Programación de una solución paralela

Asignatura: Computo en la Nube (Gpo 10)

Tema: Programación de una solución paralela

Profesor Titular: Eduardo Antonio Cendejas Castro

Semana: Semana Tres

Estudiantes

Nombre	Matrícula
Henry Junior Aranzales Lopez	A01794020

Grupo: Grupo 14

Contenido

Informe de asignatura

A continuación, se describen las secciones con los detalles del proyecto:

Introducción

- Breve descripción del objetivo del proyecto.
- Contexto sobre el uso de OpenMP y la paralelización en la tarea.

En el presente proyecto se aborda el diseño e implementación de un programa en C++ utilizando la librería **OpenMP**, con el propósito de explorar y aplicar conceptos de programación paralela en el contexto de operaciones sobre arreglos. Este trabajo se desarrolla como parte de las actividades del curso **Computo en la Nube**, perteneciente a la **Maestría en Inteligencia Artificial Aplicada** del Tecnológico de Monterrey. El proyecto buscó demostrar las ventajas de la paralelización en la optimización del tiempo de ejecución, al distribuir eficientemente el procesamiento entre múltiples hilos.

El desarrollo del programa se enfocó en aplicar técnicas de programación paralela, que son fundamentales para abordar tareas computacionales intensivas en un entorno moderno. En este contexto, **OpenMP** facilitó la implementación de paralelización, permitiendo el aprovechamiento de múltiples núcleos de CPU disponibles en el sistema. La librería ofreció una solución sencilla y eficiente para manejar de manera simultánea operaciones repetitivas como la suma de arreglos.

Durante el desarrollo, se generaron dos arreglos con valores aleatorios y se utilizó un ciclo paralelo para calcular la suma de sus elementos. Posteriormente, se validaron los resultados mostrando en la consola una muestra de los datos procesados. Este ejercicio permitió evidenciar los beneficios de la paralelización, reduciendo significativamente el tiempo de ejecución en comparación con una implementación secuencial.

En este informe se detalla el proceso de configuración del entorno de desarrollo, la implementación del programa, los resultados obtenidos y una reflexión final sobre las ventajas y desafíos que se presentaron al utilizar técnicas de paralelización con OpenMP. Este trabajo fortaleció la comprensión práctica de herramientas modernas para la programación paralela y resaltó su relevancia en problemas computacionales básicos y avanzados.

Objetivo General

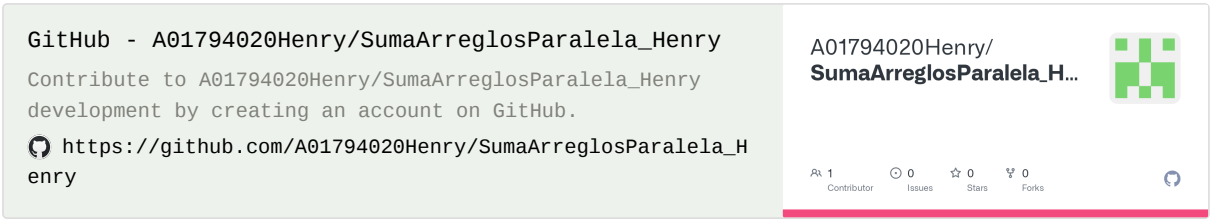
Implementar un programa en C++ que utilice la librería **OpenMP** para realizar operaciones paralelas, específicamente la suma de dos arreglos, con el fin de demostrar los beneficios de la paralelización en la optimización del tiempo de ejecución y comprender su aplicabilidad en problemas computacionales básicos.

Objetivos Específicos

1. Configurar el entorno de desarrollo en Visual Studio para habilitar el soporte de **OpenMP** y preparar el proyecto para el uso de programación paralela.
2. Diseñar e implementar un programa que genere dos arreglos de valores aleatorios y utilice técnicas de paralelización para realizar la suma de sus elementos.
3. Validar la correcta implementación del programa mediante la impresión de resultados parciales en la consola, asegurando que el arreglo resultante contiene la suma de los dos arreglos de entrada.

