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 3 secciones, la Introducción del Taller 1, 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.

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.

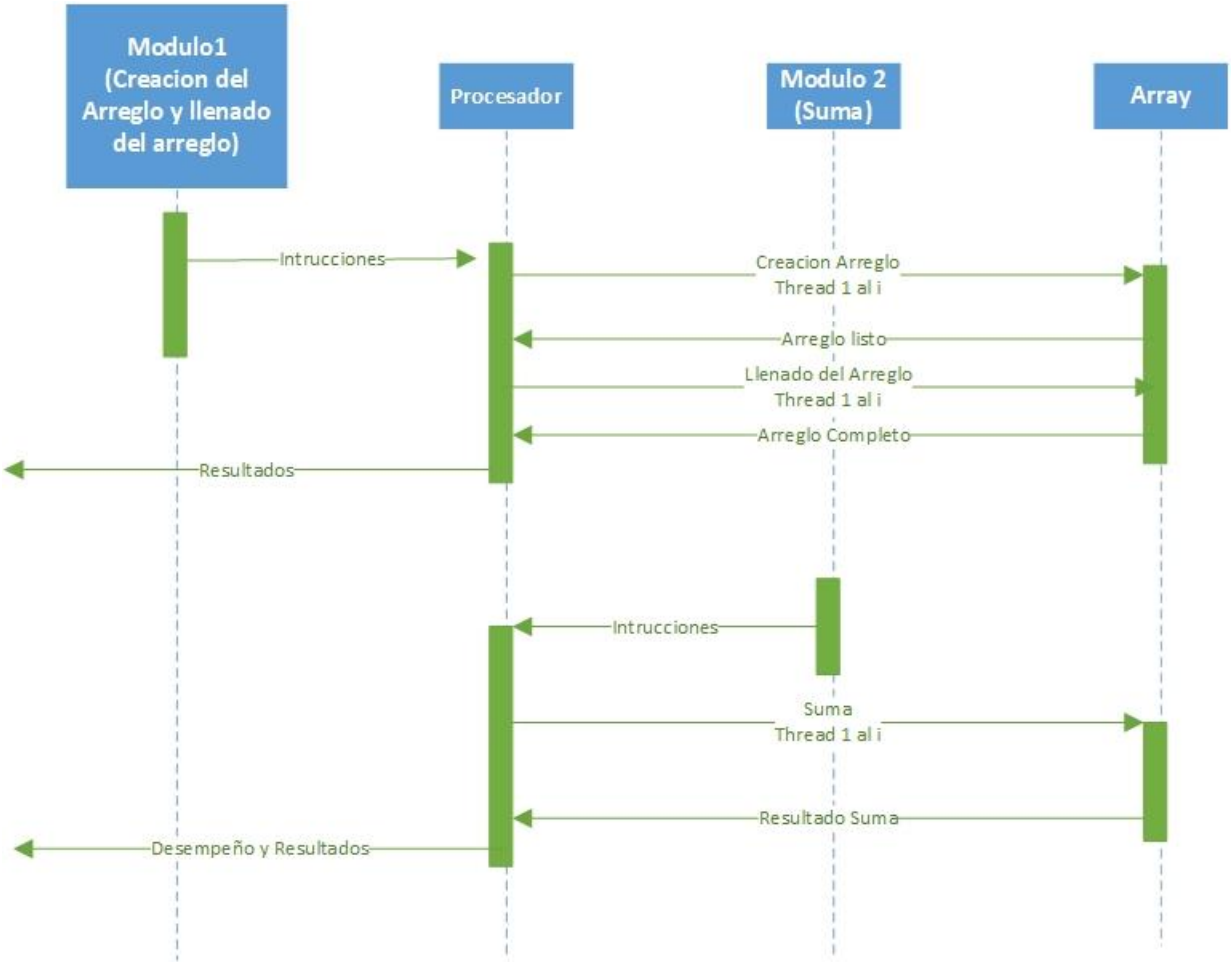

Figura 1. Diagrama de Secuencia solución general

3.2. Modulo 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.

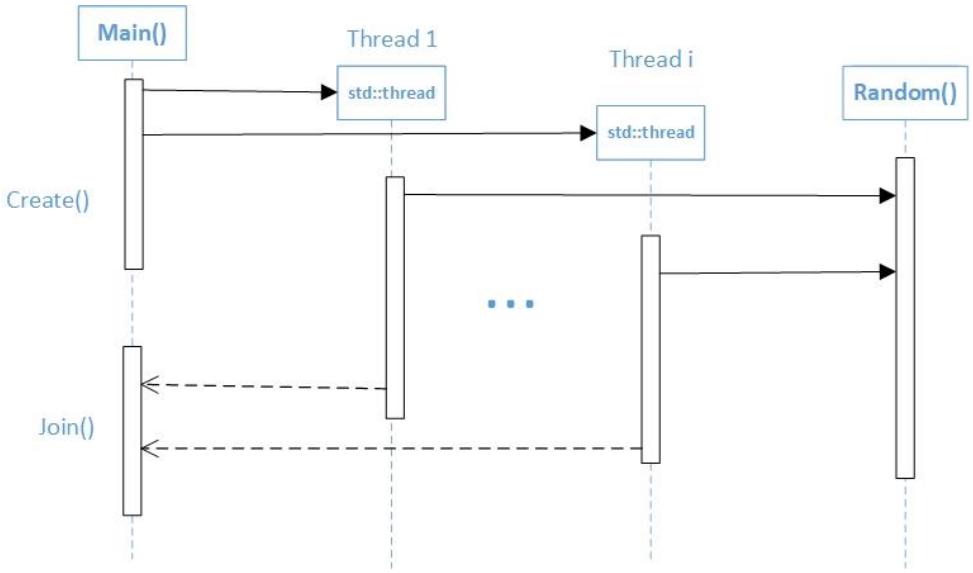

Figura 2. Funcionamiento de los threads para el llenado del arreglo

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

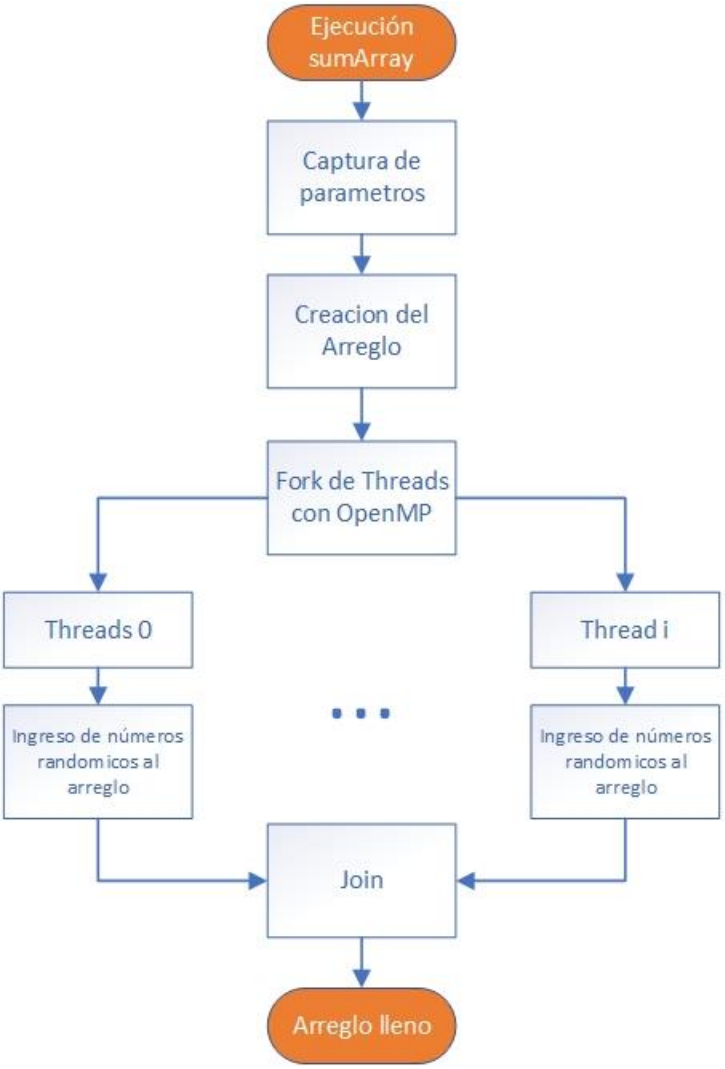

Figura 3. Creación y llenado del Arreglo

3.3. Modulo 2

En el modulo 2 se solicita sumar todos los elementos del arreglo creado en el modulo 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.

