pesar de presentarse muchas veces por así decirse constante su ejecución, de vez en cuando presentaba mayor tiempo de complejidad a la hora de ejecución.

4. Conclusiones.

En base a los diseños planteados, se consideró que su entendimiento era suficiente para la correcta implementación del código, pero dados los resultados, no se concordó con los resultados que se esperaban, refiriéndonos a una gran mejora esperada en los tiempos de llenado y suma gracias a la utilización de programación paralela. En base a este problema, se estima conveniente realizar un estudio más profundo del

comportamiento del procesamiento de los resultados obtenidos, para analizar de mejor manera una solución mas optima para el desempeño de cada hilo. Como se dijo en algunos ejemplos: “muchas veces no se logra mayor desempeño con programación paralela, si no más bien, con una mejor implementación del código”, donde es por esto ultimo que se buscara en algún otro momento, el retomar esta actividad para comprender los factores críticos que puedan aportar un mejora en el desarrollo de una mejor utilización de cada hilo, dejando de lado técnicas de exclusión mutua que aportan mucho más tiempo al no soltar la región critica para otros hilos, teniendo procesos de espera prolongados. En conclusión, se estudiará y analizará para conocimientos futuros una mejor opción de implementación para realizar las tareas de este taller, con la premisa de mejorar los tiempos de llenado, sumado y consistencia de datos por parte del mejor uso de hilos. Sin más que decir, se concluye este reporte de actividad.

5. Referencias.

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

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