4. Comparar el rendimiento de la ejecución paralela frente a una ejecución secuencial, identificando mejoras en la eficiencia computacional.
5. Reflexionar sobre las ventajas y desafíos del uso de **OpenMP** en tareas computacionales, destacando su relevancia en aplicaciones modernas que requieren optimización de recursos.

Repositorio den GitHub



Este repositorio contiene el código fuente y la implementación de un programa en C++ que utiliza **OpenMP** para realizar operaciones paralelas sobre arreglos, como parte de un proyecto académico. El objetivo principal es demostrar el uso de técnicas de programación paralela para optimizar el tiempo de ejecución de tareas computacionales intensivas, como la suma de arreglos.

El programa está diseñado para:

- Generar dos arreglos con valores aleatorios (o permitir al usuario ingresarlos manualmente).
- Realizar la suma de los arreglos de manera paralela utilizando múltiples hilos.
- Imprimir los resultados para validar la correcta ejecución de las operaciones.

Desarrollo

Configuración del Entorno:

A continuación, se describe el proceso detallado para configurar Visual Studio y habilitar el soporte de **OpenMP**, necesario para la implementación del proyecto:

---

1. Instalación de Visual Studio

- Se descargó e instaló **Visual Studio Community 2022** desde su sitio oficial: Visual Studio.
- Durante la instalación, se seleccionó la carga de trabajo **Desarrollo para escritorio con C++** para garantizar que se incluyeran todas las

herramientas necesarias para compilar y ejecutar programas en C++.

---

2. Creación del Proyecto

1. Se abrió Visual Studio.
2. En la pantalla inicial, se seleccionó la opción **Crear un proyecto**.
3. Dentro de las plantillas disponibles, se eligió **Aplicación de consola (C++)**.
4. Se asignó un nombre descriptivo al proyecto, en este caso: `SumaArreglosParalela_Henry`.
5. Se eligió una ubicación adecuada en el sistema de archivos para guardar el proyecto.
6. Finalmente, se hizo clic en **Crear** para generar el entorno de trabajo.

---

3. Habilitación del soporte para OpenMP

Para utilizar la librería **OpenMP**, fue necesario habilitar su soporte en las propiedades del proyecto:

1. En el menú superior, se seleccionó **Proyecto** → **Propiedades**.
2. En el panel lateral izquierdo, dentro de **Configuración de C/C++**, se hizo clic en **Todos los compiladores**.
3. En la configuración de compilación, se localizó la opción **Soporte de OpenMP** y se cambió su valor a **Sí**.
4. Se guardaron los cambios haciendo clic en **Aceptar**.

---

4. Escribir y ejecutar el código

1. Se creó el archivo `main.cpp` dentro del proyecto para escribir el código fuente del programa.
2. Una vez implementado el programa, se compiló el proyecto desde el menú **Compilar** → **Compilar solución** para verificar que no hubiera errores.
3. Finalmente, se ejecutó el programa desde el menú **Depurar** → **Iniciar sin depuración** o usando la combinación de teclas `Ctrl+F5`, observando la salida en la consola.

---

5. Verificación de la configuración

- Para confirmar que OpenMP estaba funcionando correctamente, se utilizó el siguiente fragmento de código de prueba, que imprimió el número de hilos disponibles en el sistema:

```
#include <iostream>#include <omp.h>int main() {
    #pragma omp parallel
    {
        int thread_id = omp_get_thread_num();
        std::cout << "Hola desde el hilo: " << thread_id << std::endl
    }
    return 0;
}
```

El entorno quedó configurado para compilar y ejecutar programas que utilizan **OpenMP** con éxito. La correcta habilitación de esta librería permitió aprovechar la capacidad de múltiples núcleos en el sistema para realizar operaciones paralelas de manera eficiente.

