Tecnológico de Monterrey

INTRODUCCIÓN

La computación paralela como aprendimos es una técnica que permite a los desarrolladores mejorar el rendimiento y velocidad de una tarea o programa especifica mediante la utilización de varios procesadores o núcleos de una computadora. Esta técnica se ha vuelto cada vez más importante debido al aumento en la potencia de procesamiento en las computadoras, el exponencial crecimiento de los datos a manejar y el mejoramiento de los algoritmos capaces de procesar estos algoritmos de una manera mas eficiente.

En el siguiente trabajo resalto la importancia de la técnica de la programación de hilos de trabajo, la cual es una técnica clave para la computación paralela, la cual permite la ejecución simultanea de múltiples tareas, en este caso de operaciones matemáticas, dentro de una misma ejecución. El conocer como se utilizan los hilos es esencial para aprovechar la capacidad de procesamiento de las computadoras modernas y como es que se utilizan en las implementaciones de nube. Al conocer sobre la implementación de hilos podremos mejorar el rendimiento y la escalabilidad de nuestras aplicaciones ya que los servicios mas grandes de servicios en la nube nos ofrecen herramientas y servicios que permiten crear aplicaciones paralelas.

Por ejemplo, GCP nos ofrece servicios como Kubernetes Engine el cual permite crear y escalar contenedores o aplicaciones. Permitiendo la ejecución en varios nodos de un clúster, lo que nos ayuda a mejorar el rendimiento de nuestras aplicaciones.

En general, la idea de implementar y aprender sobre los hilos es aprender como estos son esenciales para aprovechar al máximo el potencial no solo de nuestros procesadores sino también de aplicaciones a grandes escalas, servicios en la nube y más. Todo esto debido a que los hilos permiten a los desarrolladores crear aplicaciones más rápidas, escalables y resistentes a fallos.

EXPLICACIÓN DEL CÓDIGO Y LOS RESULTADOS

En la primera sección de código se importan las librerías necesarias para el funcionamiento del programa. Entre ellas la librería “omp.h” la cual permite a mi programa utilizar las funciones de OpenMP para el procesamiento paralelo.
Así mismo importe la librería cstdlib la cual me es útil para generar los valores aleatorios que explicare mas adelante.

```
1   // Tarea1ProgramacionParalela.cpp : Este archivo contiene la función "main". La ejecución del programa comienza y termina ahí.
2   //
3
4   #include <iostream>
5   #include <omp.h>
6   #include <cstdlib>
7    using namespace std;
```

En la siguiente sección defino 3 variables globales, N nos indica el numero de elementos dentro de los arreglos, “chunk” indica el número de pedazos que cada hilo va a tomar de los arreglos para su procesamiento y finalmente mostrar la cual nos ayuda en nuestra función para imprimir la cantidad indicada de números contenidos en los arreglos.

También en esta sección se incluye la definición de la función imprimeArreglo(), la cual mediante el uso de un apuntador nos da la dirección de memoria de el arreglo que deseemos mostrar en la implementación de dicha función.

```
#define N 1000
#define chunk 100
#define mostrar 100

void imprimeArreglo(float* d);
```

En la tercera sección nos encontramos con la definición de nuestro método principal, un valor semilla de aleatoriedad con la función srand(), nuestros arreglos inicializados con tamaño N y posteriormente el llenado con valores aleatorios generados con la función rand() que utiliza la semilla previamente mencionada.
Dentro del for() de llenado de los arreglos observamos que se generan números dentro del rango 1 a 1000, esto puede cambiarse según sea necesario, pero para este ejemplo nos es útil. Al final asignamos una variable llamada pedazos con la cantidad de recursos a consumir por cada hilo de ejecución.

```
int main()
{
    std::cout << "Sumando Arreglos en Paralelo!\n\n";
    // Creando un valor semilla para los numeros aleatorios
    srand((unsigned)time(NULL));

    float a[N], b[N], c[N];
    int i;

    for (i = 0; i < N; i++) {
        a[i] = (rand() % 1000) + 1;
        b[i] = (rand() % 1000) + 1;
    }
    int pedazos = chunk;
```

A continuación, definimos que la instrucción for se debe de ejecutar de manera paralela usando la instrucción #pragma de OpenMP.
El comando shared nos sirve para indicar que los arreglos y pedazos pertenecen a un área de memoria compartida de esta manera permitimos que los hilos que se crean puedan acceder a los valores. también indicamos el tamaño del arreglo que debe tomar cada hilo e indicar que la variable i es privada, evitando que se cambie el valor en los diferentes hilos creados.
La última línea de este segmento nos indica que la planificación se realiza de manera estática, indicando también el tamaño que debe tomar cada hilo en la planificación.

```
#pragma omp parallel for \
shared(a,b,c,pedazos) private(i)\
schedule(static, pedazos)
```

