TC4031.10

Cómputo en la nube (Gpo 10)

Tarea 1. Programación de una solución paralela

Jaime Alejandro Mendivil A. A01253316

1 de Febrero del 2026

Introducción

Para esta actividad se hará uso de programación paralela para hacer la suma de 2 arreglos de números con las siguientes características:

1. El programa inicializa dos arreglos numéricos, ya sea generando los valores internamente o cargándolos desde archivos de texto externos, según la opción seleccionada por el usuario.
2. Una vez preparados los arreglos de entrada, el programa realiza la suma elemento a elemento utilizando OpenMP.
3. Cada iteración del ciclo paralelo identifica el hilo que procesa un índice específico del arreglo.
4. Los resultados de la suma se almacenan en un tercer arreglo que contiene la suma de los elementos correspondientes de los arreglos de entrada.
5. Al finalizar el cálculo, los resultados se muestran en una tabla formateada que incluye el índice del arreglo, el número de hilo, los valores de los arreglos de entrada y el resultado de la operación.

Index	Thread	# Array A	# Array B	Result
0	1	1	2	3

Liga a Github

Proyecto en funcionamiento

```
C:\Users\YOLOA\OneDrive\Desktop\Cloud Computing\Assignment_1>ParallelArrays
Choose how to initialize the arrays:
1) Generate arrays in the program
2) Load arrays from two text files (arrayA.txt, arrayB.txt)
Enter option (1 or 2): 2
```

Aquí se observa como se visualiza la pantalla al preguntar al usuario qué opción prefiere para hacer la suma de los arreglos, generar el arreglo de forma automática o proporcionar los arreglos mediante 2 archivos de texto.

```
C:\Users\YOLOA\OneDrive\Desktop\Cloud Computing\Assignment_1>ParallelArrays
Choose how to initialize the arrays:
1) Generate arrays in the program
2) Load arrays from two text files (arrayA.txt, arrayB.txt)
Enter option (1 or 2): 2

Index   Thread    # Array A       # Array B       Result
------------------------------------------------------------
100     1         184.00          902.00          1086.00
101     1         620.00          189.00          809.00
102     1         127.00          869.00          996.00
103     1         986.00          469.00          1455.00
104     1         115.00          155.00          270.00
105     1         467.00          540.00          1007.00
106     1         276.00          962.00          1238.00
107     1         747.00          687.00          1434.00
108     1         385.00          115.00          500.00
109     1         285.00          976.00          1261.00
110     1         324.00          789.00          1113.00
111     1         457.00          984.00          1441.00
112     1         628.00          188.00          816.00
113     1         197.00          982.00          1179.00
114     1         452.00          124.00          576.00
115     1         483.00          785.00          1268.00
116     1         185.00          74.00           259.00
117     1         474.00          957.00          1431.00
118     1         160.00          37.00           197.00
```

Al seleccionar la opción 2 se muestra el siguiente resultado junto con los demás números en el arreglo.

```
C:\Users\YOLOA\OneDrive\Desktop\Cloud Computing\Assignment_1>ParallelArrays
Choose how to initialize the arrays:
1) Generate arrays in the program
2) Load arrays from two text files (arrayA.txt, arrayB.txt)
Enter option (1 or 2): 1

Index   Thread    # Array A       # Array B       Result
------------------------------------------------------------
100     1         1000.00         381.10          1381.10
101     1         1010.00         384.80          1394.80
102     1         1020.00         388.50          1408.50
103     1         1030.00         392.20          1422.20
104     1         1040.00         395.90          1435.90
105     1         1050.00         399.60          1449.60
106     1         1060.00         403.30          1463.30
107     1         1070.00         407.00          1477.00
108     1         1080.00         410.70          1490.70
109     1         1090.00         414.40          1504.40
110     1         1100.00         418.10          1518.10
111     1         1110.00         421.80          1531.80
```

Si se selecciona la opción 1 queda de la siguiente forma.

Explicación del código y los resultados

```cpp
#include <iostream>
#include <fstream>
#include <iomanip>   // setw, left, fixed, setprecision
#include <omp.h>

#define N 1000
#define CHUNK 100

// Reads exactly N numbers (one per line or space-separated) into array.
// Returns false if the file can't be opened or doesn't contain enough values.
bool readArrayFromFile(const std::string& filename, float* array)
{
    std::ifstream file(filename);
    if (!file.is_open()) return false;

    for (int i = 0; i < N; i++)
    {
        if (!(file >> array[i])) return false;
    }
    return true;
}

```

Se definen las librerías a utilizar en el proyecto así como variables para definir la cantidad de números a procesar y el tamaño del chunk por hilo. Adicionalmente se crea una variable para leer el contenido de los archivos de texto que contienen el arreglo.

```cpp
int main()
{
    float a[N], b[N], c[N];
    int choice;

    // -------- MENU: choose data source --------
    std::cout << "Choose how to initialize the arrays:\n";
    std::cout << "1) Generate arrays in the program\n";
    std::cout << "2) Load arrays from two text files (arrayA.txt, arrayB.txt)\n";
    std::cout << "Enter option (1 or 2): ";
    std::cin >> choice;

    if (choice == 1)
    {
        // Generate arrays programmatically
        for (int i = 0; i < N; i++)
        {
            a[i] = i * 10.0f;
            b[i] = (i + 3) * 3.7f;
        }
    }
    else if (choice == 2)
    {
        // Load arrays from text files
        if (!readArrayFromFile("arrayA.txt", a))
        {
            std::cerr << "Error: Could not read arrayA.txt (need " << N << " numbers).\n";
            return 1;
        }
        if (!readArrayFromFile("arrayB.txt", b))
        {
            std::cerr << "Error: Could not read arrayB.txt (need " << N << " numbers).\n";
            return 1;
        }
    }
    else
    {
        std::cerr << "Invalid option. Please choose 1 or 2.\n";
        return 1;
    }
```

