Tecnológico de Monterrey

Instituto Tecnológico y de Estudios Superiores de Monterrey

Escuela de Ingeniería y Ciencias

Programas en Línea

Maestría en Inteligencia Artificial Aplicada (MNA)

Cómputo en la Nube

Tarea 1. Programación de una solución paralela.

Alumno:

Jorge Chávez Badillo A01749448

Profesores Titulares:

Eduardo Antonio Cendejas Castro

Fecha:

26 de enero de 2025

Programación de una solución paralela

Introducción

En programación, los arreglos son estructuras fundamentales utilizadas para almacenar y manejar colecciones de datos. Una operación común es la suma de dos arreglos, donde cada elemento en un subíndice de un arreglo se combina con el elemento correspondiente del otro. Esta operación es sencilla de implementar, pero su tiempo de ejecución depende del tamaño de los arreglos y la capacidad de procesamiento de la máquina.

Cuando los arreglos contienen millones de elementos, el tiempo requerido para realizar estas operaciones secuenciales puede ser considerable. Aquí es donde entra la programación paralela, que permite dividir el trabajo en tareas más pequeñas y ejecutarlas simultáneamente en múltiples hilos o procesadores. En este contexto, herramientas como OpenMP ofrecen un marco eficiente para implementar programación paralela en aplicaciones que requieren optimizar el tiempo de ejecución.

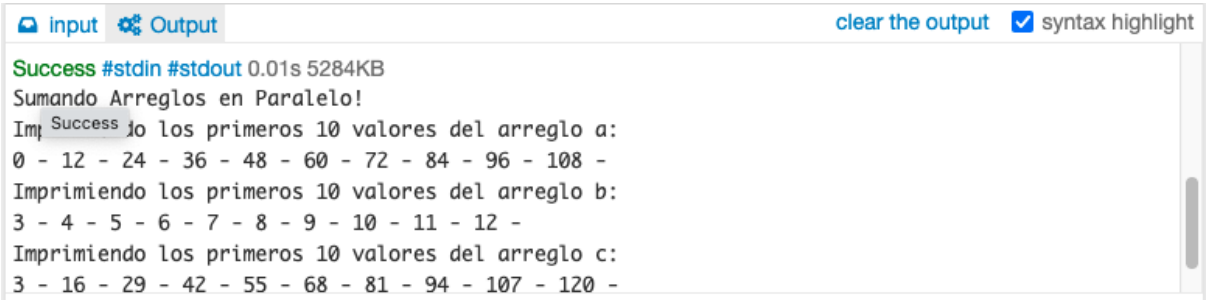
Liga de repositorio de Github

https://github.com/jorgechb/computo-en-la-nube

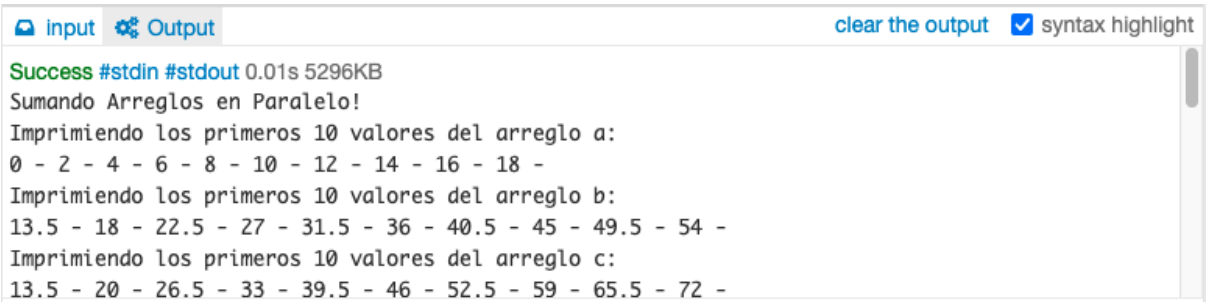
Ejecuciones del proyecto

Prueba 1:

Prueba 2:


input
Output
clear the output
syntax highlight
Success #stdin #stdout 0.01s 5284KB
Sumando Arreglos en Paralelo!
Success
Imprimiendo los primeros 10 valores del arreglo a:
0 - 12 - 24 - 36 - 48 - 60 - 72 - 84 - 96 - 108 -
Imprimiendo los primeros 10 valores del arreglo b:
3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 -
Imprimiendo los primeros 10 valores del arreglo c:
3 - 16 - 29 - 42 - 55 - 68 - 81 - 94 - 107 - 120 -

Prueba 3:


input
Output
clear the output
syntax highlight
Success #stdin #stdout 0.01s 5296KB
Sumando Arreglos en Paralelo!
Imprimiendo los primeros 10 valores del arreglo a:
0 - 2 - 4 - 6 - 8 - 10 - 12 - 14 - 16 - 18 -
Imprimiendo los primeros 10 valores del arreglo b:
13.5 - 18 - 22.5 - 27 - 31.5 - 36 - 40.5 - 45 - 49.5 - 54 -
Imprimiendo los primeros 10 valores del arreglo c:
13.5 - 20 - 26.5 - 33 - 39.5 - 46 - 52.5 - 59 - 65.5 - 72 -

