Reporte Técnico Taller 02

Taller de Sistemas Operativos
Escuela de Ingeniería Informática
Ignacio Miranda Yáñez

Ignacio.miranda@alumnos.uv.cl

1 Introducción

Este trabajo consistirá en la demostración a nivel de programación en **C ++**, de la resolución de un problema que se debe resolver en forma de programación paralela, por lo tanto el problema puede ser resuelto de otras formas como por ejemplo la secuencial, pero la idea principal de este taller es aprender sobre la programación paralela, en que consiste, a nivel teórico y práctico y analizar los resultados, beneficios que trae y sus falencias, por lo tanto se presentara un conjunto de pruebas, resultados y un análisis de estos, también se mostrara un diseño de cómo afrontar este problema y un conjunto de información a nivel teórico sobre la programación paralela y otros temas que nos ayudaran para llegar a la solución del problema.

2 Conjunto Teórico

En esta sección se presentarán un conjunto de conceptos con información y diagramas para entender que aspectos se utilizaran para que la programación paralela sea posible

2.1 Modelos De Programacion Paralela

En este trabajo en particular se habla de llegar a una solución ocupando la programación paralela, pero al momento de buscar sobre esta, nos encontramos que existen varios modelos que se basan en la programación paralela algunos de los cuales son:

. - **Posix Threads**: **POSIX** significa Portable Operating System Interface **(for Unix)**. Es un estándar orientado a facilitar la creación de aplicaciones 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**.

. - **MPI**: Es un estándar de programación en paralelo mediante paso de mensajes que permite crear programas portables y eficientes, es una biblioteca que incluye interfaces para **FORTRAN, C y C++.**
Define varias formas de comunicación lo que permite programar de manera natural cualquier algoritmo en paralelo

. - **PVM**: La máquina paralela virtual es una máquina que no existe, pero un API apropiado que permite programar como si existiese. El modelo abstracto que nos permite usar el API de la **PVM** consiste en una máquina multiprocesador completamente escalable (es decir, que podemos aumentar y disminuir el número de procesadores en caliente). Para ello, nos va a ocultar la red que estemos empleando para conectar nuestras máquinas, así como las máquinas de la red y sus características específicas

. - **Intel TBB (Intel Threading Building Blocks)**: Modelo de programación paralela basado en rutinas que utilizan hilos.
Provee una serie de plantillas, tipos de datos y algoritmos. **TBB** está pensado de cara al rendimiento, por lo que es compatible con otras bibliotecas y paquetes.

Para La resolución de este trabajo, no se ocupará un conjunto de estos modelos, sino que en específico se ocupará el modelo **Posix Threads** y ahora se indicará por medio de diagramas como funciona este

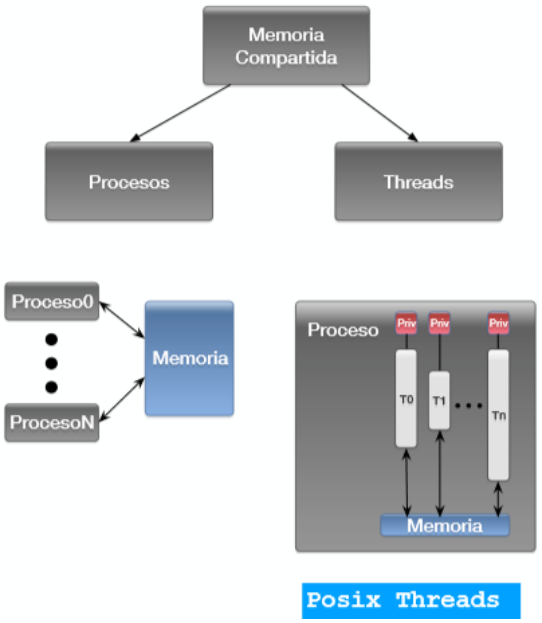

Figura 1. Posix Threads

2.2 Ventajas y Desventajas

Ventajas	Desventajas
Resuelve problemas que no se pueden resolver en un tiempo razonable	Mayor consumo de energía
Permite ejecutar problemas de un orden y complejidad mayor	Dificultad para lograr una buena sincronización y comunicación entre las tareas
Permite ejecutar código de manera más rápida (aceleración)	Retardos ocasionados por comunicación ente tareas
Permite la ejecución de varias instrucciones en simultáneo	Número de componentes usados es directamente proporcional a los fallos potenciales
Permite dividir una tarea en partes independientes	