Una vez implementada la sección anterior, ahora podemos ejecutar nuestro for() de manera tradicional, el compilador sabrá que debe de dividir el proceso entre el numero de hilos indicados y lo realizara mediante la librería OpenMP. Dentro del for(), asignamos al arreglo de resultados C la suma correspondiente al índice indicado por i de los 2 arreglos aleatorios A y B generados.

```
for (i = 0; i < N; i++)
    c[i] = a[i] + b[i];
```

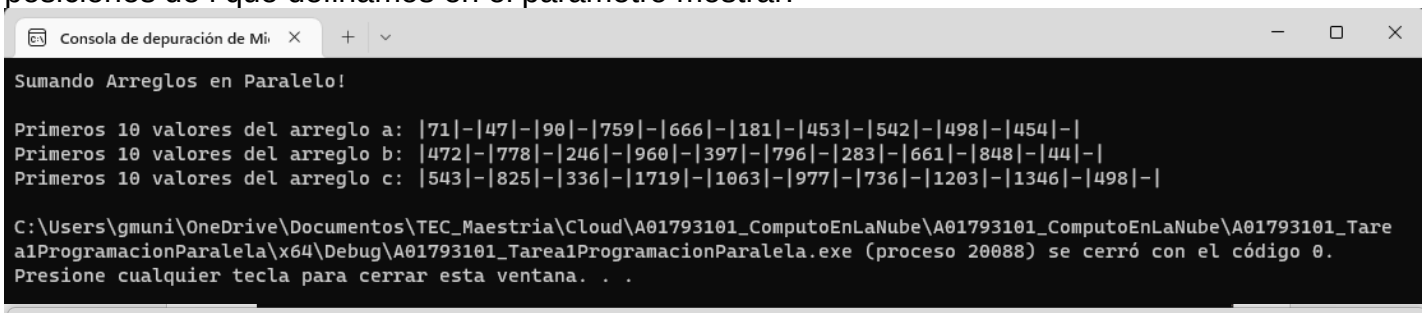
Finalmente, mediante el parámetro “mostrar” definido al inicio y la implementación de la función “imprimeArreglo()” desplegamos en la consola los n primeros elementos de cada arreglo.

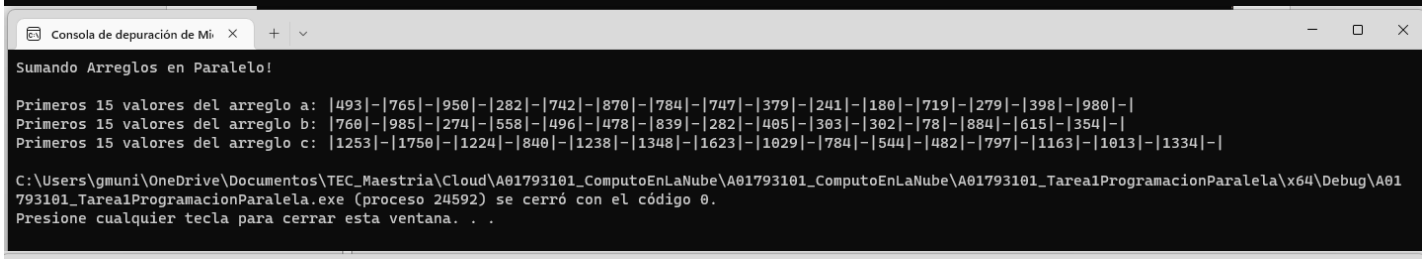
```
    std::cout << "Primeros " << mostrar << " valores del arreglo a: |";
    imprimeArreglo(a);
    std::cout << "Primeros " << mostrar << " valores del arreglo b: |";
    imprimeArreglo(b);
    std::cout << "Primeros " << mostrar << " valores del arreglo c: |";
    imprimeArreglo(c);
}

void imprimeArreglo(float* d) {
    for (int x = 0; x < mostrar; x++) {
        std::cout << d[x] << "|-|";
    }
    std::cout << std::endl;
}
```

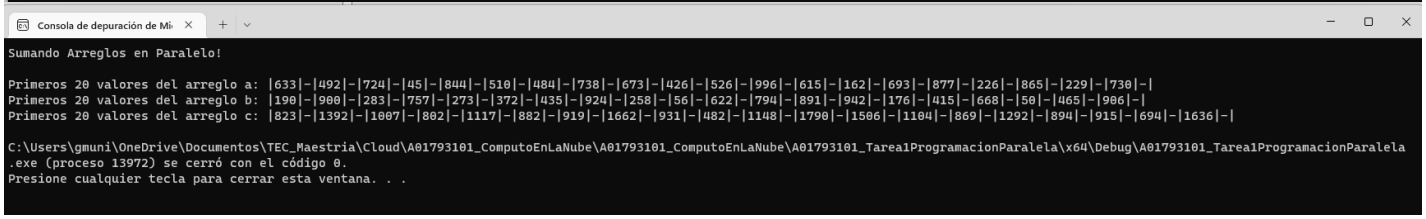
CAPTURAS DE PANTALLA

A continuación, se presentan 4 diferentes ejecuciones del código anterior. Todos estos con diferente valor de números a mostrar (10,15,20,100).

