Reporte Técnico – Taller 3

Taller de Sistemas Operativos

Escuela de Ingeniería Informática de la Universidad de Valparaíso

Yian Vera Soto

yian.vera@alumnos.uv.cl

Resumen. *Este documento está basado en el trabajo a realizar en el taller 3 del ramo de Taller de Sistemas Operativos. El objetivo de este taller es aprender el uso de threads para el procesamiento de tareas específicas. En este caso se busca llenar un arreglo con números generados con una función random y sumarlos así logrando la multitarea consiguiendo un mayor desempeño y eficiencia en el procesamiento y el tiempo, en este taller a diferencia con el anterior se solicita utilizar Implementar el Código con OpenMP.*

1. Introducción

Este reporte técnico está basado en el Taller 3 donde el objetivo de este es Implementar un programa que llene un arreglo de números enteros y luego los sume. Ambas tareas se deben realizar en forma paralela, implementadas con OpenMP y el lenguaje de programación C++ 2014 o superior,

Paralelismo es la técnica de cómputo que permite procesar varios cálculos al mismo tiempo, se basa en dividir un problema grande y trabajar simultáneamente sus partes pequeñas a nivel de bit, instrucciones, datos y tareas. Esta técnica con los años ha logrado solucionar problemas que se consideraban muy largos y costosos incluso abarcando áreas de la biología y economía.

La Multitarea en sistemas operativos es la característica que permite que varios procesos se ejecuten aparentemente al mismo tiempo dando servicio a estos procesos para que se puedan ejecutar varios programas al a vez, todo esto es debido a que se realiza una operación llamada cambio de contexto, esta quita un proceso del CPU, ingresa uno nuevo, y luego vuelve a ingresar el proceso que quitó del CPU en una especie de cola de ejecución permitiendo la multitarea

Los Threads o hilos en Informática son simplemente una tarea que puede ser ejecutada al mismo tiempo que otra tarea, estos poseen un estado de ejecución y pueden sincronizarse entre ellos para evitar problemas de compartición de recursos. Generalmente cada hilo tiene una tarea específica y determinada, para aumentar la eficiencia del uso del procesador.

POSIX significa Portable Operating System Interface (for Unix). Es un estándar orientado a facilitar la creación de aplicaciones confiables y portables. La mayoría de las versiones populares de UNIX (Linux, Mac OS X) cumplen este estándar en gran medida. La biblioteca para el manejo de hilos en POSIX es pthread.

OpenMP es una interfaz de programación de aplicaciones para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join.

Este reporte técnico consta con 6 secciones, la descripción del Problema donde se contextualizará este y se explicaran las tareas a realizar y el Diseño de la solución que explicara la solución de las tareas a través de diagramas de alto nivel y los métodos empleados para la solución, las Pruebas de Ejecución aleatorias realizadas sobre el programa, las Mediciones de Desempeño, Conclusión y Anexo donde se incluirán tablas sobre las pruebas de desempeño realizadas.

2. Descripción del Problema

El problema por enfrentar en el Taller 3 es implementar un programa que llene un arreglo de números enteros y luego los sume. Ambas tareas se deben realizar en forma paralela, implementadas con OpenMP. Y el lenguaje de programación C++ 2014 o superior.

Este programa tiene que estar compuesto de dos módulos, uno tiene que llenar un arreglo de números enteros aleatorios del tipo **`uint32_t`** en forma paralela y el otro que sume el contenido del arreglo 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 threads utilizados. También se generarán números aleatorios con funciones que sean *thread safe* para observar una mejora en el desempeño del programa

2.1. Forma de uso

```
./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   : límite superior rango aleatorio.
[-h] : muestra la ayuda de uso y termina.
```

2.2. Ejemplo de Uso

1) Crea un arreglo de 1000000 posiciones, con 4 threads. Los números enteros aleatorios están en el rango [10,50]

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

2) Muestra la ayuda y termina

```
./sumArray -h
./sumArray
```

3. Diseño de la Solución

3.1. Solución General

El diagrama de secuencia de la Figura 1 demuestra la solución general al problema al ejecutar el programa y sus relaciones con procesador, la creación y llenado del arreglo y la suma de sus valores, describiendo cómo funciona el proceso que se requiere para cumplir los dos módulos.

