

Taller 02

Taller de Sistemas Operativos
Escuela de Ingeniería Informática

Sebastián Lillo Núñez

sebastian.lillon@alumnos.uv.cl

Resumen. En este trabajo se realizan dos tipos de procesamiento de datos, enfocados en el llenado, de una estructura de dato del tipo array, y la suma de elementos aleatorios almacenados en un arreglo, el primero es de forma secuencial, es decir, trabajar de una iteración a la vez; la segunda es a través de threads o hilos de procesamiento, en donde se realizan 2 o mas iteraciones de forma paralela. El programa requiere de cuatro parámetros de entrada para poder ser ejecutado, cantidad de datos, numero de hilos, cota inferior del rango de los posibles numero aleatorios y su cota superior. A través de ambos métodos de procesamiento se realizar una comparación del desempeño de cada uno, reflejado en el tiempo de ejecución de cada procesamiento evidenciado en la sección de Resultados. Finalmente se concluye que el trabajo paralelo es más rápido en cuanto al tiempo de ejecución, lo cual es beneficiario para el procesamiento de enormes cantidades de datos.

1. Introducción

La programación paralela es un modelo en cual se presenta de forma concurrente, es decir, varios procesos trabajando en la solución de un problema con varios procesadores a la vez, con distintas caracterizas incluso entre sí. Este modelo de programación tiene características adaptivas frente al sistema donde se esté implementando, además es distribuida en cuanto a los procesadores se refiere con una infraestructura adecuada. Algunas de las necesidades que resuelve la programación paralela son, por ejemplo, los limitantes de una sola CPU (Central Processing Unit), resolver problemas con tiempos de ejecución no razonables, todo esto se debe a que las maquinas secuenciales están comenzando a no ser suficiente para el procesamiento de algunos programas de problemas complejos.

Considerando lo expuesto anteriormente, el objetivo de este informe es evidenciar el proceso de creación de un programa implementando en el lenguaje de programación C++, el cual se dividirá en dos módulos, uno que agrega elementos aleatorios del tipo `uint_32` en forma paralela a un arreglo; el segundo modulo deberá sumar el contenido del arreglo también de forma paralela. Además, se realizarán pruebas de desempeño para poder ver de forma clara el comportamiento del tiempo de ejecución de ambos módulos y los threads utilizados. Un thread es un proceso que a su vez genera un grupo de threads generando el paralelismo a la hora de la ejecución del programa [1]. Así, el documento cuenta distintas secciones: procedimiento, en donde se mostrarán los datos utilizados, metodología empleada y el diseño de implementación para la resolución del problema, una sección de resultados referente a lo anterior y una sección de conclusiones obtenidas tras el desarrollo del taller.

2. Procedimiento

2.1 Descripción de los datos

Los datos por utilizar en este taller provienen de la una función aleatoria que los generará, siendo todos ellos del tipo `uint_32`, es decir, que pueden estar entre 0 hasta $2^{32}-1$ como muestra la figura 1.

```
Arr = { 2 , 56 , 489 , 64 , 5 , 45684 , ... , 36 }
```

Fig. 1 muestra un ejemplo de un arreglo llenado de forma aleatoria con n elementos.

2.2 Forma de Uso

El programa debe ser ejecutado con parámetros de entrada, definiendo así la cantidad de datos que se van a utilizar, la cantidad de hilos con los que se va a paralelizar y el rango de valores que pueden tomar los valores aleatorios, los cuales, serán los datos que procesar, tal como lo muestra la figura 2.

```
./sumArray -N <nro> -t <nro> -l <nro> -L <nro> [-h]  
-N : tamaño del arreglo.  
-t : número de threads.  
-l : límite inferior rango aleatorio.  
-L : límite superior rango aleatorio.  
[-h] : muestra la ayuda de uso y termina.
```

Fig. 2 Muestra la forma de uso para ejecutar el programa.

