Descripción del código y de la estrategia de paralelización

Librerías

- ***iostream:*** proporciona facilidades para entrada y salida a través de flujos, como cin, cout, cerr y clog, permitiendo la lectura desde la entrada estándar y la escritura a la salida estándar y al error estándar, respectivamente.
- ***vector:*** implementa secuencias de elementos de manera que pueden redimensionarse automáticamente de forma dinámica. Proporciona acceso aleatorio a los elementos, inserción eficiente y eliminación de elementos al final.
- ***cmath:*** incluye funciones para operaciones matemáticas básicas, como funciones trigonométricas, raíz cuadrada, exponencial, logaritmo y más, facilitando cálculos matemáticos complejos.
- ***limits***: proporciona una manera de consultar propiedades de tipos aritméticos fundamentales, como los valores máximo y mínimo que pueden ser representados por un tipo.
- ***cstdlib:*** contiene funciones de propósito general, incluyendo la gestión de memoria dinámica, generación de números aleatorios y comandos del sistema.
- ***omp.h:*** archivo de cabecera para OpenMP, un conjunto de directivas de compilador, bibliotecas y variables de entorno que extienden programas en C, C++ y Fortran con directivas para explotar el paralelismo en entornos de memoria compartida.
- ***stdio.h***: la Biblioteca Estándar de Entrada y Salida en C, utilizada para operaciones relacionadas con entrada y salida (I/O), como leer y escribir datos en la consola o en archivos. Es parte de la biblioteca estándar de C y a menudo se utiliza en C++ para operaciones de E/S al estilo de C.
- ***chrono:*** parte de la Biblioteca Estándar de C++ para tratar con el tiempo. Proporciona medición de tiempo, duración y funcionalidades de reloj, permitiendo operaciones de tiempo de alta precisión.
- ***fstream:*** facilita el manejo de archivos en C++. Incluye clases como ifstream, ofstream y fstream para crear, leer y escribir en archivos.
- ***string***: proporciona la clase std::string junto con funciones y operadores que facilitan la manipulación de cadenas, como la concatenación, comparación y búsqueda.

Por Mariana Luna y Mauricio Vázquez

- ***sstream:*** incluye la clase stringstream, que permite tratar objetos basados en cadenas como flujos. Esto es particularmente útil para analizar datos de cadenas.
- ***algorithm:*** contiene una colección de plantillas de función para algoritmos que realizan operaciones en rangos de elementos, como buscar, ordenar, contar, manipular y más.
- ***random***: proporciona facilidades para generar números aleatorios usando diversas distribuciones. Ofrece una mejora significativa sobre técnicas más antiguas, permitiendo un mayor control sobre la generación de números aleatorios.

Descripción de las funciones implementadas

(Para mayor visualización del código, visitar:

- https://github.com/MauricioVazquezM/K_MEANS_PARALLELIZED/tree/main/K_MEANS_IMPLEMENTATION)

- ***función load_CSV:***

  Esta función lee un archivo CSV que contiene puntos bidimensionales en un arreglo bidimensional. Está diseñada para archivos donde cada línea representa un punto con sus coordenadas x e y separadas por un delimitador. La función proporciona un manejo básico de errores para problemas al abrir el archivo, pero asume que el archivo de entrada está bien formateado y que el arreglo de puntos ha sido asignado apropiadamente antes de llamar a la función.
- ***función save_to_CSV:***

  Esta función es de utilidad para guardar datos de puntos 3D de un arreglo en memoria a un archivo CSV, lo que hace posible persistir los datos en disco, compartirlos con otras aplicaciones o simplemente mantener un registro de los datos para uso futuro. La función maneja la verificación básica de errores para las operaciones con archivos y asume que los puntos están almacenados en un bloque contiguo de memoria accesible a través de un dato tipo float.
- ***función kmeans_paralelo:***

  Esta función implementa el algoritmo de clustering K-means en paralelo, utilizando la librería OpenMP para mejorar el rendimiento en sistemas multicore. A través de la

paralelización y una gestión cuidadosa de la memoria, se logra mejorar significativamente el rendimiento del clásico algoritmo de K-means.

- ***descripción main:***

  En esta parte del código, sirve como punto de entrada a la ejecución del programa en C++. Es donde la ejecución del programa comienza. Cuando se ejecuta un programa en C++, la función principal es llamada por el entorno de ejecución. Aquí se verifica que se recibieron los argumentos correctos en la línea de comando para su ejecución. Se llaman a la función implementada para la ejecución del modelo de K-means desarrollado y, por último, se llaman a las funciones para guardar y cargar los resultados en un archivo del tipo. csv.

Estrategia de paralelización

En primer lugar, se analizó, detalladamente, el código serial para identificar secciones de código que podían ser candidatas para ser paralelizables.
En segunda instancia, se implementó la siguiente estrategia de paralelización.Nos basamos en los siguientes puntos:

- ***#pragma omp parallel for:*** Directiva de OpenMP que indica al compilador que paralelice el for-loop que sigue. OpenMP divide automáticamente las iteraciones del bucle entre los hilos disponibles, permitiendo que el bucle se ejecute mucho más rápido en los procesadores multicore.

  Esta directiva se utilizó, inicialmente, para asignar cada punto a un cluster aleatorio al comienzo del algoritmo. Más adelante, esta directiva seria usada, en un for loop, para actualizar las coordenadas de los centroides.
- ***#pragma omp atomic:*** Directiva de OpenMP que garantiza que las adiciones hechas se realicen de manera atómica. Fue usada para prevenir condiciones de “carrera” cuando múltiples hilos intentaban actualizar las coordenadas del mismo centroide simultáneamente. Las operaciones atómicas fueron críticas para la implementación paralela de nuestro algoritmo. Asegurando la integridad de los datos cuando se actualizaban datos compartidos.
- ***#pragma omp parallel for reduction(&& : converge):*** Directiva de OpenMP que indica al compilador que paralelice el for-loop subsiguiente. OpenMP divide automáticamente

las iteraciones del bucle entre los hilos disponibles, permitiendo que el bucle se ejecute mucho más rápido en los procesadores multicore.

Se utilizó la cláusula *reduction(&&: converge)* para la variable *converge*. La cual es una bandera del tipo boolean que indica si el algoritmo ha convergido. En otras palabras, si no hubo cambios en las asignaciones de los clusters en dicha iteración. Con la cláusula mencionada, se le especifica al compilador que los valores de la variable *converge* deben ser combinados al final de la ejecución paralela del for-loop. Esto fue necesario para determinar, de manera eficiente y precisa, si se ha alcanzado la convergencia.