Tabla 1. Ventajas y Desventajas

2.3 Hilos y Sincronización

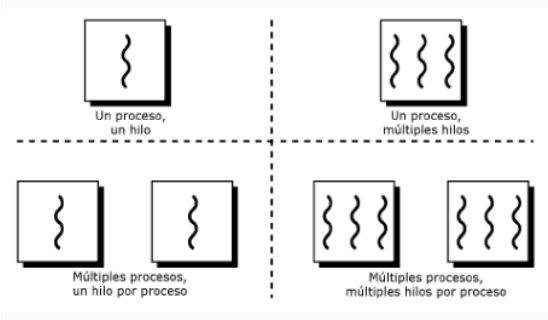

Figura 2. Hilos

Un proceso pesado padre puede convertirse en varios procesos livianos hijos, ejecutados de manera concurrente. Cada uno de estos procesos livianos se conoce como hilo. Estos se comunican entre ellos a través de la memoria global.

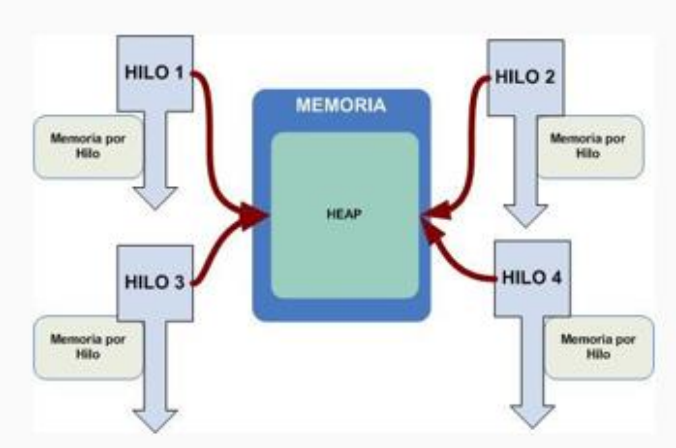

Figura 3. Sincronización

Los programas en paralelo necesitan la coordinación de procesos e hilos, para que haya una ejecución correcta. Los métodos de coordinación y sincronización en la programación paralela están fuertemente asociados a la manera en que los procesos o hilos intercambian información, y esto depende de cómo está organizada la memoria en el hardware.

2.4 Patrones De Diseño Paralelo

Se mostrarán un conjunto de patrones, no se mostrarán absolutamente todos ya que son varios en existencia y el que se utiliza en este trabajo será el patrón **Fork-Join**

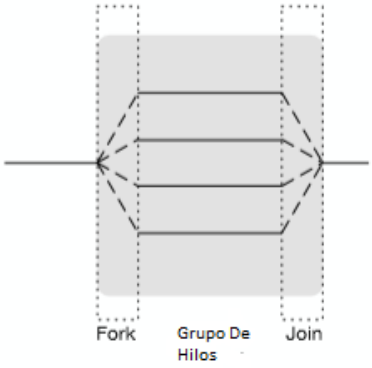

Figura 4. Fork-Join

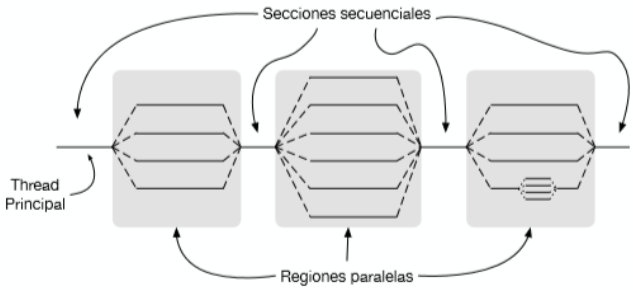

Figura 5. Fork-Join

Fue diseñado para la ejecución de tareas que pueden dividirse en otras subtareas más pequeñas, ejecutándose estas en paralelo y combinando posteriormente sus resultados para obtener el resultado de la tarea única. Las subtareas deberán ser independientes unas de otras, y no contendrán estado.