Explicación de código y resultados

```cpp
#include <iostream>
#include <omp.h>

#define N 1000
#define chunk 100
#define mostrar 10

void imprimeArreglo(float *d);

int main()
{
    std::cout << "Sumando Arreglos en Paralelo!\n";
    float a[N], b[N], c[N];
    int i;

    for (i = 0; i < N; i++)
    {
        a[i] = i * 2;
        b[i] = (i + 3) * 4.5;
    }

    int pedazos = chunk;

#pragma omp parallel for shared(a, b, c, pedazos) private(i) \
    schedule(static, pedazos)
    for (i = 0; i < N; i++)
        c[i] = a[i] + b[i];

    std::cout << "Imprimiendo los primeros " << mostrar << " valores del arreglo a: " << std::endl;
    imprimeArreglo(a);
    std::cout << "Imprimiendo los primeros " << mostrar << " valores del arreglo b: " << std::endl;
    imprimeArreglo(b);
    std::cout << "Imprimiendo los primeros " << mostrar << " valores del arreglo c: " << std::endl;
    imprimeArreglo(c);

    return 0;
}

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

A continuación se desglosan los pasos principales del código y cómo es que funciona al implementar el paralelismo para resolver el problema.

Declaración de constantes y arreglos:

- Se definen tres constantes:
    - N: Tamaño de los arreglos (1000 elementos).
    - chunk: Tamaño de los bloques de datos que cada hilo procesará (100 elementos).
    - mostrar: Número de elementos a imprimir (10).
- Se crean tres arreglos:
    - a y b: Se inicializan con valores calculados.
    - c: Almacena el resultado de la suma de los arreglos a y b.

Inicialización de los arreglos:

- a[i] contiene valores como i * 2 (por ejemplo: 0, 2, 4, ...).

- b[i] contiene valores como (i + 3) * 4.5 (por ejemplo: 13.5, 18, ...).

Paralelización con OpenMP:

- Se utiliza la directiva #pragma omp parallel for para paralelizar el ciclo que calcula la suma de los arreglos.
- Los arreglos (a, b, c) están en memoria compartida entre hilos (shared).
- Cada hilo trabaja con una sección de chunk (100 elementos) y utiliza su propia variable i (private(i)), evitando conflictos con otros hilos.
- La planificación estática (schedule(static, pedazos)) asegura que cada hilo procese bloques consecutivos de tamaño chunk.

Cálculo paralelo:

- Cada hilo realiza la suma c[i] = a[i] + b[i] en paralelo, dividiendo el trabajo entre los núcleos del procesador.

Impresión de resultados:

- La función imprimeArreglo imprime los primeros mostrar elementos de cada arreglo (a, b y c) para verificar los valores iniciales y el resultado de la suma.

Finalmente, los resultados obtenidos en la ejecución del código muestran la correcta suma de los arreglos en paralelo, tal como se esperaba:

1. Arreglo a: Los primeros 10 elementos son valores calculados como i * 2:
    - 0, 2, 4, 6, 8, 10, 12, 14, 16, 18
2. Arreglo b: Los primeros 10 elementos son valores calculados como (i + 3) * 4.5:
    - 13.5, 18, 22.5, 27, 31.5, 36, 40.5, 45, 49.5, 54
3. Arreglo c (resultado): Los primeros 10 elementos corresponden a la suma de a[i] + b[i]:
    - 13.5, 20, 26.5, 33, 39.5, 46, 52.5, 59, 65.5, 72

Esto confirma que el cálculo paralelo realizado con OpenMP se ejecutó correctamente, y los resultados obtenidos son consistentes con la operación de suma de arreglos. La paralelización permitió optimizar el tiempo de ejecución en comparación con un bucle secuencial.

Reflexión sobre la programación paralela

La programación paralela representa una herramienta fundamental en la era de los procesadores multinúcleo, donde la optimización del tiempo y los recursos es clave para resolver problemas complejos. La suma de arreglos, como se ejemplifica con el uso de OpenMP, demuestra cómo las operaciones independientes pueden dividirse eficientemente

para ejecutarse simultáneamente, reduciendo el tiempo de procesamiento y maximizando el desempeño del hardware.

OpenMP sobresale al ofrecer una interfaz accesible y flexible para implementar paralelismo sin necesidad de manejar directamente la complejidad asociada a la creación y sincronización de hilos. Esto permite a los programadores concentrarse en el problema a resolver, dejando la gestión de recursos al entorno de ejecución.

En reflexiones finales, la programación paralela no solo mejora el rendimiento computacional, sino que también abre puertas a nuevas posibilidades en el manejo de grandes volúmenes de datos y tareas intensivas. Sin embargo, es importante recordar que su aplicación debe ser cuidadosa: no todos los problemas son paralelizables, y la eficiencia puede depender de factores como el balance de carga entre hilos y la sobrecarga de coordinación. Por lo tanto, comprender los principios y herramientas como OpenMP es esencial para aprovechar su potencial de manera efectiva y sostenible.

Referencias

- Robey y Zamora. (2021). Parallel and High Performance Computing. Manning Publications. https://learning.oreilly.com/library/view/parallel-and-high/9781617296468/OEBPS/Text/ch01_Robey.html
- OpenMP Architecture Review Board. (2025). *OpenMP Application Program Interface.* https://www.openmp.org
- B. Kirk y W. Hwu, 2016, Programming Massively Parallel Processors, 3er edition. Morgan Kaufmann. https://learning.oreilly.com/library/view/programming-massively-parallel/9780128119877/xhtml/chp001.xhtml