Maestría en Inteligencia Artificial Aplicada (MNA-V)

Materia: Cómputo en la nube (Gpo 10)

Profesor: Eduardo Antonio Cendejas Castro

Actividad: “Tarea 1. Programación de una solución paralela”

Alumno: Samuel Elías Flores González

Matrícula: A01793668

Fecha: 29 de enero de 2023

índice

Reporte: Programación de una solución paralela

Introducción

La programación paralela o paralelismo es una técnica usada en la computación que le permite a un algoritmo ser ejecutado en secciones, haciendo uso de múltiples recursos como lo son las unidades de procesamiento para que finalmente sean unidas nuevamente. Esta técnica nos permite la ejecución de algoritmos en un menor tiempo, especialmente útil para la resolución de problemas que requieren un tiempo de elevado de procesamiento.

El propósito de la actividad consta en desarrollar un algoritmo paralelo que sume dos arreglos denominados A y B, y almacene el resultado en un tercer arreglo C. Aunque pueda parecer una tarea sencilla, si la cantidad de elementos a procesar fuera bastante elevada, el tiempo de ejecución también lo seria, por lo cual la solución paralela sería la opción ideal para reducir el tiempo de resolución.

El desarrollo de esta actividad contribuye al cumplimiento de los objetivos:

- 2.2 Diseñar algoritmos paralelos, implementándolos para resolver problemas numéricos y no numéricos.
- 2.3 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.

Liga del repositorio de github donde se encuentra el proyecto desarrollado

https://github.com/SamuelFlores11/Computo-en-la-nube/tree/main/Tarea%201%20Programacion%20de%20una%20solucion%20paralela/pruebaOMP

Capturas de pantalla de ejecución de proyecto

Podemos observar que al modificar los valores de las variables definidas obtenemos los mismos resultados, sin embargo podemos llegar a la conclusión que el tiempo de ejecución si varia dependiendo el valor del chunk, el cual es el valor de las secciones o hilos en que se divide el procesamiento.

En la primera ejecución usamos el valor de chunk mas alto (100) y obtuvimos un tiempo de procesamiento más reducido que las siguientes dos ejecuciones.

Primera ejecución

Valores:

```
//Definimos variables
#define N 1000
#define chunk 100
#define mostrar 10
```

Resultado:

```
Microsoft Visual Studio Debug Console
Sumando Arreglos en Paralelo!
Imprimiendo los primeros 10 valores del arreglo a:
0 - 10 - 20 - 30 - 40 - 50 - 60 - 70 - 80 - 90 -
Imprimiendo los primeros 10 valores del arreglo b:
11.1 - 14.8 - 18.5 - 22.2 - 25.9 - 29.6 - 33.3 - 37 - 40.7 - 44.4 -
Imprimiendo los primeros 10 valores del arreglo c:
11.1 - 24.8 - 38.5 - 52.2 - 65.9 - 79.6 - 93.3 - 107 - 120.7 - 134.4 -

C:\Users\user\source\repos\ProyectosOMP\pruebaOMP\x64\Debug\pruebaOMP.exe (process 10612) exited with code 0.
Press any key to close this window . . .
```

Segunda ejecución

Valores:

```
//Definimos variables
#define N 100
#define chunk 2
#define mostrar 10
```

Resultado:

```
Microsoft Visual Studio Debug Console
Sumando Arreglos en Paralelo!
Imprimiendo los primeros 10 valores del arreglo a:
0 - 10 - 20 - 30 - 40 - 50 - 60 - 70 - 80 - 90 -
Imprimiendo los primeros 10 valores del arreglo b:
11.1 - 14.8 - 18.5 - 22.2 - 25.9 - 29.6 - 33.3 - 37 - 40.7 - 44.4 -
Imprimiendo los primeros 10 valores del arreglo c:
11.1 - 24.8 - 38.5 - 52.2 - 65.9 - 79.6 - 93.3 - 107 - 120.7 - 134.4 -

C:\Users\user\source\repos\ProyectosOMP\pruebaOMP\x64\Debug\pruebaOMP.exe (process 25820) exited with
Press any key to close this window . . .
```

Tercera ejecución

Valores:

```
//Definimos variables
#define N 10
#define chunk 2
#define mostrar 5
```

Resultado:

```
Microsoft Visual Studio Debug Console
Sumando Arreglos en Paralelo!
Imprimiendo los primeros 5 valores del arreglo a:
0 - 10 - 20 - 30 - 40 -
Imprimiendo los primeros 5 valores del arreglo b:
11.1 - 14.8 - 18.5 - 22.2 - 25.9 -
Imprimiendo los primeros 5 valores del arreglo c:
11.1 - 24.8 - 38.5 - 52.2 - 65.9 -

C:\Users\user\source\repos\ProyectosOMP\pruebaOMP\x64\Debug\pruebaOMP.exe (process 20804) exited with code
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically cl
le when debugging stops.
Press any key to close this window . . .
```

Explicación del código

1- Se declaran las librerías a utilizar, donde destaca el uso de la librería **OpenMP.**

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

2- Se declarar variables definidas N, chunk y mosta, donde:

- N – es la cantidad de elementos contenidos los arreglos.
- chunk – es el tamaño de la sección que manejará cada hilo.
- mostrar - es la cantidad de elementos a mostrar en la consola.