Realiza la paralelización de tareas de forma recursiva, aplicando el principio Divide y Vencerás. Existirá un umbral bajo el cual una tarea será indivisible, definido por el tamaño de la misma. Una vez el tamaño de las subtareas llegue al umbral, el rendimiento de estas disminuirá en caso de seguir dividiéndolas.

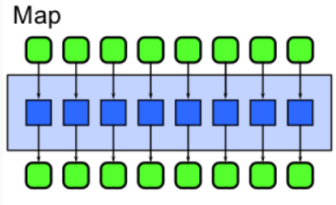

Figura 6. Map

Es un patrón que replica una función sobre todos los elementos de un conjunto de entrada. La función que está siendo replicada se llama función elemental, dada que la misma se aplica a una colección real de datos

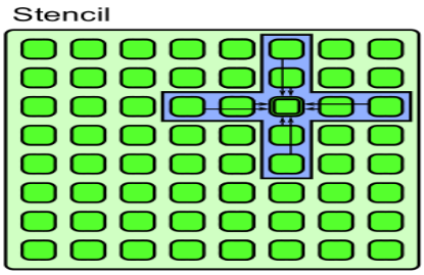

Figura 7. Stencil

Stencil es una generalización del patrón de **Map**, en el cual una función elemental tiene acceso no solo a un elemento del conjunto de entrada sino también a un conjunto de "vecinos".
Como la estructura de datos no es infinita se deben tener en cuenta el manejo de excepciones para los bordes de la misma

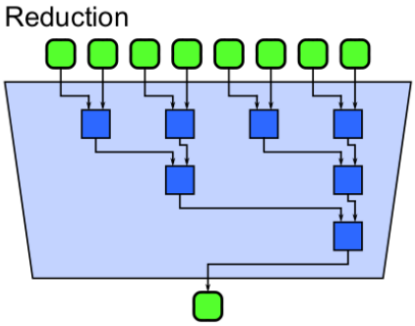

Figura 8. Reduction

Una reducción combina cada elemento de una colección en uno solo utilizado una función asociativa conocida como función combinatoria. Como es asociativa las tareas se pueden distribuir de muchas maneras y si la función resultara ser también conmutativa el número de posibilidades aumentaría aún más.

3 Descripción Del Problema

Específicamente se deben crear dos módulos, uno de estos módulos debe llenar un arreglo con números enteros de forma randomica del tipo **uint32_t** en forma paralela y el otro modulo vaya sumando el contenido del arreglo del primer módulo también de forma paralela, el problema también consiste en la entrada de un conjunto de parámetros como el tamaño del arreglo, numero de threads a utilizar y el límite inferior y superior del rango del conjunto de números aleatorios que entraran en el arreglo descripción

Campo	**Descripción**	**Ejemplo**
-N	Tamaño Del arreglo	0 – 1000000 enteros
-t	Numero de threads	0 – 16
-l	Límite inferior del rango aleatorio	10
-L	Límite superior del rango aleatorio	60
[-h]	Comando de ayuda para mostrar el uso del programa	Uso: ./sumArray -N <nro> -t <nro> -l <nro> -L <nro> [-h] Descripción: -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

Tabla 2. Forma de uso y ejemplo de parámetros

Arreglo[] = Arreglo dinámico que se llena con números aleatorios indicado por el parámetro -N

ArregloSuma[] = Arreglo dinámico que se llena con la suma de los números aleatorios del arreglo Arreglo[] y su tamaño es indicado por el parámetro -N

Llenado() = función que llena el arreglo ArregloRandom de números aleatorios y se basa en la concepción thread safe y se indica por los parámetros -l y -L

Para medir el tiempo de ejecución del programa se utilizara el método std::chrono que está en la biblioteca $\langle chrono \rangle$

4 Diseño De La Solución

Primero que nada, para la realización de este diseño se analizó los requerimientos que se pedían y las variables que se iban a utilizar como también el formato de salida, entonces lo que se pide es tomar dos módulos uno llenara un arreglo dinámico con números randomicos y el otro modulo sumara los números y los dejara en otro arreglo, también el sistema general del código tiene variables dinámicas que se indican por parámetros vistos anteriormente