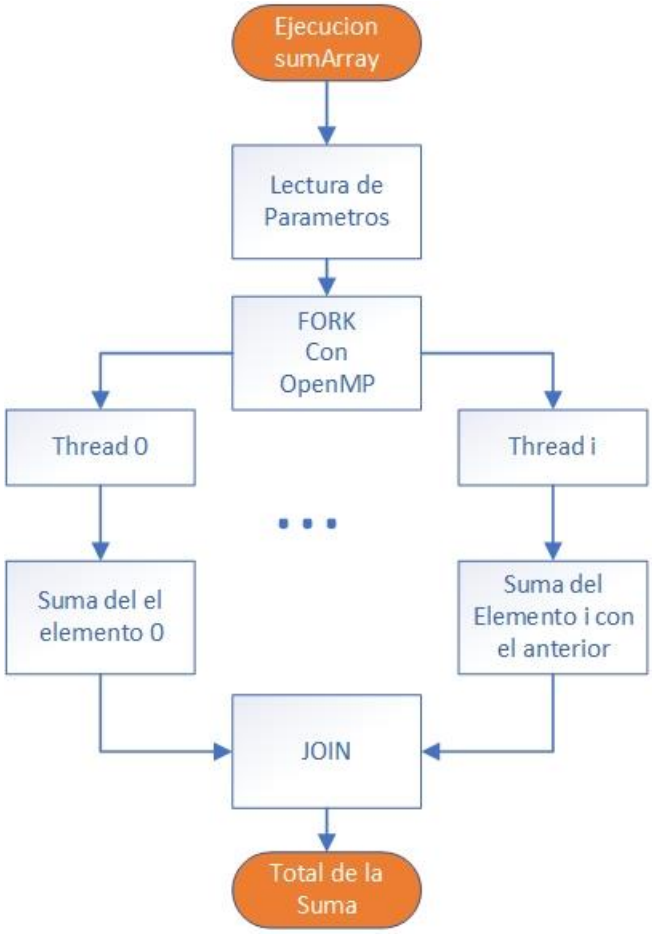

Figura 4. Modulo de suma de los elementos del arreglo

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.

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

```
suvin@fracaso:~/Talleres/U2/TSSO02$
suvin@fracaso:~/Talleres/U2/TSSO02$ ./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

suvin@fracaso:~/Talleres/U2/TSSO02$
```

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 100 -t 1 -l 1 -L 50
Total de Elementos : 100
Total de Hilos     : 1
====Llenado del arreglo con OpenMP====
Tiempo de llenado  :0.00769[ms]
===========Sumas con OpenMP===========
Suma total paralela: 2381
============Tiempos de Suma===========
Tiempo paralelo    :0.000717[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 1000000 -t 1 -l 1 -L 50000
Total de Elementos : 1000000
Total de Hilos     : 1
====Llenado del arreglo con OpenMP====
Tiempo de llenado  :24.1615[ms]
===========Sumas con OpenMP===========
Suma total paralela: 25014189225
============Tiempos de Suma===========
Tiempo paralelo    :0.65839[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.

```
suvin@fracaso:~/Talleres/U2/TSSOO2$ ./sumArray -N 72316123 -t 6 -l 343424 -L 2141325
Total de Elementos : 72316123
Total de Hilos     : 6
====Llenado del arreglo con OpenMP====
Tiempo de llenado  :1764.44[ms]
===========Sumas con OpenMP===========
Suma total paralela: 89844467160784
============Tiempos de Suma===========
Tiempo paralelo    :40.0692[ms]
suvin@fracaso:~/Talleres/U2/TSSOO2$
```