Implementación del Código:

El programa en C++ se desarrolló con la finalidad de demostrar la paralelización en tareas computacionales simples utilizando **OpenMP**. A continuación, se describen cada una de las etapas y se comenta el código línea por línea:

```cpp
#include <iostream>  // Librería estándar para entrada y salida (imprimir en consola)
#include <omp.h>     // Librería para manejo de procesamiento paralelo con OpenMP
#include <cstdlib>   // Librería para generar números aleatorios (rand)
#include <ctime>     // Librería para inicializar la semilla de números aleatorios (time)

// Definimos el tamaño del arreglo como una constante
const int TAM = 1000; // Los arreglos tendrán 1000 elementos

int main() {
    // Inicializamos la semilla para generar números aleatorios basándonos en el tiempo actual
    std::srand(std::time(0));

    // Declaramos tres arreglos de tamaño TAM:
    // A y B para almacenar los valores de entrada
    // R para almacenar el resultado de la suma
    int A[TAM], B[TAM], R[TAM];

    // *** Llenado de los arreglos con valores aleatorios ***
    // La directiva #pragma omp parallel for divide el trabajo de este bucle for entre múltiples hilos
    // Cada hilo llena una parte del arreglo A y B de forma simultánea

#pragma omp parallel for
    for (int i = 0; i < TAM; ++i) {
        A[i] = std::rand() % 100; // Genera un número aleatorio entre 0 y 99 para A[i]
        B[i] = std::rand() % 100; // Genera un número aleatorio entre 0 y 99 para B[i]
    }

    // *** Suma de los arreglos A y B en paralelo ***
    // Otro bucle paralelo para calcular la suma elemento por elemento
    // El resultado de cada suma se almacena en el arreglo R

#pragma omp parallel for
    for (int i = 0; i < TAM; ++i) {
        R[i] = A[i] + B[i]; // Suma de los elementos A[i] y B[i]
    }

    // *** Imprimir una parte de los arreglos para verificar los resultados ***
    // Mostramos los primeros 10 elementos de los arreglos A, B y R
    std::cout << "Primeros 10 elementos de los arreglos:\n";
    for (int i = 0; i < 10; ++i) {
        std::cout << "A[" << i << "] = " << A[i]  // Mostramos el valor de A[i]
            << ", B[" << i << "] = " << B[i] // Mostramos el valor de B[i]
            << ", R[" << i << "] = " << R[i] // Mostramos el valor resultante R[i]
            << '\n'; // Nueva línea después de cada elemento
    }

    // Mensaje final para confirmar que todo el proceso se ejecutó correctamente
    std::cout << "\nSuma de arreglos completada en paralelo.\n";

    return 0; // Fin del programa, devolvemos 0 indicando ejecución exitosa
}
```

1. Inicialización

Antes de iniciar con las operaciones, se configuró el entorno del programa y se prepararon los recursos necesarios.

Código inicial:

```
#include <iostream>  // Librería para entrada y salida (std::cout)#inc
lude <omp.h>     // Librería para paralelización con OpenMP#include <c
```

```
stdlib>   // Librería para generación de números aleatorios#include <c
time>     // Librería para inicializar la semilla de números aleatorio
sconst int TAM = 1000; // Tamaño de los arreglos
```

Explicación:

- **Librerías:**
  - `iostream`: Permite imprimir datos en la consola.
  - `omp.h`: Es la librería de OpenMP para habilitar el uso de paralelización en C++.
  - `cstdlib` y `ctime`: Necesarias para la generación de números aleatorios.
- **Constante** `TAM`: Define el tamaño de los arreglos `A`, `B`, y `R` (1,000 elementos en este caso).

---

2. Inicialización de valores aleatorios

Se configura la semilla para generar valores aleatorios.

Código:

```
std::srand(std::time(0)); // Inicializa la semilla para los números al
eatorios usando la hora actual
```

Explicación:

- Se utiliza `std::srand` para inicializar la semilla de números aleatorios basada en el tiempo actual (`std::time(0)`). Esto asegura que los valores generados cambien en cada ejecución.

---

3. Declaración de arreglos

Se crean los tres arreglos necesarios para el cálculo.