4.1 Módulo 1 (Llenado Del Arreglo)

En este proceso se indica el arreglo dinámico llamado **Arreglo_random[]** y este mismo se dividirá según los hilos que se hayan introducido por medio de los parámetros, cada hilo sabe cuál es la parte que le toca del arreglo por medio de un inicio **(beginIdx)** y un final **(endIdx)** indicados por lo que en esos rangos debe llenar su conjunto de números(ver Figura 9)[1], esto se realiza en forma paralela con el módulo 2 que se mostrara en la próxima etapa

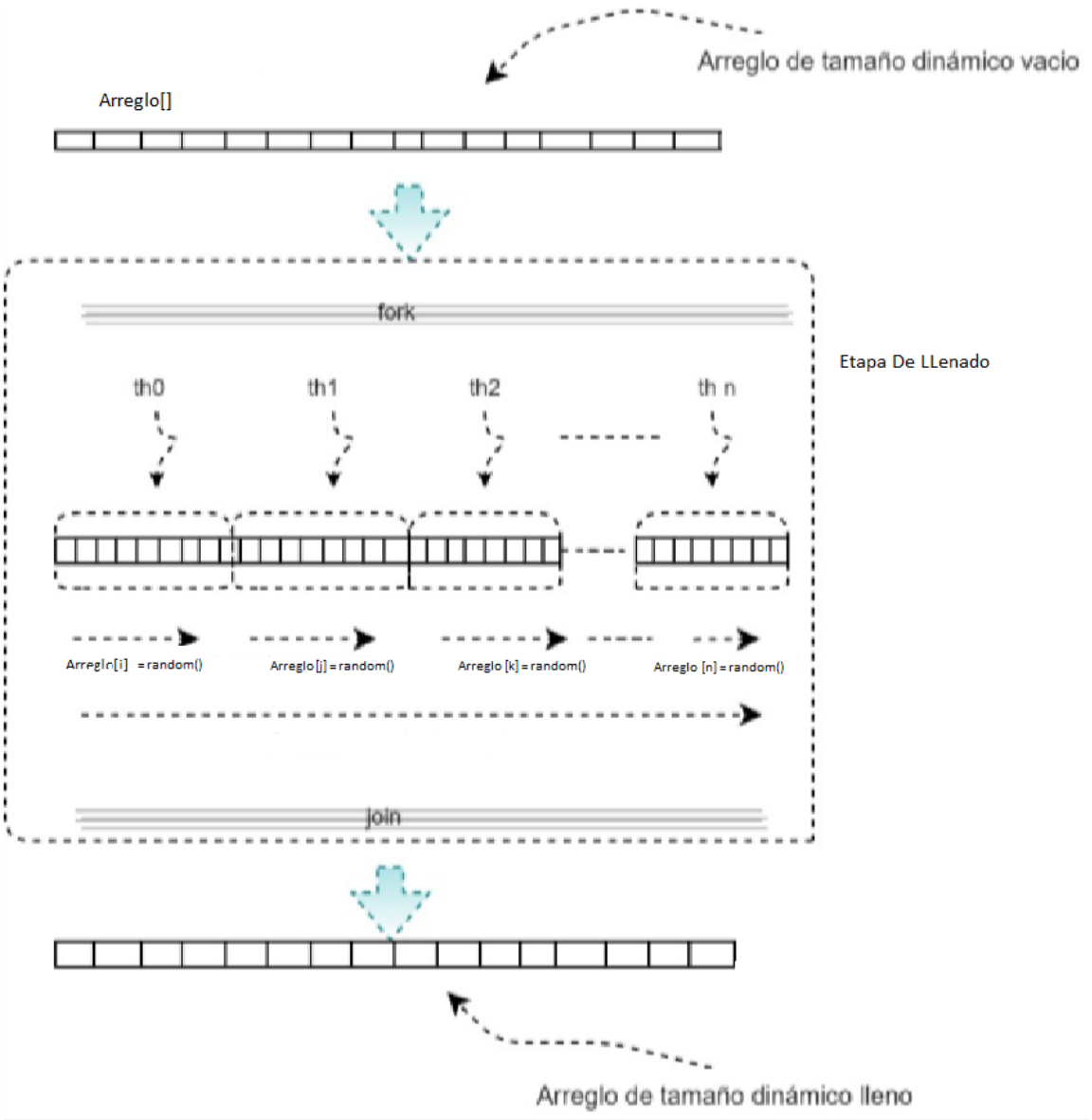

Figura 9. Modulo 1

4.2 Módulo 2 (Suma Del Arreglo)

En este módulo se toma el arreglo ya llenado anteriormente y como se realizó en el módulo anterior se divide el arreglo dependiendo de los hilos que se hayan indicado, cada hilo le corresponde una porción de arreglo y después de esto, se indican las sumas parciales que indicaron cada hilo, al final de todo el módulo, este conjunto de sumas se suma en total valga la redundancia