2.3 Metodología

La metodología para realizar este taller esta dada por los siguientes puntos: 1) Declarar las variables necesarias para poder resolver el problema, esto no incluye variables temporales. 2) Implementar una función aleatoria que genere los números del tipo `uint_32` e ingresarlos al arreglo ya declarado. 3) Desarrollar una función que recorra cada elemento de un arreglo de tamaño n y a su vez que sume el contenido de este. 4) Evidenciar resultados en gráficos para comparar tiempos de ejecución.

2.4 Diseño

Para poder llevar a cabo esta implementación es necesario el uso de estructuras de datos del tipo lista como son los arreglos y los vectores, en donde el primero es de tamaño fijo de memoria definido en el código fuente del programa; y el segundo abarca un tamaño dinámico de memoria, lo cual nos da el beneficio de no conocer de manera inicial la cantidad de datos a procesar, también se puede definir que tipo de datos almacenara la lista [2].

El diseño de forma generalizada sin considerar el modelo de programación se puede evidenciar en la figura 2.

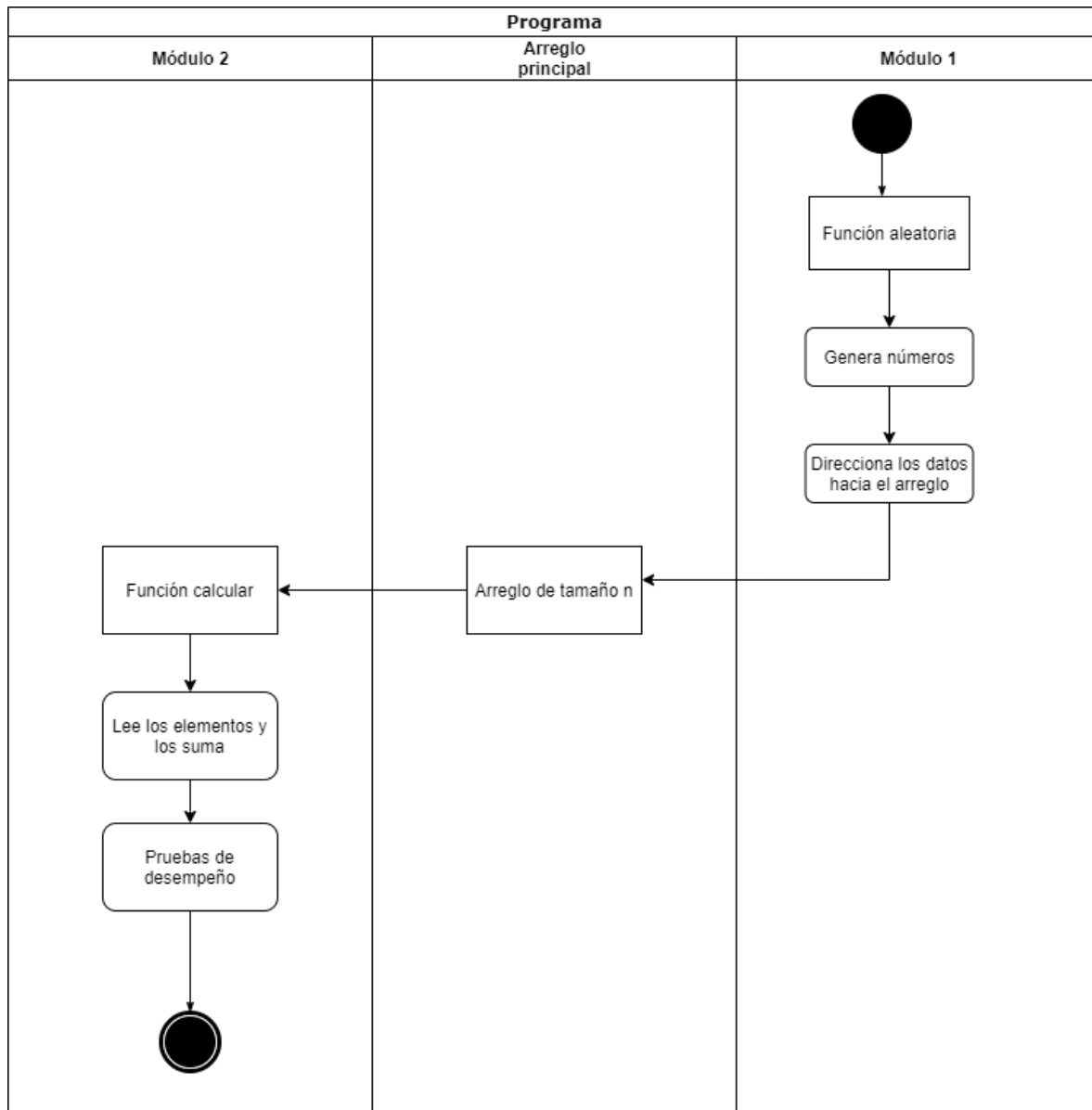


Fig. 3 muestra el proceso global de ejecución del problema.

2.4.1 Módulo 1

En primera instancia es pertinente declarar el arreglo principal donde se almacenarán los números aleatorios y a través de estructura repetitivas como los ciclos for o while se ingresarán los datos, el rango será dado al momento de ejecutar el programa, todo esto contemplando un modelo de programación paralela, siendo el thread principal el ingreso de datos, con esto se derivan los demás threads para realizar la operación fiel al modelo (figura 3).

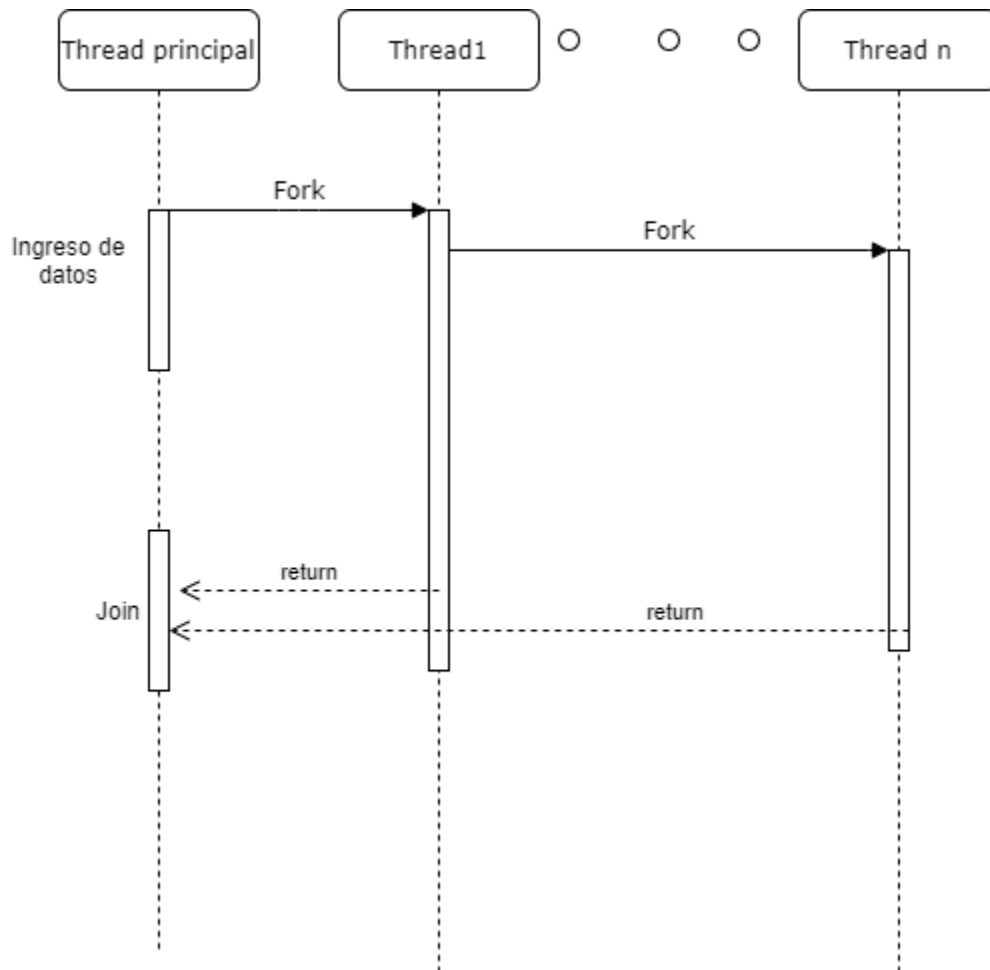


Fig. 4 muestra los procesos a utilizar en la implementación del módulo 1.