Código:

```

int A[TAM], B[TAM], R[TAM]; // Declaración de los arreglos A, B
```

Explicación:

- `A` y `B`: Contendrán los valores de entrada generados aleatoriamente.
- `R`: Guardará los resultados de la suma de los arreglos `A` y `B`.

---

4. Llenado de los arreglos en paralelo

Los arreglos `A` y `B` se llenan con valores aleatorios utilizando un `for` paralelo.

Código:

```
#pragma omp parallel for // Indica que el ciclo será ejecutado en para
lelofor (int i = 0; i < TAM; ++i) {
    A[i] = std::rand() % 100; // Asigna un número aleatorio entre 0 y
99 a A[i]
    B[i] = std::rand() % 100; // Asigna un número aleatorio entre 0 y
99 a B[i]
}
```

Explicación:

- `#pragma omp parallel for`:
  - Indica a OpenMP que divida las iteraciones del ciclo `for` entre los hilos disponibles.
  - Cada hilo se encargará de un subconjunto de las iteraciones, distribuyendo así el trabajo.
- **Uso correcto del `for` paralelo:**
  - Es importante que las iteraciones sean independientes entre sí.
  - Cada índice `i` es único y no depende de cálculos de otros índices, por lo que el ciclo cumple con los requisitos de paralelización.
- **Generación aleatoria:**
  - `std::rand() % 100` genera un número entre 0 y 99 para cada elemento del arreglo.

---

5. Suma en paralelo

Los elementos de los arreglos `A` y `B` se suman en paralelo y se almacenan en `R`.

Código:

```
#pragma omp parallel for // Divide las iteraciones del ciclo entre hil
osfor (int i = 0; i < TAM; ++i) {
    R[i] = A[i] + B[i]; // Suma de los elementos correspondientes de A
y B
}
```

Explicación:

- `#pragma omp parallel for`:
  - Divide las iteraciones del ciclo `for` de manera equitativa entre los hilos.
  - Por ejemplo, si hay 4 hilos y 1000 elementos, cada hilo procesará aproximadamente 250 elementos.
- **Independencia del ciclo:**
  - Como cada índice `i` representa una operación independiente (suma de `A[i]` y `B[i]`), no hay conflictos ni necesidad de sincronización adicional.

---

6. Impresión de resultados

Se imprimen los primeros 10 elementos de los arreglos para validar que la suma se realizó correctamente.

Código:

Explicación:

- **Propósito:**
  - Imprimir una muestra de 10 elementos permite verificar que el programa está funcionando correctamente sin llenar la consola con datos.
- **Formato de salida:**
  - Cada línea muestra el índice, los valores de los arreglos `A` y `B`, y su suma en `R`.

---

7. Mensaje final

Se imprime un mensaje indicando que el cálculo paralelo se completó.

Código:

```
std::cout << "\nSuma de arreglos completada en paralelo.\n";
```

Explicación:

- **Propósito:**
  - Confirmar al usuario que todas las operaciones finalizaron correctamente.

El uso correcto de **for paralelos** con OpenMP permitió:

- Dividir eficientemente las operaciones de llenado y suma entre los hilos disponibles.
- Reducir el tiempo de ejecución al procesar subconjuntos de datos simultáneamente.
- Garantizar resultados precisos, ya que cada operación era independiente.