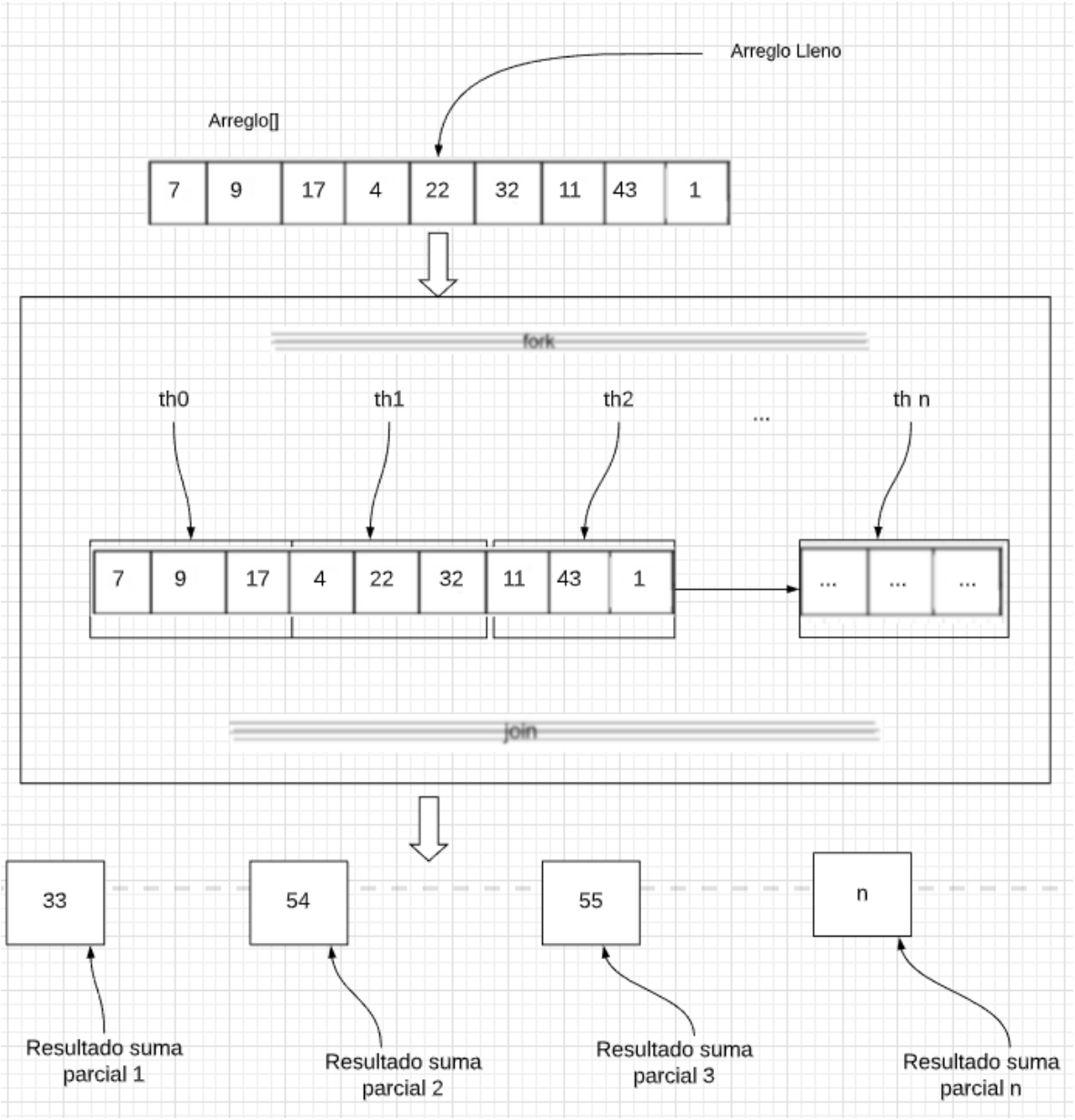

Figura 10. Suma del arreglo

5 Proceso De Ejecución

En esta etapa se indicará el proceso que se debe realizar para clonar el repositorio indicado y ejecutar el script, se ingreso a una carpeta vacía y se comprobó con el comando **ls-l** que no había archivos, después de eso se clono el repositorio correspondiente

```
ignacio@ignacio:~/tso02/U2$ cd pruebatarea02/
ignacio@ignacio:~/tso02/U2/pruebatarea02$ ls
ignacio@ignacio:~/tso02/U2/pruebatarea02$ ls -l
total 0
ignacio@ignacio:~/tso02/U2/pruebatarea02$ git clone https://github.com/Ignacio-my/TSSOO-taller02.git
Cloning into 'TSSOO-taller02'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 25 (delta 2), reused 10 (delta 0), pack-reused 0
Unpacking objects: 100% (25/25), done.
```

Clonado el repositorio se revisan si todos los archivos están en la carpeta clonada

```
ignacio@ignacio:~/tso02/U2/pruebatarea02$ ls -l
total 4
drwxrwxr-x 4 ignacio ignacio 4096 Jul 16 01:28 TSSOO-taller02
ignacio@ignacio:~/tso02/U2/pruebatarea02$ cd TSSOO-taller02/
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ ls
Makefile  README.md  src
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ cd src/
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02/src$ ls
Makefile  include  main.cc  objs
```

Ya revisadas las carpetas, se procede a hacer un **make clean** correspondiente y un **make** para compilar el script, este se compila y se creara en una carpeta anterior la cual es **TSSOO-taller02**, y el script estaría listo para su uso, este ocupara los parámetros ya mencionados

```
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02/src$ make clean
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02/src$ ls
Makefile  include  main.cc  objs
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02/src$ make
g++ -c -o objs/main.o main.cc -std=c++14 -Wall  -I../include -I. -I./include
g++ -o ../sumArray objs/main.o -std=c++14 -Wall  -lpthread
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02/src$ ls