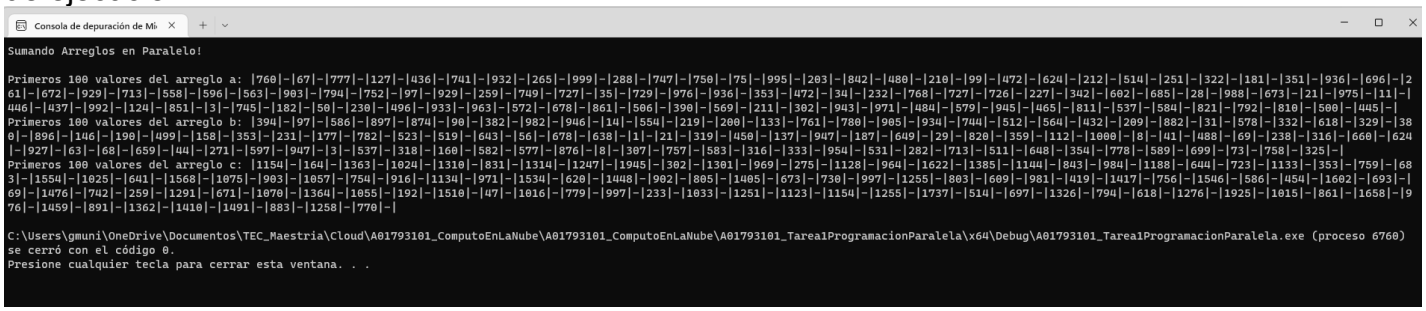
En cada una de las ejecuciones podemos observar como la suma de los valores en la posición i de los arreglos a y b es correspondiente con lo observado en el arreglo C. Es decir, si tomamos el elemento mostrado en la posición 1 de los arreglos A y B, aplicamos una operación de suma el resultado se encontrara en el arreglo C en la posición 1; Y esto es extensivo a cada una de las posiciones de i que definamos en el parámetro mostrar.







También podemos observar como para este algoritmo cuando el valor de la variable mostrar incrementa la visualización se vuelve mas difícil, sin embargo el rendimiento del algoritmo solo se ve afectado por números demasiado grandes, esto es debido a la distribución de la carga en los hilos de ejecución.



Enseguida se deja la implementación explicada anteriormente para su revisión.

LIGA DEL REPOSITORIO DEL PROYECTO EN GITHUB

Repo:
https://github.com/PosgradoMNA/A01793101_ComputoEnLaNube/tree/main/A01793101_Tarea1ProgramacionParalela

Cpp File:
https://github.com/PosgradoMNA/A01793101_ComputoEnLaNube/blob/main/A01793101_Tarea1ProgramacionParalela/A01793101_Tarea1ProgramacionParalela/A01793101_Tarea1ProgramacionParalela.cpp

REFLEXIÓN

Mi reflexión personal sobre esta tarea es que para aprender sobre la computación paralela es necesaria la comprensión de lo que es, como se usa y ejemplos de lo que son los hilos. El uso de hilos es de vital importancia en la computación paralela, ya que permite dividir una tarea en partes, las cuales se ejecutan simultáneamente en diferentes núcleos o procesadores. Esto ayuda a aumentar la velocidad de procesamiento y mejorar el rendimiento del sistema.
En este ejemplo la implementación fue sencilla debido a que el proceso no requería grandes esfuerzos de sincronización o de grandes cantidades de recursos compartidos. Como explique anteriormente mediante la librería OpenMP fue posible especificar que recursos serian compartidos (la instrucción shared fue útil para este propósito) pero al ser esta una pequeña demostración de como funcionan los hilos me parece que no tuvimos mayores desafíos que si se pueden presentar en futuras implementaciones con mayores cantidades de datos y recursos, dichas acciones requieren una planificación cuidadosa y mucho análisis de nuestra parte para poder ser capaces de entregar aplicaciones o tareas eficientes y escalables.

En general, me quedo con el aprendizaje de que el uso de hilos es una herramienta poderosa la cual podemos implementar en nuestro viaje por la computación paralela, el desarrollo de aplicaciones o el computo en la nube y su uso adecuado puede mejorar significativamente el rendimiento de cualquiera de estas tareas tan variadas a las que nos podemos enfrentar en nuestra vida profesional.