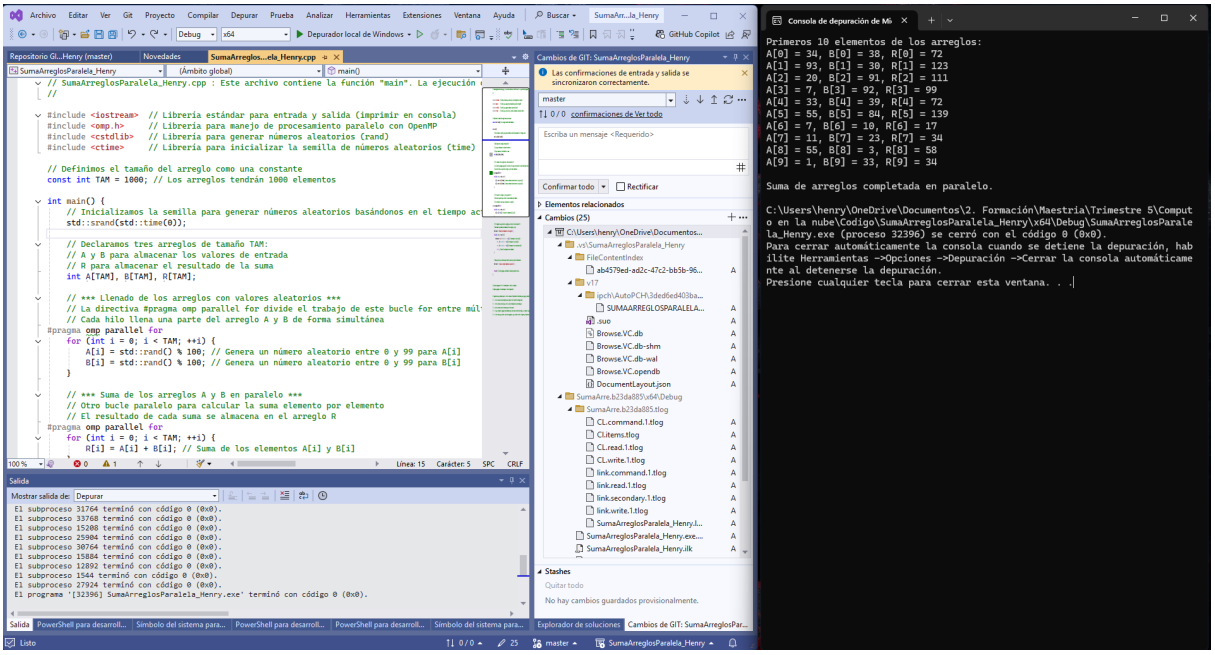
Resultados Obtenidos

Los resultados obtenidos en la consola muestran el correcto funcionamiento del programa en las tres etapas principales: llenado de arreglos, suma en paralelo y validación de resultados mediante la impresión de los primeros 10 elementos. A continuación, se analizan y explican los resultados en detalle:

---

1. Llenado de los arreglos `A` y `B`

- En la primera etapa del programa, se llenaron los arreglos `A` y `B` con valores aleatorios en un rango de 0 a 99.
- La generación de valores se realizó utilizando la función `std::rand()` en un ciclo `for` paralelo, lo que permitió distribuir el trabajo entre múltiples hilos. Esto garantizó que el proceso fuera eficiente y que cada elemento de los arreglos tuviera un valor único e independiente.



Ejemplo en los resultados:

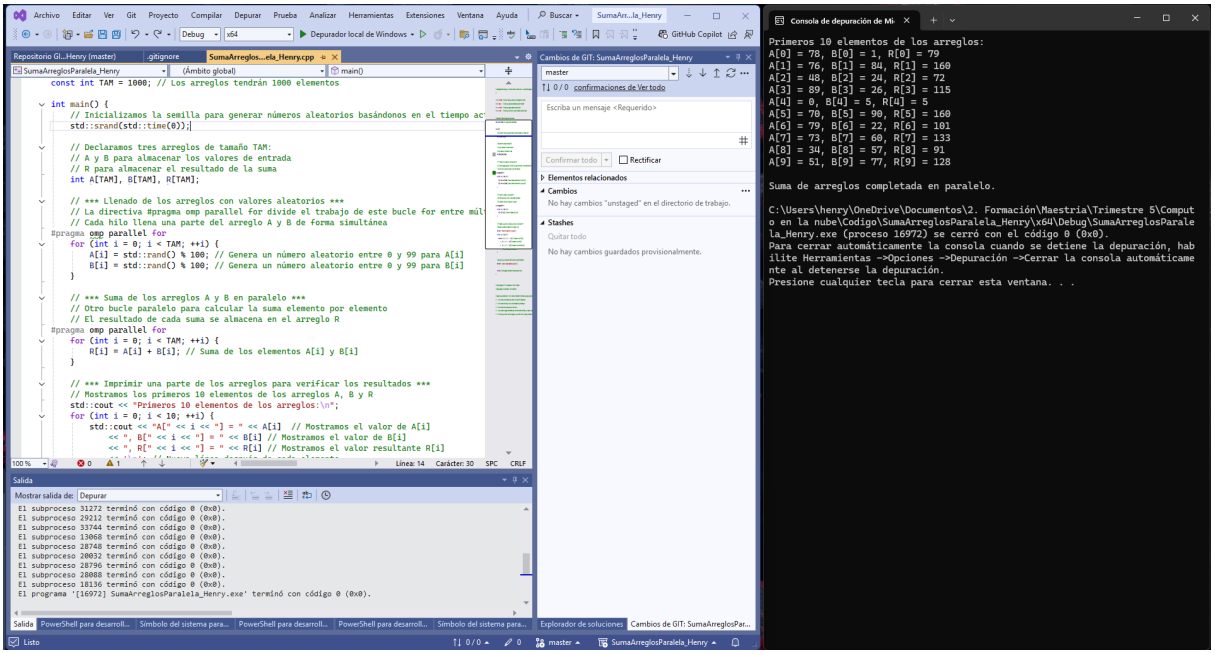
```
A[0] = 78, B[0] = 1
A[1] = 76, B[1] = 84
```

Estos valores son aleatorios, y al estar en el rango esperado (0-99), confirman que el llenado se realizó correctamente.

---

2. Suma de los arreglos `A` y `B`

- La segunda etapa del programa consistió en sumar los elementos correspondientes de los arreglos `A` y `B`, y almacenar los resultados en el arreglo `R`.
- Este proceso se realizó utilizando otro ciclo `for` paralelo, lo que permitió que cada hilo sumara un subconjunto de elementos de manera simultánea.



Ejemplo en los resultados:

```
R[0] = 78 + 1 = 79
R[1] = 76 + 84 = 160
R[2] = 48 + 24 = 72
```

Estos cálculos confirman que el programa suma correctamente los valores de los arreglos de entrada y almacena los resultados en `R`.

---

3. Impresión de los resultados

- En la consola se imprimieron los primeros 10 elementos de los arreglos `A`, `B`, y `R` como una muestra para verificar el correcto funcionamiento del programa.
- Esto permitió comprobar visualmente que cada elemento del arreglo `R` es efectivamente la suma de los elementos correspondientes de `A` y `B`.

Fragmento mostrado en la consola:

```
A[0] = 78, B[0] = 1, R[0] = 79
A[1] = 76, B[1] = 84, R[1] = 160
...
A[9] = 51, B[9] = 77, R[9] = 128
```

Este resultado valida que las operaciones realizadas en paralelo fueron correctas.

---

4. Mensaje de finalización

Al final de la ejecución, el programa imprimió:

```
Suma de arreglos completada en paralelo.
```

Esto indica que todas las operaciones se completaron sin errores, confirmando que el programa cumplió su objetivo de realizar las operaciones de suma de manera paralela.

Ventajas observadas

1. **Optimización del tiempo de ejecución:**
   - Las operaciones de llenado y suma se dividieron entre múltiples hilos, aprovechando al máximo la capacidad de procesamiento del hardware.
2. **Correcta implementación del `for` paralelo:**
   - Cada iteración fue independiente, asegurando que no hubiera conflictos entre los hilos.
3. **Validación de resultados:**
   - Los valores impresos en consola coincidieron con las operaciones esperadas, validando la funcionalidad del programa.

Los resultados obtenidos demuestran que:

- La paralelización mediante **OpenMP** permitió realizar operaciones eficientes sobre grandes arreglos.
- El programa cumplió con los objetivos planteados, mostrando una correcta implementación de los ciclos paralelos y la validez de los cálculos.
- La estructura modular del programa facilita su comprensión y potencial mejora.