Makefile  include  main.cc  objs
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02/src$ cd ..
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ ls
Makefile  README.md  src  sumArray
```

6 Resultados

Conjunto De resultados de la demostración de 4 pruebas con el mismo parámetro el cual sería 4 hilos, 1000000 de elementos y un rango de números aleatorios de 10 a 50, por lo que se indica la suma siempre varia pero no de gran manera, y prácticamente en los resultados del tiempo total hay una diferencia mínima, y con respecto al tiempo de llenado y sumado, el de llenado se demorara generalmente mas que el de sumado

```
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ ./sumArray -N 1000000 -t 4 -l 10 -L 50
Conjunto de Numeros Del Arreglo: 1000000
Numero de Hilos  : 4
Limite Inferior Rango Aleatorio: 10
Limite Superior Rango Aleatorio: 50
Suma Total en Paralelo: 30005363
TiempoLLenadoParalelo:3[ms]
TiempoSumaParalelo:2[ms]
TiempoEjecucionTotal:5[ms]
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ ./sumArray -N 1000000 -t 4 -l 10 -L 50
Conjunto de Numeros Del Arreglo: 1000000
Numero de Hilos  : 4
Limite Inferior Rango Aleatorio: 10
Limite Superior Rango Aleatorio: 50
Suma Total en Paralelo: 30003510
TiempoLLenadoParalelo:3[ms]
TiempoSumaParalelo:2[ms]
TiempoEjecucionTotal:5[ms]
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ ./sumArray -N 1000000 -t 4 -l 10 -L 50
Conjunto de Numeros Del Arreglo: 1000000
Numero de Hilos  : 4
Limite Inferior Rango Aleatorio: 10
Limite Superior Rango Aleatorio: 50
Suma Total en Paralelo: 29994917
TiempoLLenadoParalelo:4[ms]
TiempoSumaParalelo:2[ms]
TiempoEjecucionTotal:6[ms]
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ ./sumArray -N 1000000 -t 4 -l 10 -L 50
Conjunto de Numeros Del Arreglo: 1000000
Numero de Hilos  : 4
Limite Inferior Rango Aleatorio: 10
Limite Superior Rango Aleatorio: 50
Suma Total en Paralelo: 30010878
TiempoLLenadoParalelo:3[ms]
TiempoSumaParalelo:2[ms]
TiempoEjecucionTotal:5[ms]
```

