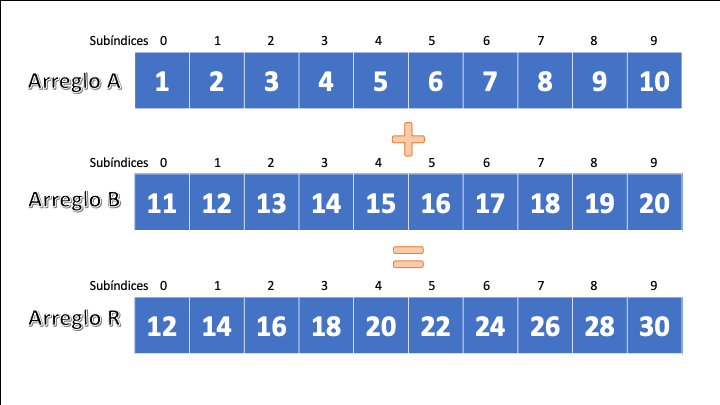
|  |  |
| --- | --- |
| Gravity with More Than Two Bodies | Astronomy  Tarea 1. Programación de una solución paralela  Diseñar algoritmos paralelos, implementándolos para resolver problemas numéricos y no numéricos | Descripción breve  Seleccionar modelos de paralelización apropiados a un programa aplicándolos en la creación de una versión paralela correcta que explote el ambiente de cómputo paralelo de última generación.  Raúl Omar Ayala Sevilla  Cómputo en la nube |

Introducción

La programación paralela permite ejecutar varios procesos al mismo tiempo, optimizando el rendimiento y los tiempos de ejecución de aplicaciones complejas. Este documento detalla el desarrollo de un proyecto de programación paralela, incluyendo la explicación del código, los resultados obtenidos y una reflexión sobre los beneficios e importancia de este enfoque.

Instruccciones

La utilización de arreglos en la programación es muy común, por ejemplo piense en una suma de dos arreglos como la mostrada en la siguiente imagen, donde tenemos dos arreglos llamados A y B de 10 elementos cada uno y deseamos crear un tercer arreglo con la sumatoria de cada uno de los elementos en los mismos subíndices.



Esto es bastante sencillo en la programación pero el tiempo que tomará en calcularse el resultado depende directamente de la cantidad de elementos que tienen los arreglos y la velocidad del procesador en la que se ejecutan las operaciones. Las sumas individuales de los elementos se realizan una por una en orden según los subíndices, entonces ¿Qué pasaría si los arreglos tienen millones de elementos?

La respuesta es que la suma de los arreglos tardaría bastante en resolverse. Entonces aquí podríamos aplicar la programación paralela para que la suma de algunos elementos de los arreglos se realicen en un thread, otra parte las controle otro thread y así sucesivamente, esto aprovechando que las sumas son independientes entre ellas.

Entonces podríamos pensar en algo como lo mostrado en la siguiente imagen.

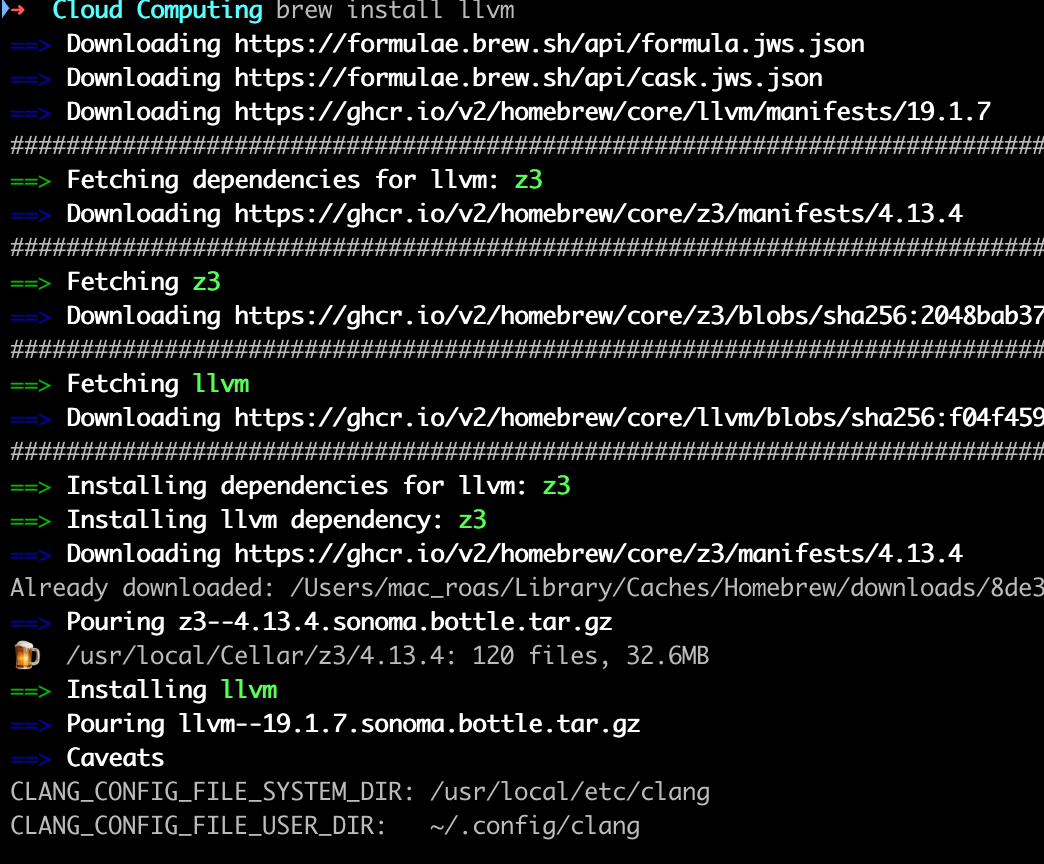
Este es el proceso que deseamos replicar, para lo que debes:

1. Crear un proyecto en Visual Studio
2. Configurarlo para que pueda utilizar la librería OpenMP
3. Crear los arreglos y asignarles valores aleatorios o pedirle los valores al usuario (la cantidad de elementos puede ser de 1000 para ejemplificar el proceso y obtener rápidamente resultados)
4. Realizar un for paralelo a través de las funciones de la librería OpenMP
5. Imprimir los arreglos o parte de ellos para comprobar que el arreglo resultante contiene la suma de los dos arreglos

Instalando Compiladores

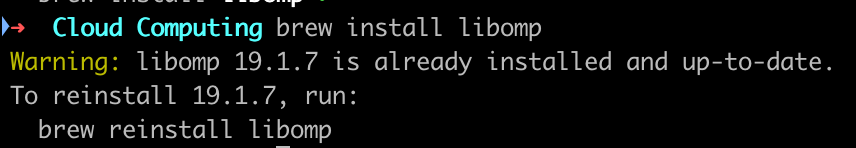
Por tener Mac he tenido que hacer un poco más de trabajo para poder compilar el programa y correr la librería de OpenMP. Tuve que buscar un compilador que fuera compatible con el sistema operativo y en este caso con mi procesador que en este caso es Intel.

La solución fue utilizar el compilador Clang, que instalé con Homebrew.



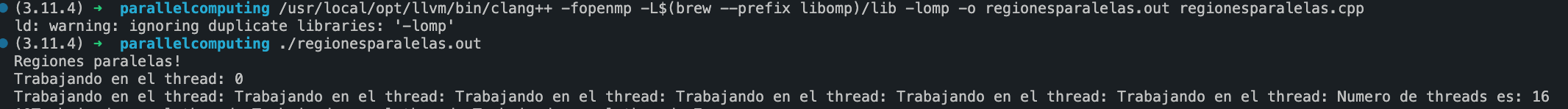
Aquí es importante señalar los Caveats de la dirección del compilador, porque tienes que seguir esta dirección al correr el programa y ejecutar la salida.

También instalé la librearía libomp, que es Openmp que sirve para con Clang o GCC