Debido a que el tamaño del arreglo no es fijo es importante destacar que la estructura de dato se debe ajustar a esa característica del problema, considerando esto mismo se plantea el uso de un grupo de threads indicados en la invocación del programa para realizar la operación, así recorrer el arreglo cada uno por su cuenta, cada proceso terminará una vez que no encuentre una posición vacía. Se condicionará a partir del tamaño del arreglo si es posible particionar más de 2 veces.

2.4.2 Módulo 2

Para esta instancia del problema se implementará una función que recorra el arreglo y además sume cada posición de este, utilizando la misma idea que en el modulo uno, con dos procesos o más, dependiendo de la cantidad de hilos solicitada. La condición de termino será dependiente de la cantidad elementos que posea el arreglo.

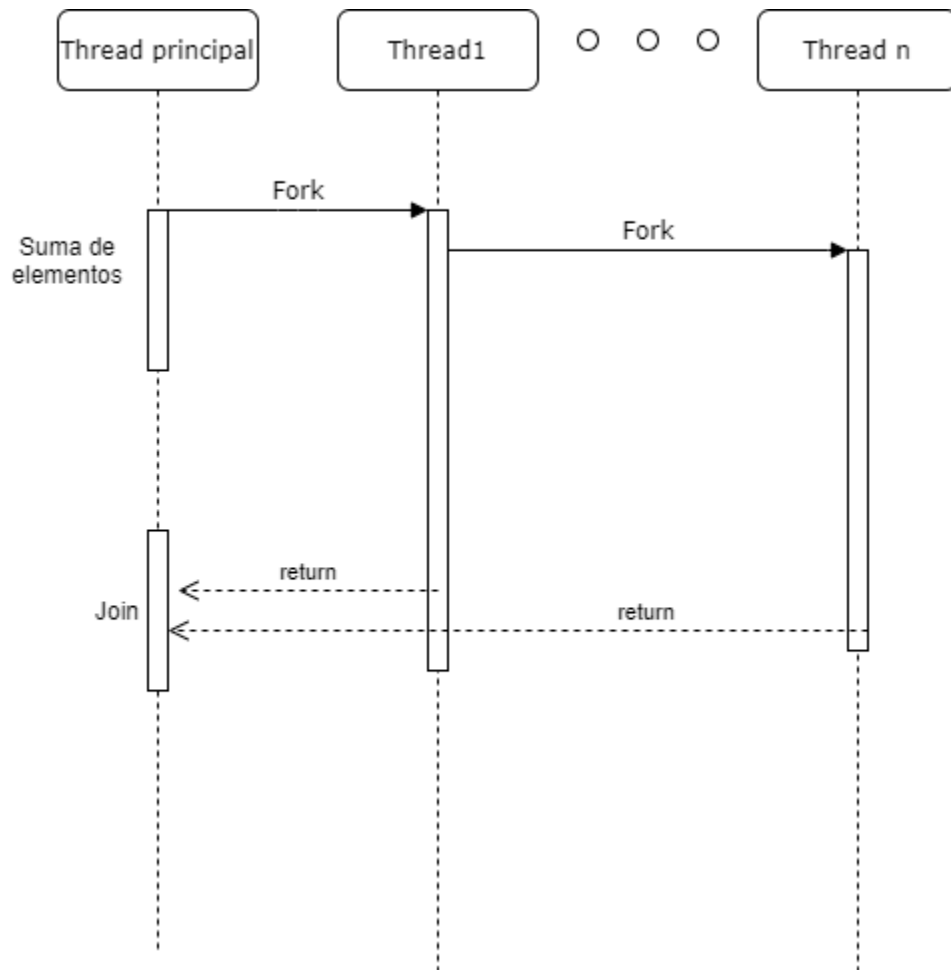


Fig. 5 muestra los procesos a utilizar en la implementación del módulo 2.