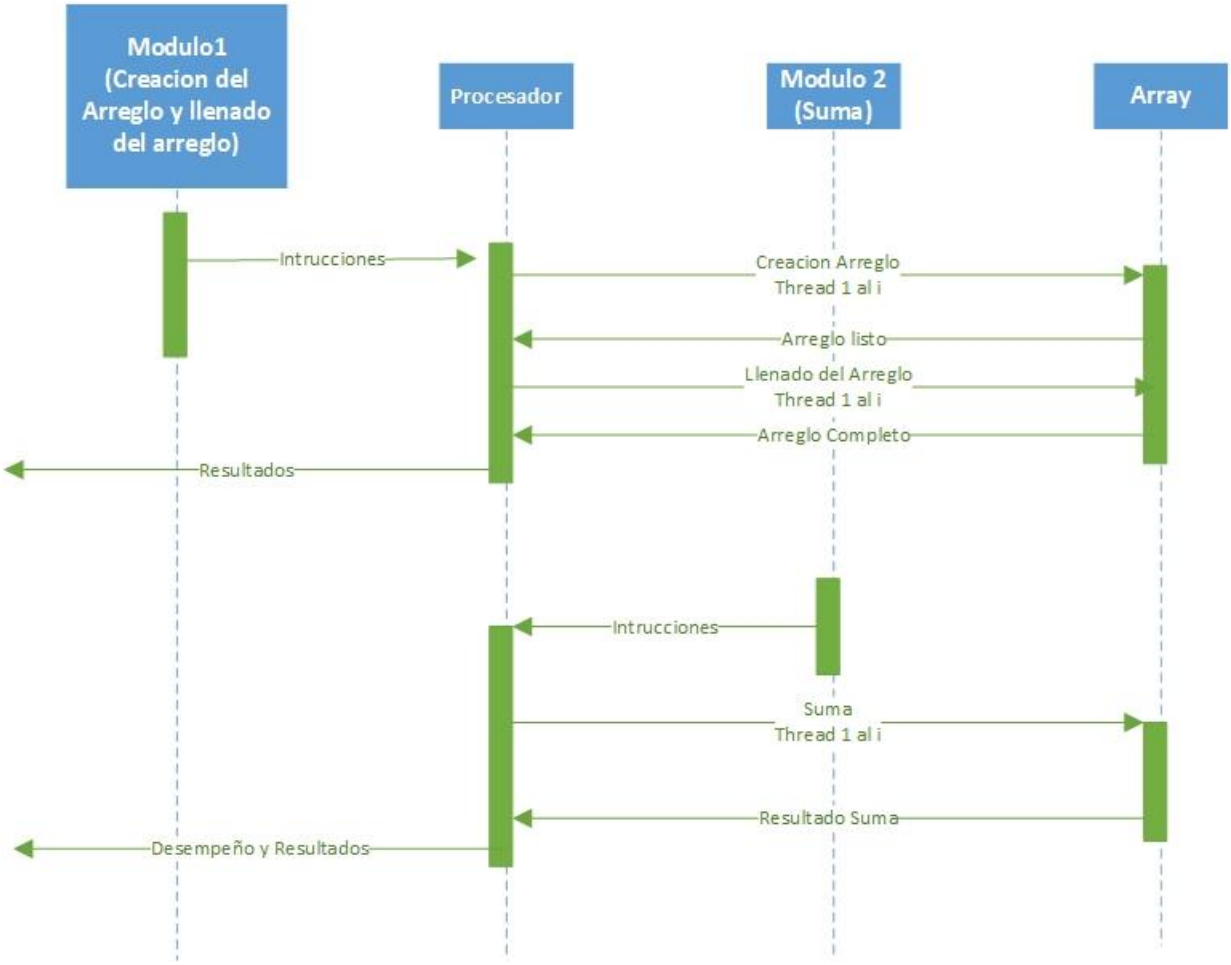

Figura 1. Diagrama de Secuencia solución general

3.2. Módulo 1

En la Figura 2 se demuestra a través de un diagrama de secuencia el proceso de llenado del arreglo con los threads, la función `Main()` se encargara de generar el arreglo, crear los threads y matarlos cuando estos hayan cumplido su tarea la que es junto a la función `random()` que generara los números aleatorios llenar el arreglo.