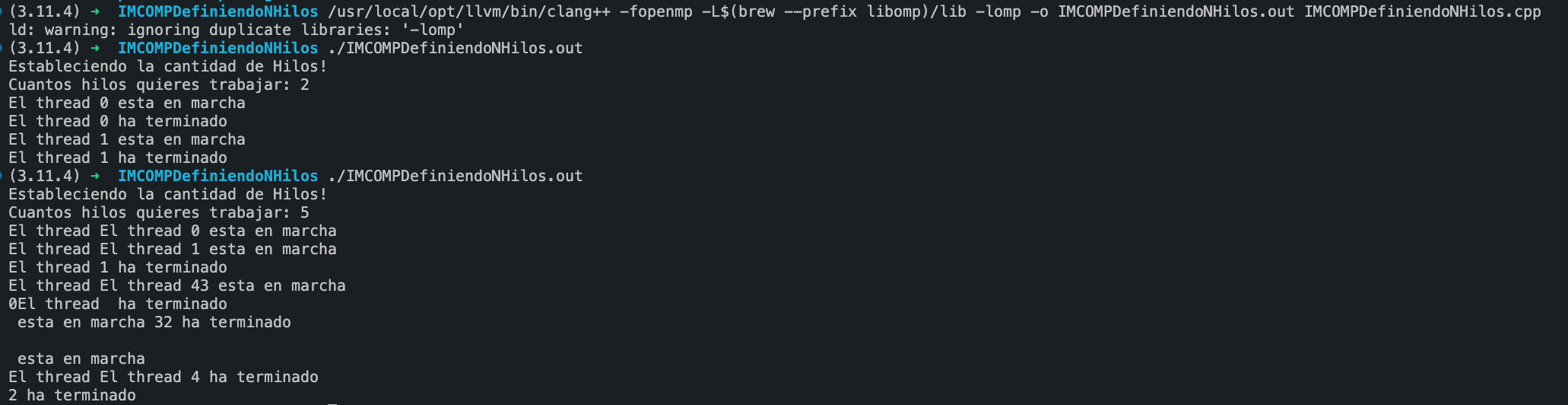


Finalmente, realicé los programas utlizando Visual Code y probé todos los ejercicios que estaban previos a la tarea probando la paralezación de los códigos.

Por ejemplo en el ejercicio de regiones paralelas, obtuve el siguiente resultado



Utilizando diferentes hilos:



De estos ejercicios no agrego el código porque es el que está en Canvas, solo agrego esta sección porque fue la parte de configuración en la que me demoré más tiempo y me parece muy importante.

Liga del repositorio de GitHub

El proyecto completo, así como su código fuente y documentación adicional, están disponibles en el siguiente repositorio de GitHub:

[https://github.com/roayala/ParallelArrays]

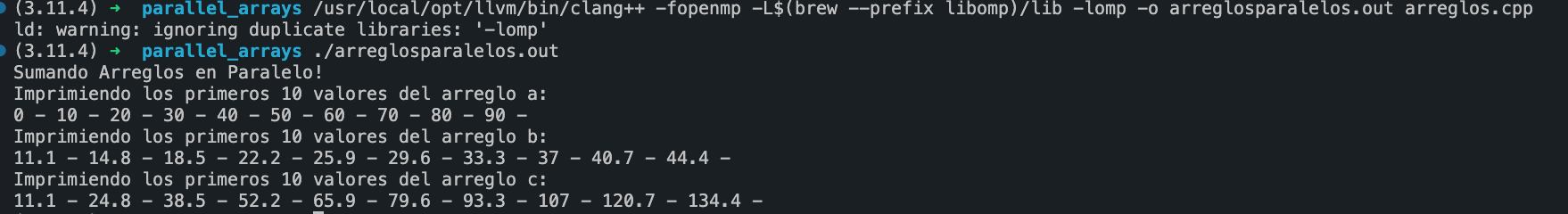
Capturas de pantalla

A continuación, se presentan capturas de pantalla de dos ejecuciones del proyecto:

Captura 1: Código completo, es el que se utilza para este ejercicio, esta es la versión que compartió el profesor como parte de la tarea.



Captura 2: Salida de la ejecución del código.



Explicación del código y los resultados

1. **Definiciones al inicio:**

cpp

CopiarEditar

#define N 1000

#define chunk 100

#define mostrar 10

* N: Es el tamaño de los arreglos (a, b, c) → tienen 1000 elementos.
* chunk: Tamaño de los "pedazos" en los que se dividirá el trabajo.
* mostrar: Número de elementos que se van a imprimir.

1. **Arreglos y variables:**

cpp

CopiarEditar

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

* a: Arreglo con valores iniciales de tipo float.
* b: Otro arreglo con valores iniciales.
* c: Arreglo donde se guarda la suma de los otros dos.

1. **Inicialización de datos:**

cpp

CopiarEditar

for (i = 0; i < N; i++) {

a[i] = i \* 10;

b[i] = (i + 3) \* 3.7;

}

Aquí se llenan los arreglos a y b con valores:

* a[i] es igual a i \* 10.
* b[i] es igual a (i + 3) \* 3.7.

1. **Sección paralela (con OpenMP):**

cpp

CopiarEditar

#pragma omp parallel for \

shared(a, b, c, pedazos) private(i) \

schedule(static, pedazos)

* **#pragma omp parallel for:** Indica que el bucle que sigue se ejecutará en paralelo.
* **shared(...):** Especifica que las variables a, b, c y pedazos son compartidas entre todos los hilos.
* **private(i):** Cada hilo usa su propia copia de i (índice del bucle).
* **schedule(static, pedazos):** Divide el trabajo en bloques de tamaño chunk (en este caso, 100).

La parte más importante de la práctica es el procedimiento que se encarga de crear los hilos y ejecutar las operaciones de manera paralela.  Esto es bastante sencillo con la librería OpenMP.

