



**Tecnológico
de Monterrey**

Escuela de Ingeniería y Ciencias

Cómputo en la Nube

Programación de una Solución Paralela

Armando Bringas Corpus

A01200230

1 Introducción

La siguiente práctica tiene como objetivo el implementar un algoritmo paralelo para resolver un problema numérico. De forma particular para la suma de dos arreglos donde se aplicará programación paralela donde la suma de algunos de los elementos de los arreglos se puedan realizar en un hilo, otra porción en otro hilo y así sucesivamente empleando la librería OMP, realizando el código en lenguaje de programación C++ y ejecutándolo a través del entorno de desarrollo Visual Studio.

2 Liga al repositorio

- Repositorio General: https://github.com/armandoBringas/Cloud_Computing
- Código original de la actividad: https://github.com/armandoBringas/Cloud_Computing/tree/main/M2_Modelos_de_Programaci%C3%B3n/PruebaOMP_original
- Código refactorizado de la actividad: https://github.com/armandoBringas/Cloud_Computing/tree/main/M2_Modelos_de_Programaci%C3%B3n/PruebaOMP_refactorizado

3 Capturas de pantalla

Se realizó la práctica con el entorno de desarrollo integrado (IDE) Microsoft Visual Studio Enterprise 2022 (64-bit) en el cual se configuró un proyecto en el cual se activo la compatibilidad con la librería Open MP para permitir gestionar e interpretar directivas e instrucciones de hilos y paralelismo.

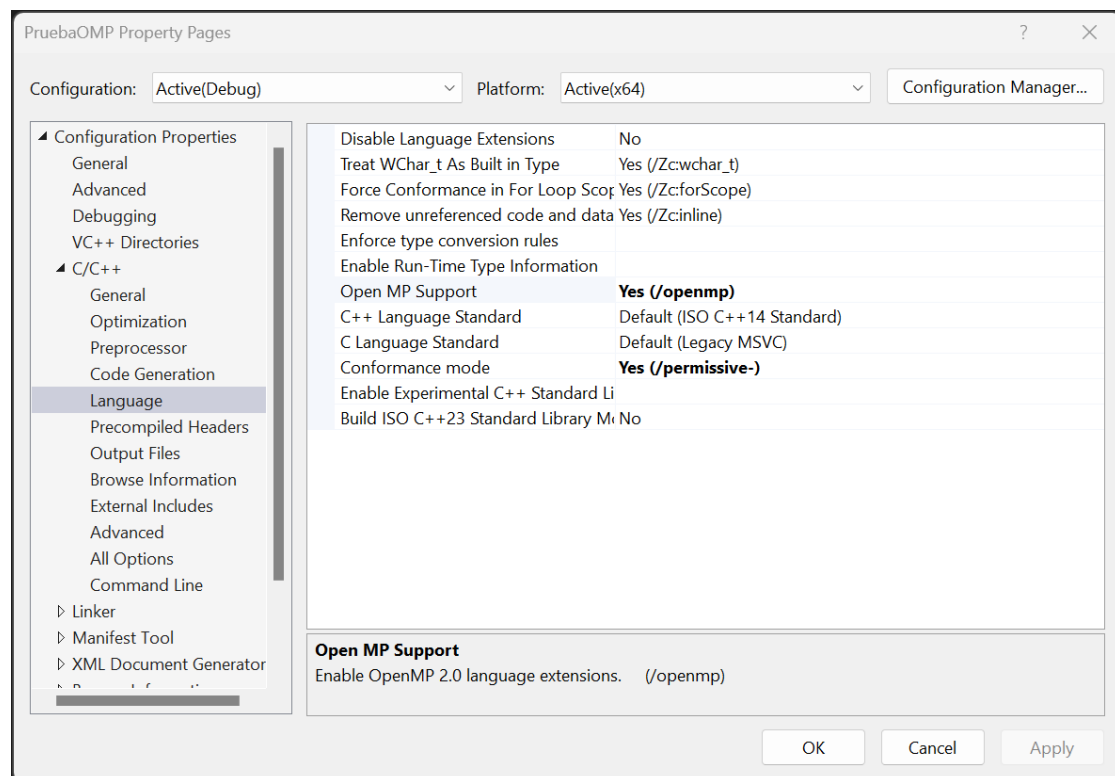


Figura 1: Configuración de librería OpenMP en Visual Studio

Con el código original de la actividad utilizando los siguientes parámetros se obtuvieron los siguientes resultados:

Ejemplo 1 - Código original:

```
#define N 1000
#define chunk 150
#define mostrar 20
```

```
Microsoft Visual Studio Debu... X + - □ X
Sumando Arreglos en Paralelo!
Imprimiendo los primeros 20 valores del arreglo a:
0 - 10 - 20 - 30 - 40 - 50 - 60 - 70 - 80 - 90 - 100 - 110 - 120 - 130 - 140 - 150 - 160 - 170 - 180 - 190 -
Imprimiendo los primeros 20 valores del arreglo b:
11.1 - 14.8 - 18.5 - 22.2 - 25.9 - 29.6 - 33.3 - 37 - 40.7 - 44.4 - 48.1 - 51.8 - 55.5 - 59.2 - 62.9 - 66.6 - 70.3 - 74
- 77.7 - 81.4 -
Imprimiendo los primeros 20 valores del arreglo c:
11.1 - 24.8 - 38.5 - 52.2 - 65.9 - 79.6 - 93.3 - 107 - 120.7 - 134.4 - 148.1 - 161.8 - 175.5 - 189.2 - 202.9 - 216.6 - 2
30.3 - 244 - 257.7 - 271.4 -

C:\Users\bring\Documents\Github\Cloud_Computing\M2_Modelos_de_Programación\PruebaOMP_original\x64\Debug\PruebaOMP.exe (p
rocess 15164) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Figura 2: Salida del Ejemplo 1 - Código original

Ejemplo 2 - Código original:

```
#define N 20000
#define chunk 500
#define mostrar 75
```

```
Microsoft Visual Studio Debu... X + - □ X
Sumando Arreglos en Paralelo!
Imprimiendo los primeros 75 valores del arreglo a:
0 - 10 - 20 - 30 - 40 - 50 - 60 - 70 - 80 - 90 - 100 - 110 - 120 - 130 - 140 - 150 - 160 - 170 - 180 - 190 - 200 - 210 -
220 - 230 - 240 - 250 - 260 - 270 - 280 - 290 - 300 - 310 - 320 - 330 - 340 - 350 - 360 - 370 - 380 - 390 - 400 - 410 -
420 - 430 - 440 - 450 - 460 - 470 - 480 - 490 - 500 - 510 - 520 - 530 - 540 - 550 - 560 - 570 - 580 - 590 - 600 - 610 -
620 - 630 - 640 - 650 - 660 - 670 - 680 - 690 - 700 - 710 - 720 - 730 - 740 -
Imprimiendo los primeros 75 valores del arreglo b:
11.1 - 14.8 - 18.5 - 22.2 - 25.9 - 29.6 - 33.3 - 37 - 40.7 - 44.4 - 48.1 - 51.8 - 55.5 - 59.2 - 62.9 - 66.6 - 70.3 - 74
- 77.7 - 81.4 - 85.1 - 88.8 - 92.5 - 96.2 - 99.9 - 103.6 - 107.3 - 111 - 114.7 - 118.4 - 122.1 - 125.8 - 129.5 - 133.2 -
136.9 - 140.6 - 144.3 - 148 - 151.7 - 155.4 - 159.1 - 162.8 - 166.5 - 170.2 - 173.9 - 177.6 - 181.3 - 185 - 188.7 - 192
.4 - 196.1 - 199.8 - 203.5 - 207.2 - 210.9 - 214.6 - 218.3 - 222 - 225.7 - 229.4 - 233.1 - 236.8 - 240.5 - 244.2 - 247.9
- 251.6 - 255.3 - 259 - 262.7 - 266.4 - 270.1 - 273.8 - 277.5 - 281.2 - 284.9 -
Imprimiendo los primeros 75 valores del arreglo c:
11.1 - 24.8 - 38.5 - 52.2 - 65.9 - 79.6 - 93.3 - 107 - 120.7 - 134.4 - 148.1 - 161.8 - 175.5 - 189.2 - 202.9 - 216.6 - 2
30.3 - 244 - 257.7 - 271.4 - 285.1 - 298.8 - 312.5 - 326.2 - 339.9 - 353.6 - 367.3 - 381 - 394.7 - 408.4 - 422.1 - 435.8
- 449.5 - 463.2 - 476.9 - 490.6 - 504.3 - 518 - 531.7 - 545.4 - 559.1 - 572.8 - 586.5 - 600.2 - 613.9 - 627.6 - 641.3 -
655 - 668.7 - 682.4 - 696.1 - 709.8 - 723.5 - 737.2 - 750.9 - 764.6 - 778.3 - 792 - 805.7 - 819.4 - 833.1 - 846.8 - 860
.5 - 874.2 - 887.9 - 901.6 - 915.3 - 929 - 942.7 - 956.4 - 970.1 - 983.8 - 997.5 - 1011.2 - 1024.9 -

C:\Users\bring\Documents\Github\Cloud_Computing\M2_Modelos_de_Programación\PruebaOMP_original\x64\Debug\PruebaOMP.exe (p
rocess 10388) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Figura 3: Salida del Ejemplo 2 - Código original

Se realizaron algunas refactorizaciones del código original de la actividad que en la siguiente sección serán explicadas. En este código refactorizado se implementó la opción mediante una macro de ejecución el poder correr los valores de los parámetros definidos dentro del código o que estos se seleccionaran de forma aleatoria dentro de un rango definido.