3. Resultados

Los resultados se obtuvieron a través de la librería `chrono`, para poder visualizar los resultados en función del tiempo de ejecución que tardó cada módulo tanto en su forma de procesamiento secuencial, como en el paralelo y también el desempeño basado en la razón del tiempo de ejecución serial sobre el tiempo de ejecución paralela. Además de esto se realizaron pruebas para corroborar el desempeño del modelo de programación, variando la cantidad de hilos en cada ejecución con la misma cantidad de datos.

```

sebastian@tallerso:~/TSS00-taller02/TSS00-taller02$ ./sumArray -N 100000000 -t 2 -l 10 -L 50
Elementos          : 100000000
Threads            : 2
Limite Inferior     : 10
Limite Superior     : 50
Suma en Paralelo    : 19217
Suma en Serie       : 19217
Tiempo Total Llenado Serial : 3312 ms
Tiempo Total Suma Serial   : 155 ms
Tiempo Total Llenado Paralelo : 1932 ms
Tiempo Total Suma Paralela  : 122 ms
Desempeño Serial         : 1.71429
Desempeño Paralelo       : 1.27049
  
```

Fig. 6 muestra una salida con 100000000 datos y dos hilos.

Cantidad de datos: 100000000				
Cantidad de threads	Tiempo de Llenado secuencial [ms]	Tiempo de Suma secuencial [ms]	Tiempo de Llenado Paralelo [ms]	Tiempo de Suma Paralelo [ms]
2	3253	180	1880	119
4	3248	175	754	65
6	3342	188	287	61
8	3298	181	162	31
10	3301	179	84	12

Tabla 1 muestra un promedio de 5 ejecuciones con las distintas cantidades de hilos.

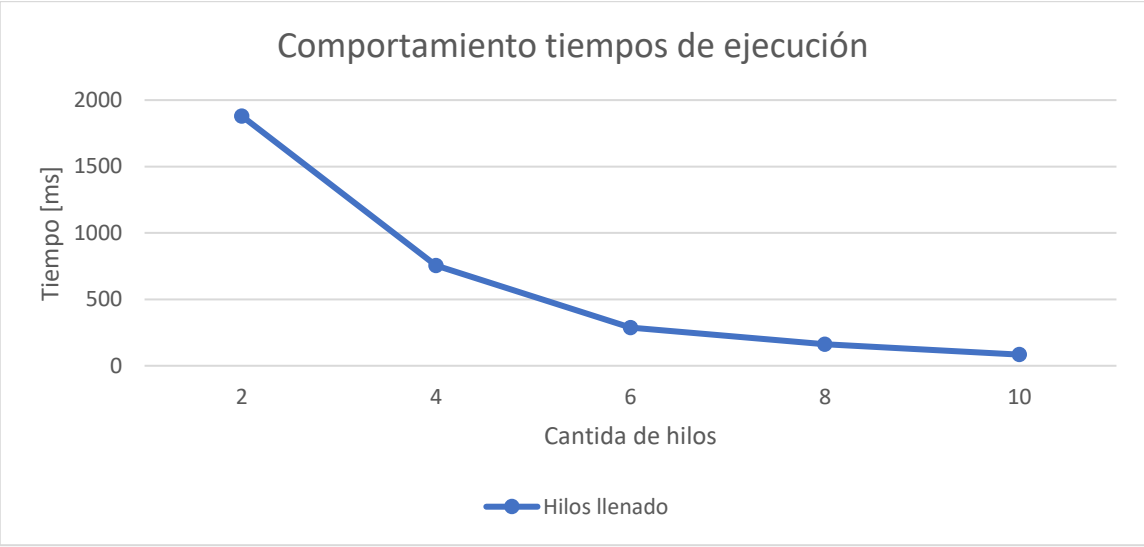


Gráfico 1 muestra el comportamiento de los tiempos de ejecución en el módulo de llenado según la cantidad de hilos.