Primero define que la instrucción for se realizará en paralelo por medio de las instrucciones de la librería OpenMP, además debes indicarle que los arreglos estarán en un área de memoria compartida para que los diferentes hilos puedan accesar a los valores, también debes indicarle el tamaño de los pedazos que debe tomar cada hilo de los arreglos sin olvidar que la variable i de cada hilo creado debe ser privada para que no se mezcle con la de los ciclos de otro hilo, finalmente estableces la planificación como estática y le indicas también el tamaño de los pedazos que debe tomar la planificación para cada hilo. Ahora pudes colocar la instrucción for como si fuera una suma de arreglos común, pero en realidad ya todo está configurado para que se ejecute en paralelo. Esto se realiza a través del siguiente fragmento de código.

1. **El bucle paralelo:**

cpp

CopiarEditar

for (i = 0; i < N; i++) {

c[i] = a[i] + b[i];

}

* Cada hilo suma una parte de los arreglos a y b y guarda el resultado en c.

1. **Impresión de resultados:**

cpp

CopiarEditar

std::cout << "Imprimiendo los primeros " << mostrar << " valores del arreglo a: " << std::endl;

imprimeArreglo(a);

* Usa la función imprimeArreglo para mostrar los primeros 10 elementos de cada arreglo (a, b, c).

1. **Función imprimeArreglo:**

cpp

CopiarEditar

void imprimeArreglo(float \*d) {

for (int x = 0; x < mostrar; x++) {

std::cout << d[x] << " - ";

}

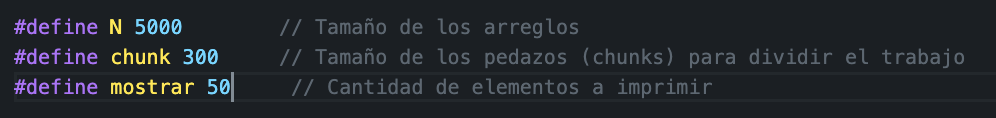
std::cout << std::endl;

}

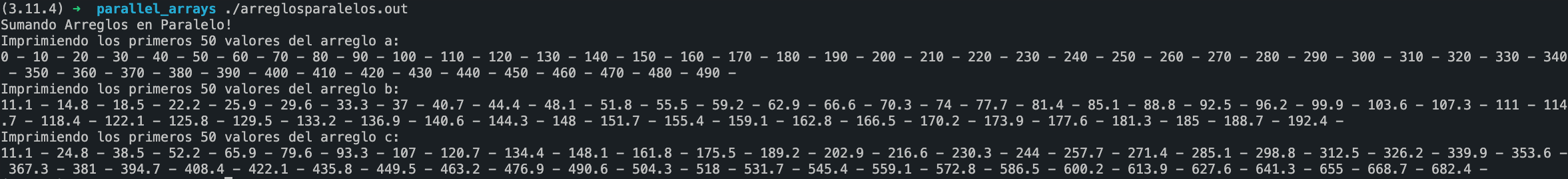
* Recorre los primeros mostrar elementos del arreglo y los imprime en pantalla.

Reflexión sobre la programación paralela

Para probar y jugar un poco más con el código, realicé los siguientes ejemplos. Lo que me parece importantes para resaltar es que se utilizó la programación con diferentes hilos de trabajo en los procesadores dentro de la computadora para generar el resultado.



Con el tamaño de arreglo de 5000, la cantidad de divisiones del procesos y pidiendo mostrar 50 resultados, esta fue la salida:



En este momento, con los avances que la computación ha tenido en tan poco tiempo, el desarrollo de aplicaciones utilizando programación paralela es clave. Al menos para mi ha sido impresionante el cambio que ha tenido, cuando empezó el crecimiento de las redes sociales, todo se enfocaba en el hardware, todo paso a paso fue migrando a la nube, y el uso de internet se volvió algo común.

Hoy el enfoque está en la programación, me parece que ese despegue se puede atribuir al cambio de paradigma de programación paralela. Al distribuir tareas entre varios hilos o procesos, se logra un uso más eficiente de los recursos del sistema, reduciendo significativamente los tiempos de ejecución y aumentando la capacidad de procesamiento.

Además de mejorar el rendimiento, la programación paralela también promueve un enfoque más modular y escalable en el desarrollo de software. Aunque como lo experimenté, la ejecución de los programas requiere un conocimiento de la ejecución de los programas y de la sincronización entre hardware y software.

Idealmente, es una herramienta que al menos a mi me demuestra que su uso tiene una gran capacidad de realizar tareas de forma eficiente y permite otro enfoque en el desarrollo de aplicaciones.