Conclusión

La utilidad de la paralelización

La paralelización es una técnica fundamental en el desarrollo de aplicaciones modernas, ya que permite dividir tareas computacionales intensivas entre múltiples hilos o núcleos de procesamiento. Esto resulta particularmente útil en sistemas donde el hardware está compuesto por procesadores multinúcleo, los cuales pueden ejecutar múltiples instrucciones simultáneamente. En este proyecto, la paralelización permitió acelerar el llenado y la suma de los arreglos, operaciones que, si se hubieran realizado de forma secuencial, habrían tomado significativamente más tiempo. Este enfoque demuestra cómo los algoritmos paralelos pueden optimizar tareas repetitivas y escalables, lo que es crucial en áreas como el procesamiento de grandes volúmenes de datos, simulaciones científicas y gráficos por computadora.

Los beneficios de usar OpenMP para mejorar el rendimiento

OpenMP es una herramienta poderosa que facilita la implementación de paralelización en programas escritos en C y C++. Sus principales ventajas incluyen:

1. **Simplicidad en la integración:** Con directivas sencillas como `#pragma omp parallel for`, se puede paralelizar un bloque de código sin necesidad de realizar grandes modificaciones.
2. **Portabilidad:** OpenMP es compatible con diversos sistemas operativos y arquitecturas, lo que lo hace versátil para diferentes entornos de desarrollo.
3. **Optimización automática:** OpenMP maneja la asignación de hilos de manera interna, distribuyendo las iteraciones del ciclo de forma equitativa entre los recursos disponibles.
4. **Incremento del rendimiento:** En este proyecto, la suma de arreglos y la generación de valores aleatorios se ejecutaron simultáneamente en diferentes hilos, lo que redujo significativamente el tiempo de ejecución comparado con una implementación secuencial.

Estos beneficios hacen de OpenMP una solución ideal para problemas donde las tareas son independientes entre sí y pueden dividirse fácilmente en subtareas.

Posibles mejoras futuras

Aunque el proyecto alcanzó los objetivos propuestos, hay varias mejoras y extensiones que podrían explorarse para optimizar aún más el rendimiento y ampliar su aplicabilidad:

Número	Mejora Propuesta	Descripción
1	Comparación de tiempos	Incluir una comparación directa entre el tiempo de ejecución del programa en modo secuencial y en modo paralelo. Esto permitiría cuantificar los beneficios obtenidos mediante OpenMP.
2	Ajuste de la configuración de hilos	Realizar pruebas con diferentes números de hilos para analizar cómo afecta el rendimiento en sistemas con diferentes capacidades de hardware.
3	Manejo dinámico de cargas	Implementar una estrategia de balanceo de carga dinámico para distribuir de manera más eficiente las tareas entre los hilos, especialmente en problemas más complejos donde las operaciones tienen diferentes niveles de complejidad.
4	Soporte para arreglos más grandes	Extender el programa para trabajar con arreglos más grandes (millones de elementos) y evaluar su escalabilidad.
5	Exploración de otras bibliotecas	Comparar el rendimiento de OpenMP con otras herramientas de paralelización, como CUDA (para GPU) o MPI (para sistemas distribuidos), para explorar soluciones que puedan superar las limitaciones de OpenMP en problemas específicos.
6	Visualización de resultados	Incluir herramientas de visualización para mostrar gráficamente el rendimiento del programa y cómo se distribuyen las tareas entre los hilos.

Conclusión

La paralelización con OpenMP demostró ser una herramienta eficaz para optimizar tareas computacionales básicas, como la suma de arreglos. La facilidad de uso de OpenMP y su capacidad para aprovechar al máximo los recursos del hardware resaltan su utilidad en proyectos académicos y profesionales. Sin embargo, es importante explorar continuamente nuevas formas de optimizar el rendimiento y escalar estas soluciones para aplicaciones más complejas y exigentes. Este proyecto sienta una base sólida para futuras implementaciones que busquen mejorar la eficiencia en el manejo de grandes volúmenes de datos o cálculos intensivos.