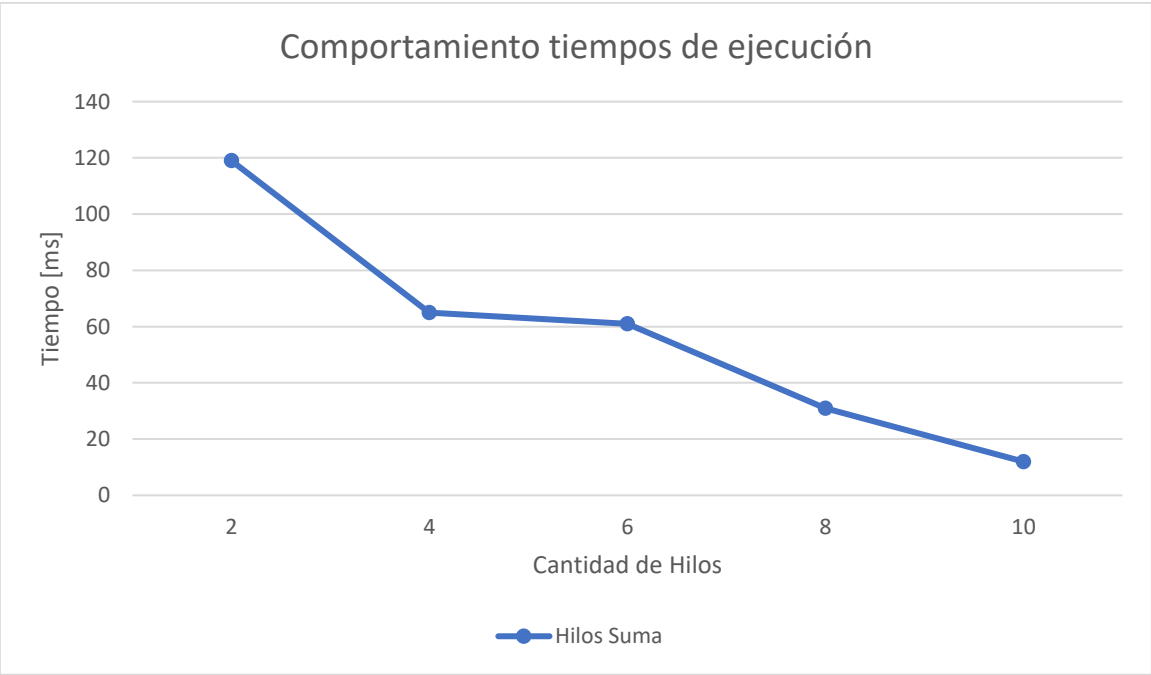


Gráfico 2 muestra el comportamiento de los tiempos de ejecución en el módulo de suma según la cantidad de hilos.

4. Conclusiones

En este trabajo se realizó el procesamiento de datos aleatorios almacenados en estructuras de datos de tipo lista, como vectores y arreglos, en donde muestra con evidencia estadística básica (media) el comportamiento del tiempo de ejecución de dicho procesamiento en dos módulos distintos, uno trabajando de manera secuencial y el otro de forma paralela. Se pudo evidenciar que el método donde se implementa el modelo de programación paralela presenta menores tiempos de ejecución a medida que se aumentan la cantidad de tareas a realizar en forma paralela, lo que nos evidencia que este diseño de implementación permite agilizar el procesamiento de datos para grandes cantidades de datos, todo esto sujeto a el equipo de hardware que se posea, siendo esto el límite de lo que puede ofrecer esta herramienta.

5. Referencias

[1] `cplusplus.com, std::thread`

C++ official website.

[2] Arrays, arreglos o vectores en C++. Uso, declaración y sintaxis de los vectores en C++ Available:

<https://www.programarya.com/Cursos/C++/Estructuras-de-Datos/Arreglos-o-Vectores>