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

Los parámetros utilizados para estas pruebas serán treinta millones de elementos, el rango inferior será uno y el mayor cien mil y un thread el cual ira aumentando en uno en cada medición.

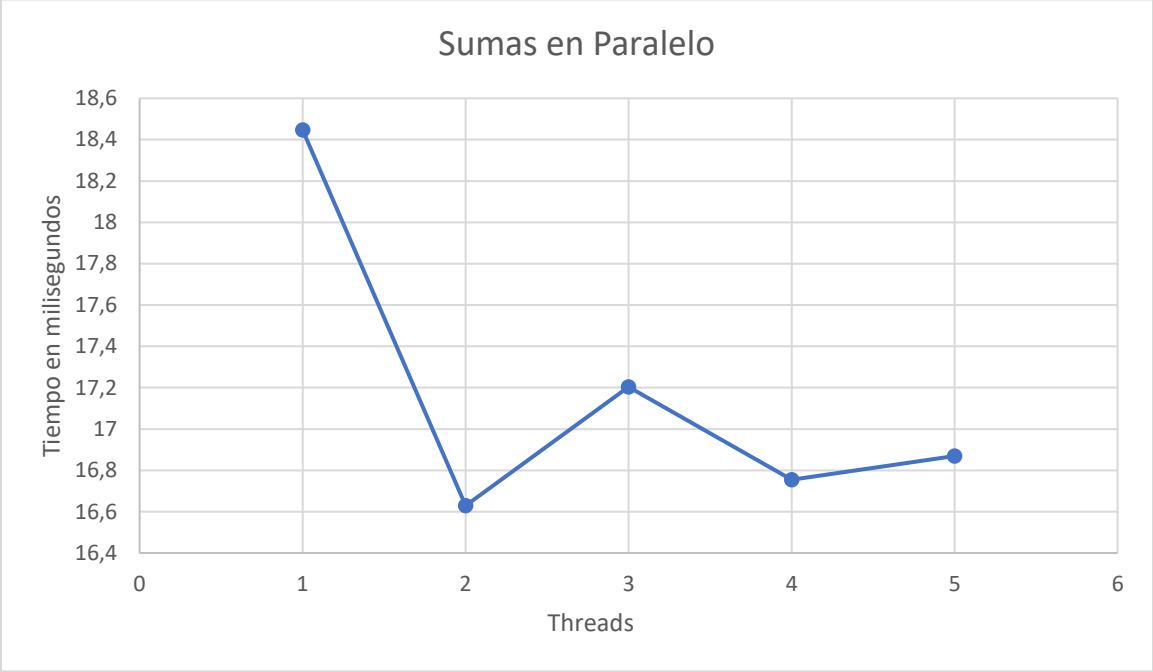

Figura 9. Tiempo de suma en paralelo por OpenMp

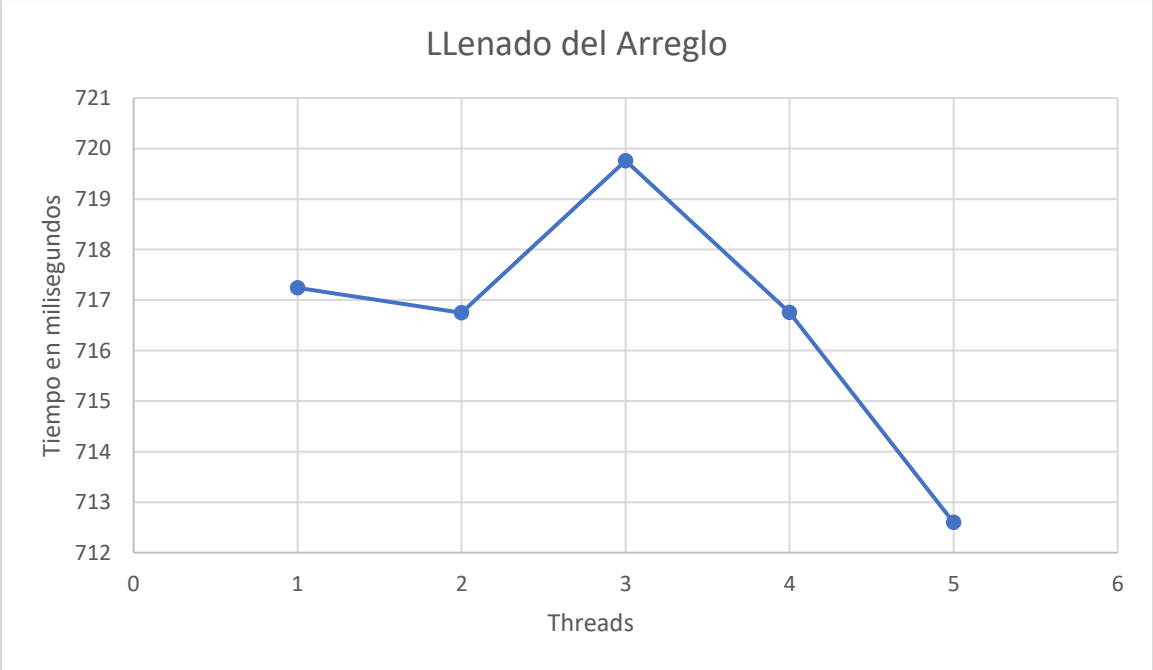

Figura 10. Tiempo de llenado en paralelo por OpenMp

6. Conclusión

Finalizando este taller se concluye completando los objetivos correspondientes a la creación de 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 segundo 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 17[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 mas eficientes, pero esto no se puede confirmar por una errónea codificación de los objetivos del taller02.

7. Anexo

[1]: Prueba 2 del Taller02

```
suvin@fracaso:~/Talleres/U2/TSSOO2$ ./sumArray -N 100 -t 1 -l 1 -L 50
Total de Elementos: 100
Total de Hilos  : 1
===Llenado del arreglo===
Tiempo de llenado :0[ms]