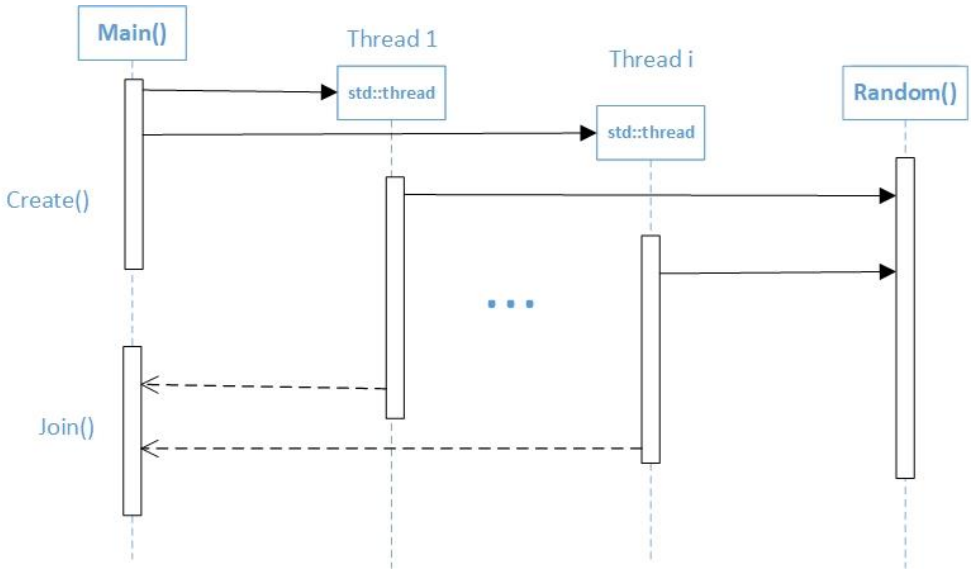

Figura 2. Funcionamiento de los threads para el llenado del arreglo

En la Figura 3. Se demuestra a través de un diagrama de flujo el funcionamiento del módulo 1, la creación y llenado del arreglo utilizando OpenMP, explicando la función de los threads para esta tarea.

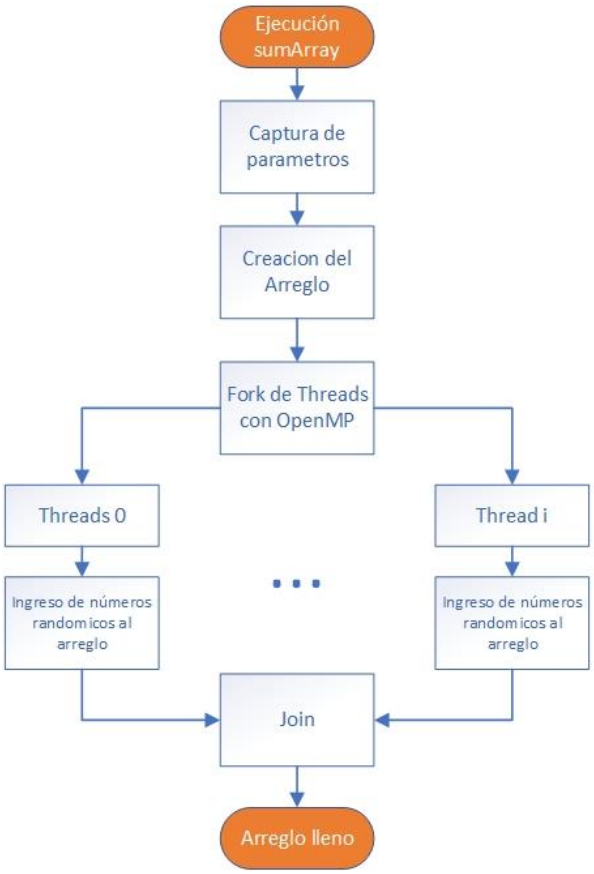

Figura 3. Creación y llenado del Arreglo

3.3. Módulo 2

En el módulo 2 se solicita sumar todos los elementos del arreglo creado en el módulo 1, esta suma tiene que ser realizada de forma paralela lo cual lo logramos usando threads con las funciones de OpenMP este proceso se explica a través un diagrama de flujo, Figura 4.