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

3- Se declara la función para imprimir elementos de los arreglos, el cual se utilizará más adelante.

```
void imprimeArreglo(float* d);
```

4- Se inicia el programa principal, se imprime el encabezado del proyecto y se declaran los arreglos, así como una variable auxiliar “i”.

```
int main()
{
    //Imprimimos mensaje y declaramos variables
    std::cout << "Sumando Arreglos en Paralelo!\n";
    float a[N], b[N], c[N];
    int i;
```

5- Se generan los valores de los elementos de los arreglos a y b.

```
for (i = 0; i < N; i++)
{
    a[i] = i * 10;
    b[i] = (i + 3) * 3.7;
}
```

6- Se declara variable y se le da valor de variable definida chunk.

```
int pedazos = chunk;
```

7- Se define el código para generar el procesamiento paralelo.

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

8- Se realiza la suma de los elementos de cada arreglo y se almacenan en el arreglo c.

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

9- Se muestran los encabezados de cada arreglo, se manda a llamar la función para imprimir los elementos de cada arreglo, y termina el programa principal.

```
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);

}
```

10- Se declara la función que muestra los elementos de cada arreglo.

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

Código completo:

```
//Importamos las librerias
#include <iostream>
#include <omp.h>

//Definimos variables
#define N 10
#define chunk 2
#define mostrar 5

void imprimeArreglo(float* d);

//Iniciamos programa principal
int main()
{
    //Imprimimos mensaje y declaramos variables
    std::cout << "Sumando Arreglos en Paralelo!\n";
    float a[N], b[N], c[N];
    int i;

    //Generamos los valores de los arreglos a y b
    for (i = 0; i < N; i++)
    {
        a[i] = i * 10;
        b[i] = (i + 3) * 3.7;
    }


    int pedazos = chunk;

    //generamos el procesamiento paralelo
    #pragma omp parallel for \
    shared(a, b, c, pedazos) private(i) \
    schedule(static, pedazos)

    //Realizamos la suma de arreglos
    for (i = 0; i < N; i++)
        c[i] = a[i] + b[i];

    //Imprimimos y mandamos a llamar funcion para mostrar los elementos de los arreglos
    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);
}

//Funcion para mostrar los valores de cada elemento del arreglo
void imprimeArreglo(float* d)
{
    for (int x = 0; x < mostrar; x++)
        std::cout << d[x] << " - ";
    std::cout << std::endl;
}
```

Reflexión sobre la programación paralela

La programación paralela es una técnica especialmente útil cuando se esta trabajando con grandes cantidades de información, ya que procesar la información de manera secuencial, nos estamos limitando a ejecutar paso a paso los algoritmos, en ves de dividirlos en secciones. Bien dice el dicho “divide y vencerás”.

Sin embargo, al utilizar esta técnica hay algunas consideraciones y desventajas que se deben tomar en cuenta como lo es: mayor consumo de energía ya que estamos utilizando varias unidades de procesamiento en vez de solo una, dificultad para lograr una buena sincronización y comunicación entre las tareas, costos elevados debido a lo que implica su producción y mantenimiento, las condiciones de carrera, etc.

La computación paralela es considerada importante para el ámbito de la investigación científica, más específicamente en las simulaciones, donde la gran cantidad de cálculos y operaciones complejas exigen una gran capacidad de procesamiento. Otra de las aplicaciones que podemos encontrar para la computación paralela es en la creación de modelos matemáticos y estadísticos.

Bibliografía

[1] Cendejas E. (s.f.) PARALELISMO Y ALGORITMOS PARALELOS. Instituto Tecnológico y de Estudios Superiores de Monterrey.

[2] ferestrepoca (s.f.) Programación Paralela. . Recuperado el 29 de enero del 2023 de http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html

[3] Microsoft. (2022). Algoritmos paralelos. Recuperado el 29 de enero del 2023 de https://learn.microsoft.com/es-es/cpp/parallel/concrt/parallel-algorithms?view=msvc-170

[4] Ortiz, J. (2022, 9 diciembre).La computación paralela: alta capacidad de procesamiento. Teldat. Recuperado el 29 de enero del 2023 de https://www.teldat.com/es/blog/computacion-paralela-capacidad-procesamiento/