Conjunto De resultados de la demostración de 4 pruebas con el mismo parámetro el cual sería 2 hilos, 1000000 de elementos y un rango de números aleatorios de 10 a 50, por lo que se demuestra hay una demora promedio total del doble que en las pruebas anteriores

```
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ ./sumArray -N 1000000 -t 2 -l 10 -L 50
Conjunto de Numeros Del Arreglo: 1000000
Numero de Hilos  : 2
Limite Inferior Rango Aleatorio: 10
Limite Superior Rango Aleatorio: 50
Suma Total en Paralelo: 29987791
TiempoLLenadoParalelo:6[ms]
TiempoSumaParalelo:6[ms]
TiempoEjecucionTotal:12[ms]
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ ./sumArray -N 1000000 -t 2 -l 10 -L 50
Conjunto de Numeros Del Arreglo: 1000000
Numero de Hilos  : 2
Limite Inferior Rango Aleatorio: 10
Limite Superior Rango Aleatorio: 50
Suma Total en Paralelo: 30008423
TiempoLLenadoParalelo:6[ms]
TiempoSumaParalelo:5[ms]
TiempoEjecucionTotal:11[ms]
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ ./sumArray -N 1000000 -t 2 -l 10 -L 50
Conjunto de Numeros Del Arreglo: 1000000
Numero de Hilos  : 2
Limite Inferior Rango Aleatorio: 10
Limite Superior Rango Aleatorio: 50
Suma Total en Paralelo: 30007745
TiempoLLenadoParalelo:6[ms]
TiempoSumaParalelo:4[ms]
TiempoEjecucionTotal:10[ms]
ignacio@ignacio:~/tso02/U2/pruebatarea02/TSSOO-taller02$ ./sumArray -N 1000000 -t 2 -l 10 -L 50
Conjunto de Numeros Del Arreglo: 1000000
Numero de Hilos  : 2
Limite Inferior Rango Aleatorio: 10
Limite Superior Rango Aleatorio: 50
Suma Total en Paralelo: 30015141
TiempoLLenadoParalelo:6[ms]
TiempoSumaParalelo:4[ms]
TiempoEjecucionTotal:10[ms]
```

7 Conclusión

Este trabajo se basó en la investigación en forma teórica y práctica de lo que tiene que ver con todo el concepto de la programación paralela, se dice todo el concepto ya que se estudió sobre un conjunto de modelos, patrones de diseños y las ventajas y desventajas que hay en el uso de la programación paralela, con respecto a la práctica se analizó el problema con detalle para luego indicar un diseño de la solución y se indicaron variables, métodos y ejemplos de variables a utilizar, ya que este script se basa meramente en la entrada de parámetros estos se explicaron y se indicó su uso correspondiente, también se indicaron un conjunto de resultados de las pruebas que se hicieron y se demostró a nivel de tiempo los posibles ambientes en donde se introducían diferentes parámetros para analizar el comportamiento del programa. Por lo tanto, en pocas palabras se aprendió que a nivel general la programación paralela es más eficiente que la secuencial en aspectos como la eficiencia del tiempo y recursos, pero esta debe ser bien trabajada con respecto a si se ocupan bien los hilos y sus parámetros de entrada y también es importante un buen nivel de programación

8 Referencias

[1] Astudillo, G. (2020). Etapa De Llenado. [Diagrama]. Recuperado de fillArrayParallel.pdf