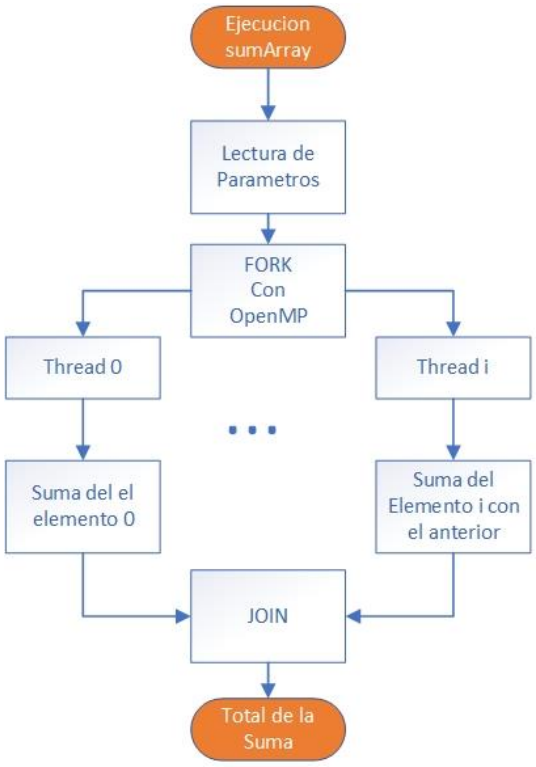

Figura 4. Módulo de suma de los elementos del arreglo

4. Pruebas

Se realizaron una serie de pruebas y estas contendrán los mismos parámetros ingresados en el taller 2, pero con la consigna de que esta vez se utilizó OpenMP. Estas pruebas se ejecutarán sobre el programa construido en el cual se reflejarán a través de capturas de pantalla sobre lo que nos arroje este, estas pruebas se realizaron sobre la máquina virtual construida en los inicios de este taller.

También para estas pruebas se rehízo la codificación de la parte paralela hecha con la librería threads del Taller 2 por una mala implementación de esta, así que en las pruebas y desempeño se tomara en cuenta.

La primera prueba corresponde a las formas de uso, en la Figura 5.

```
suvin@fracaso:~/Talleres/U2/TSSOO2$ ./sumArray
Uso: ./sumArray -N tam_problema -t threads -l limite_inferior -L limite_superior[-h] Descripción:
       -N   Tamaño del problema
       -t   Cantidad de threads a utilizar
       -l   Limite inferior del rango de numeros randomicos
       -L   Limite superior del rango de numeros randomicos
       -h   Muestra esta ayuda y termina
```

Figura 5. Formas de uso

En la segunda prueba, se ejecuta el programa con valores bajos para cada parámetro, se puede apreciar que los números son tan bajos que prácticamente el tiempo fue despreciable, Figura 6.

```
suvin@fracaso:~/Talleres/U2/TSSOO2$ ./sumArray -N 10000000 -t 1 -l 1 -L 1
======== Condiciones Iniciales =======
Total de Elementos          : 10000000
Total de Hilos              : 1
========= Llenado del arreglo ========
Tiempo Secuencial           : 244.649[ms]
Tiempo paralelo Threads     : 243.807[ms]
Tiempo paralelo OpenMP      : 237.045[ms]
=============== Sumas ================
Suma total secuencial       : 10000000
Suma total paralela Threads : 10000000
Suma total paralela OpenMP  : 10000000
========== Tiempos de Suma ==========
Tiempo secuencial           : 5.4742[ms]
Tiempo paralelo Threads     : 5.79556[ms]
Tiempo paralelo OpenMP      : 5.13018[ms]
suvin@fracaso:~/Talleres/U2/TSSOO2$
```

Figura 6. Prueba 2 con valores mínimos

En la tercera prueba se mantiene con un hilo, pero se aumentan sus valores, Figura 7.

```
suvin@fracaso:~/Talleres/U2/TSSOO2$ ./sumArray -N 30000000 -t 1 -l 1 -L 100000
======== Condiciones Iniciales =======
Total de Elementos          : 30000000
Total de Hilos              : 1
========= Llenado del arreglo ========
Tiempo Secuencial           : 731.193[ms]
Tiempo paralelo Threads     : 739.922[ms]
Tiempo paralelo OpenMP      : 718.97[ms]
=============== Sumas ================
Suma total secuencial       : 1500026813423
Suma total paralela Threads : 1500026813423
Suma total paralela OpenMP  : 1500026813423
========== Tiempos de Suma ==========
Tiempo secuencial           : 15.6465[ms]
Tiempo paralelo Threads     : 18.7696[ms]
Tiempo paralelo OpenMP      : 15.2731[ms]
suvin@fracaso:~/Talleres/U2/TSSOO2$
```

Figura 7. Prueba 3 se aumentan algunos valores

En la prueba 4 se aumentaron los hilos y se colocaron números muy grandes, Figura 8.