==========Sumas==========
Suma Serial:2469
Suma Paralela: 2469
=====Tiempos de Suma=====
Tiempo serial :0[ms]
Tiempo paralelo :0[ms]
suvin@fracaso:~/Talleres/U2/TSSOO2$
```

[2]: Prueba 3 del Taller02

```
suvin@fracaso:~/Talleres/U2/TSSOO2$ ./sumArray -N 1000000 -t 1 -l 1 -L 50000
Total de Elementos: 1000000
Total de Hilos  : 1
===Llenado del arreglo===
Tiempo de llenado :57[ms]
==========Sumas==========
Suma Serial:3518169526
Suma Paralela: 3518169526
=====Tiempos de Suma=====
Tiempo serial :21[ms]
Tiempo paralelo :21[ms]
suvin@fracaso:~/Talleres/U2/TSSOO2$
```

[3]: Prueba 4 del Taller02

```
suvin@fracaso:~/Talleres/U2/TSSOO2$ ./sumArray -N 72316123 -t 6 -l 343424 -L 2141325
Total de Elementos: 72316123
Total de Hilos  : 6
===Llenado del arreglo===
Tiempo de llenado :4345[ms]
==========Sumas==========
Suma Serial:3814067767
Suma Paralela: 3814067767
=====Tiempos de Suma=====
Tiempo serial :1602[ms]
Tiempo paralelo :9004[ms]
suvin@fracaso:~/Talleres/U2/TSSOO2$
```

[4]: Prueba de desempeño Taller02