Bucle principal que empieza declarando los arreglos así como una variable para la decisión del usuario de leer los archivos o generar el arreglo.

Si la decisión es 1 se genera el arreglo de forma no aleatoria. Si es 2 lee los 2 archivos así como qué hacer en caso de que tenga algún problema.

```
63
64      // -------- PRINT TABLE HEADER --------
65      std::cout << "\n"
66                << std::left
67                << std::setw(8)  << "Index"
68                << std::setw(10) << "Thread"
69                << std::setw(16) << "# Array A"
70                << std::setw(16) << "# Array B"
71                << std::setw(12) << "Result"
72                << "\n";
73
74      std::cout << std::string(62, '-') << "\n";
75
76      // -------- PARALLEL SUM + TABLE ROWS --------
77      // NOTE: Printing in parallel requires synchronization to avoid mixed output.
78      #pragma omp parallel for schedule(static, CHUNK)
79      for (int i = 0; i < N; i++)
80      {
81          int tid = omp_get_thread_num();
82          c[i] = a[i] + b[i];
83
84          #pragma omp critical
85          {
86              std::cout << std::left
87                        << std::setw(8)  << i
88                        << std::setw(10) << tid
89                        << std::setw(16) << std::fixed << std::setprecision(2) << a[i]
90                        << std::setw(16) << std::fixed << std::setprecision(2) << b[i]
91                        << std::setw(12) << std::fixed << std::setprecision(2) << c[i]
92                        << "\n";
93          }
94      }
95
96      return 0;
97  }
```

Se imprime en encabezado de la tabla teniendo en cuenta el ancho del parámetro (std::setw()) que dependiendo del tamaño de la columna tendrá un número mayor para verse más estético.

La línea 78 nos sirve para indicar al compilador que OpenMP paraleliza el ciclo For así como la cantidad de números por hilo (CHUNK). Después se reparte la operación en los distintos hilos, separando en cantidades iguales los arreglos para así sumar los números con sus respectivos índices. Esta es la operación principal que suma los números.

En la línea 84 se serializa la impresión de los números, ya que imprimir desde varios hilos puede ocasionar que los resultados se sobrepongan en la terminal, todo esto siguiendo el formato del encabezado. La operación de sumar se hace de forma paralela pero la impresión de los resultados se hace de forma serial.

```
 98
 99    // g++ -fopenmp  ParallelArrays.cpp -o ParallelArrays
100
101    // ./ParallelArrays
```

Comentarios que se copian para compilar el código en la terminal así como correr el ejecutable.

Reflexión sobre la programación paralela

Esta actividad fue de mucha utilidad para observar como es el proceso de conversión de un algoritmo de suma de números, para ser implementado en varios hilos, cosa que es de vital importancia para reducir el tiempo en el que se ejecuta el algoritmo. Esto conlleva bastante reflexión para interpretar un problema y observar las partes en las que se puede separar para que se ejecuten en varios procesadores sin que afecte de forma negativa los resultados y detectar cuáles operaciones no se pueden paralelizar.

A pesar de ser más complicado que realizar un algoritmo sencillo que se ejecuta en un procesador, a la hora de incrementar el volumen de las operaciones se vuelve indispensable paralelizar los programas para ser implementados en mayor escala, cosa que es muy importante para el cómputo en la nube que conlleva billones de operaciones para millones de usuarios.

Este pequeño ejercicio permite adentrarse en cómo visualizar los problemas de forma que se puedan separar en múltiples partes y cuales deben de mantenerse en una pieza.

Bibliografía

1. OpenMP Architecture Review Board. (2023). OpenMP Application Programming Interface, Version 5.2.
   https://www.openmp.org/specifications/

2. Chapman, B., Jost, G., & van der Pas, R. (2007). Using OpenMP: Portable Shared Memory Parallel Programming. MIT Press.

3. Quinn, M. J. (2004). Parallel Programming in C with MPI and OpenMP. McGraw-Hill.

4. Dagum, L., & Menon, R. (1998). OpenMP: An industry standard API for shared-memory programming. IEEE Computational Science and Engineering, 5(1), 46–55.
   https://doi.org/10.1109/99.660313

5. GCC Team. (2024). Using the GNU Compiler Collection (GCC): OpenMP.
   https://gcc.gnu.org/onlinedocs/libgomp/

6. Stroustrup, B. (2013). The C++ Programming Language (4th ed.). Addison-Wesley.

7. Pacheco, P. (2011). An Introduction to Parallel Programming. Morgan Kaufmann.