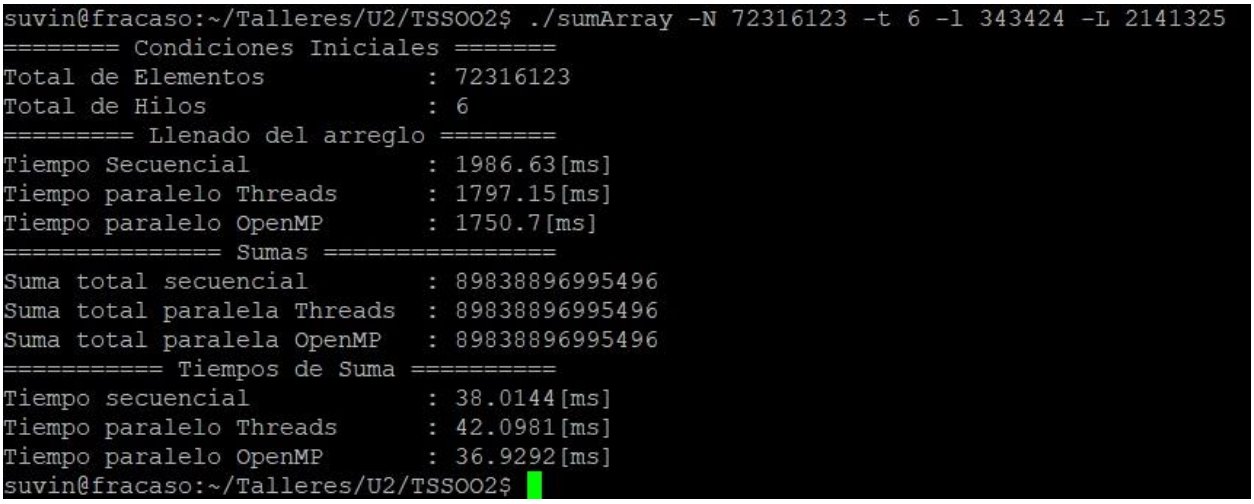

Figura 8. Prueba 4 se aumentan demasiado los valores.

5. Desempeño

Después de realizar distintas pruebas para ver el funcionamiento del programa, realizaremos pruebas de desempeño que harán una medición sobre el tiempo y el numero hilos utilizados para sobre el módulo de sumas en paralelo con la librería Thread y OpenMP, y de forma secuencial.

Al realizar varias pruebas los tiempos de trabajo variaban cuando estas eran iguales, subiendo o bajando una pequeña cantidad de tiempo, por lo que para la prueba de desempeño se realizaron 5 veces cada prueba y se calculo un promedio de los resultados obtenidos, en el Anexo se pueden ver todos estos resultados. Se utilizaron los parámetros que muestra la Tabla 1 donde el numero de hilos aumenta en 1 por prueba.

Parámetro	**Valor**
Total de Elementos	30.000.000
Numero de Hilos	{1,2,3,4,5}
Límite inferior de los números aleatorios	1
Límite superior de los números aleatorios	100.000

Tabla 1. Parámetros para medir el desempeño

La primera prueba de desempeño realizada es sobre el módulo de llenado del arreglo, en el cual se aplican tres técnicas, secuencial, paralela Thread y OpenMP los resultados se reflejan en la Figura 9. De esta medición se puede concluir que la medición mas eficiente fue la paralela con OpenMP, y entre la secuencial y Threads los resultados son parecidos siendo Thread un poco más eficiente.

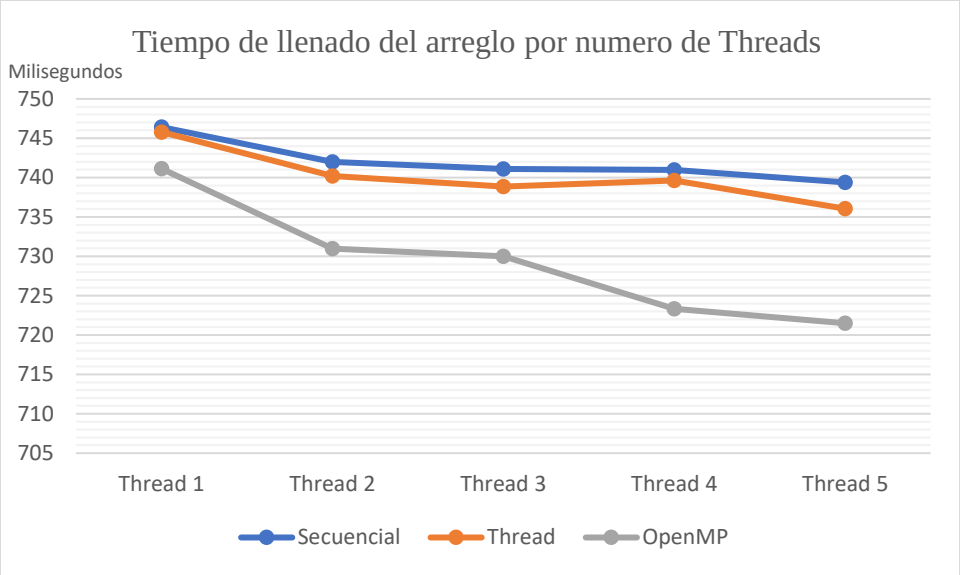

Figura 9. Tiempo de suma en paralelo por OpenMp y de forma secuencial

La segunda prueba de desempeño reflejada en la Figura 10 nos arroja que la técnica de sumado más eficiente es la de OpenMP, la técnica secuencial se mantiene constante durante la prueba con una media alrededor de los 16,5[ms], y Thread tiene una tendencia a la baja en cada prueba haciéndose mas eficiente, esta se mantiene con el tiempo después del thread4 al igual que la de OpenMP.

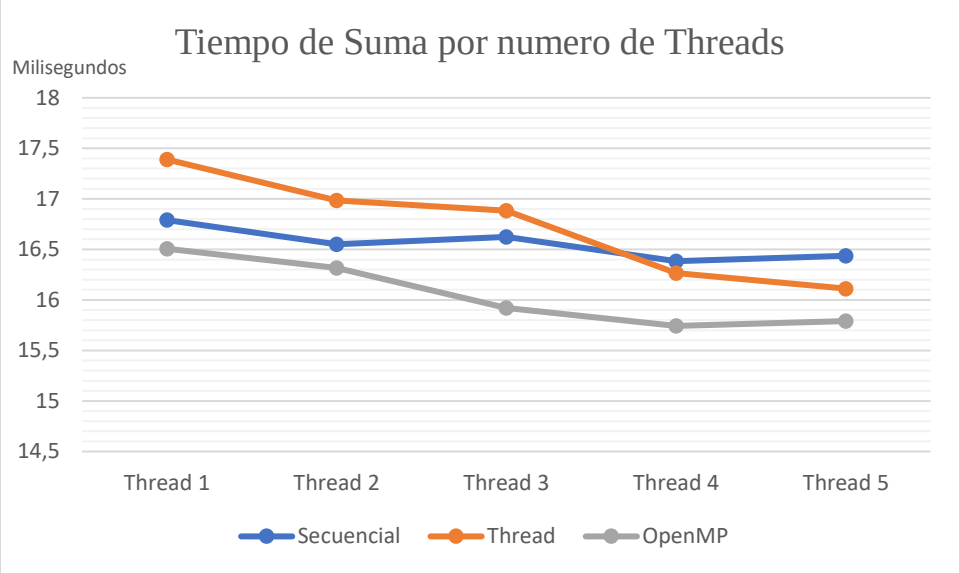

Figura 10. Tiempo de llenado en paralelo por OpenMp y de forma secuencial

6. Conclusión

Finalizando este taller se han completado los objetivos correspondientes a los dos módulos que nos pedían la creación y llenado de un arreglo con números aleatorios y sumar todos estos números de forma paralela utilizando OpenMP.

En conclusión, Utilizando OpenMP para las sumas en paralelo se puede apreciar una disminución en el tiempo del primer al tercer thread, pero después al aumentar los threads este tiempo se mantiene. Respecto al llenado en paralelo el tiempo es variable, pero se nota una media alrededor de los 16[ms] al realizar más pruebas de desempeño con diferentes parámetros en el total de elementos, el tiempo parecía no variar demasiado respecto a los threads, este tiempo podía bajar o subir, pero resaltaba una mayoría de resultados cerca de la media.

Al comparar las pruebas del Taller03 con el del Taller02 (Resultados y pruebas en el anexo) se aprecia que los resultados usando OpenMP versus la Librería Threads son más eficientes, pero esto no se puede confirmar por una errónea codificación de los objetivos del taller02, pero como se indicó al inicio de las pruebas se rehicieron estos del Taller02 y se incluyeron en este taller, pudiendo concluir que en el aspecto paralelo OpenMP es más eficiente que la librería Thread bajo los parámetros impuestos para las mediciones para los dos módulos.

Para tener una mejor compresión del funcionamiento y desempeño del programa bajo los parámetros impuestos para las medidas se recomienda un numero muy mayor a 5 pruebas como las realizadas para este taller.

7. Anexo

[1]: Promedio de los resultados del módulo de llenado

Llenado	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5
Secuencial	746,412	741,9874	741,093	740,9598	739,3942
Thread	745,794	740,2154	738,88	739,6424	736,053
OpenMP	741,133	730,9966	730,021	723,3326	721,4994

[2]: Promedio de los resultados del módulo de suma

Sumas	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5
Secuencial	16,7899	16,54964	16,62424	16,38332	16,43682
Thread	17,38974	16,98314	16,88416	16,26584	16,11196
OpenMP	16,50636	16,3178	15,92012	15,74248	15,78978

[3]: Resultados: Tiempo de Llenado paralelo con la librería thread

Prueba	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5
1	738,353	736,103	737,345	744,605	738,433
2	742,087	737,358	743,134	731,389	737,78
3	754,596	739,57	742,379	735,776	735,008
4	758,413	747,354	736,549	746,633	733,503
5	735,521	740,692	734,993	739,809	735,541

[4]: Resultados: Tiempo de suma paralelo con la librería Thread

Prueba	Thread 1	Thread 2	Thread 3	Thread 4	Thread 5

1	17,7571	17,6841	16,0542	15,9828	15,9901
2	17,0123	16,1133	16,1921	16,4866	16,1178
3	16,9234	16,1766	17,6053	16,7236	15,0091
4	17,5549	17,9081	16,7927	15,9254	16,7012
5	17,701	17,0336	17,7765	16,2108	16,7416

[5]: Resultados: Tiempo de llenado paralelo con OpenMP

Prueba	**Thread 1**	**Thread 2**	**Thread 3**	**Thread 4**	**Thread 5**
1	740,1	726,595	729,3	719,932	723,865
2	750,74	725,876	732,984	730,384	719,173
3	750,875	742,492	723,976	722,408	725,761
4	733,948	729,223	733,473	713,917	719,197
5	730,002	730,797	730,372	730,022	719,501

[6]: Resultados: Tiempo de suma paralelo con OpenMP

Prueba	**Thread 1**	**Thread 2**	**Thread 3**	**Thread 4**	**Thread 5**
1	17,9324	15,7816	15,6777	15,5592	15,9732
2	16,1213	16,6981	15,9188	16,0605	15,2932
3	15,946	15,6568	15,9563	15,5219	15,7176
4	16,9686	17,5593	16,1514	15,7561	15,9735
5	15,5635	15,8932	15,8964	15,8147	15,9914

[7]: Resultados y promedio: Tiempo de suma y llenado secuencial. Al ser 25 pruebas en total se tomo en cuenta todos estos resultados para medición secuencial, y se dividió en grupos por thread.

Prueba	**Llenado**	**Suma**
1.1	744,685	17,3444
1.2	739,235	16,201
1.3	745,417	16,4207
1.4	747,468	17,2372
1.5	755,255	16,7462
2.1	731,81	15,8481
2.2	751,673	16,0104
2.3	743,896	18,6546
2.4	736,593	16,4785
2.5	745,965	15,7566
3.1	742,424	18,2657
3.2	746,81	16,232
3.3	740,906	15,8397
3.4	742,916	16,7762
3.5	732,409	16,0076
4.1	733,243	15,9828

4.2	748,825	16,3477
4.3	745,422	16,7889
4.4	740,506	16,7277
4.5	736,803	16,0695
5.1	735,3	15,9276
5.2	736,424	15,4879
5.3	732,767	16,8283
5.4	752,682	17,5965
5.5	739,798	16,3438