Rangos seleccionados para los ejemplos usando el código refactorizado:

```
const int N = numeroAleatorio(50000, 60000);
const int CHUNK_SIZE = numeroAleatorio(700, 1000);
const int MOSTRAR = numeroAleatorio(10, 25);
```

Con el código refactorizado de la actividad se obtuvieron los siguientes resultados:

Ejemplo 1 - Código refactorizado:

```
Sumando Arreglos en Paralelo!
N = 57281
CHUNK_SIZE = 960
MOSTRAR = 30

Imprimiendo los primeros 30 valores del arreglo a:
0 - 10 - 20 - 30 - 40 - 50 - 60 - 70 - 80 - 90 - 100 - 110 - 120 - 130 - 140 - 150 - 160 - 170 - 180 - 190 - 200 - 210 -
220 - 230 - 240 - 250 - 260 - 270 - 280 - 290 -

Imprimiendo los primeros 30 valores del arreglo b:
11.1 - 14.8 - 18.5 - 22.2 - 25.9 - 29.6 - 33.3 - 37 - 40.7 - 44.4 - 48.1 - 51.8 - 55.5 - 59.2 - 62.9 - 66.6 - 70.3 - 74 -
77.7 - 81.4 - 85.1 - 88.8 - 92.5 - 96.2 - 99.9 - 103.6 - 107.3 - 111 - 114.7 - 118.4 -

Imprimiendo los primeros 30 valores del arreglo c:
11.1 - 24.8 - 38.5 - 52.2 - 65.9 - 79.6 - 93.3 - 107 - 120.7 - 134.4 - 148.1 - 161.8 - 175.5 - 189.2 - 202.9 - 216.6 - 230.3 -
244 - 257.7 - 271.4 - 285.1 - 298.8 - 312.5 - 326.2 - 339.9 - 353.6 - 367.3 - 381 - 394.7 - 408.4 -

C:\Users\bring\Documents\Github\Cloud_Computing\M2_Modelos_de_Programación\PruebaOMP_refactorizado\x64\Debug\PruebaOMP.exe (process 28512) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Figura 4: Salida del Ejemplo 1 - Código refactorizado

Ejemplo 2 - Código refactorizado:

```
N = 57634
CHUNK_SIZE = 761
MOSTRAR = 38

Imprimiendo los primeros 38 valores del arreglo a:
0 - 10 - 20 - 30 - 40 - 50 - 60 - 70 - 80 - 90 - 100 - 110 - 120 - 130 - 140 - 150 - 160 - 170 - 180 - 190 - 200 - 210 -
220 - 230 - 240 - 250 - 260 - 270 - 280 - 290 - 300 - 310 - 320 - 330 - 340 - 350 - 360 - 370 -

Imprimiendo los primeros 38 valores del arreglo b:
11.1 - 14.8 - 18.5 - 22.2 - 25.9 - 29.6 - 33.3 - 37 - 40.7 - 44.4 - 48.1 - 51.8 - 55.5 - 59.2 - 62.9 - 66.6 - 70.3 - 74 -
77.7 - 81.4 - 85.1 - 88.8 - 92.5 - 96.2 - 99.9 - 103.6 - 107.3 - 111 - 114.7 - 118.4 - 122.1 - 125.8 - 129.5 - 133.2 -
136.9 - 140.6 - 144.3 - 148 -

Imprimiendo los primeros 38 valores del arreglo c:
11.1 - 24.8 - 38.5 - 52.2 - 65.9 - 79.6 - 93.3 - 107 - 120.7 - 134.4 - 148.1 - 161.8 - 175.5 - 189.2 - 202.9 - 216.6 - 230.3 -
244 - 257.7 - 271.4 - 285.1 - 298.8 - 312.5 - 326.2 - 339.9 - 353.6 - 367.3 - 381 - 394.7 - 408.4 - 422.1 - 435.8 -
449.5 - 463.2 - 476.9 - 490.6 - 504.3 - 518 -

C:\Users\bring\Documents\Github\Cloud_Computing\M2_Modelos_de_Programación\PruebaOMP_refactorizado\x64\Debug\PruebaOMP.exe (process 18824) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Figura 5: Salida del Ejemplo 2 - Código refactorizado

4 Explicación del código y resultados

Código original

Por medio del código que se implementó en la práctica se buscó ejemplificar el uso del **paralelismo a nivel de instrucción** en la cual a través de la librería OpenMP buscamos ejecutar más de una instrucción a la vez, es decir, en paralelo.

Al inicio del código se definen tres directivas del preprocesador (macros) las cuales serán empleadas para modificar la distribución de la carga entre los hilos del procesador:

```
#define N 20000
#define chunk 500
#define mostrar 75
```

En donde,

- **N**: Es la cantidad de elementos que se manejarán en los arreglos.
- **chunk**: El tamaño de los pedazos de los arreglos para que cada hilo creado maneje esta cantidad específica de elementos.
- **mostrar**: Es la cantidad de los elementos a imprimir de la operación que se hará con los arreglos

Posteriormente se define el prototipo de la función que imprimirá los arreglos que se definan en el programa, esta función recibe un puntero el cual hace referencia al arreglo que se le da de entrada.

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

En nuestra función `main()` la cual es la rutina principal del programa se hace la definición de los arreglos en los cuales los arreglos `a` y `b` se les asignarán los valores a cada uno de sus elementos por medio de un bucle y una operación establecida para facilitar la población de los arreglos, el arreglo `c` es el que contendrá el resultado de la suma de los arreglos `a` y `b`. El tamaño de cada uno de los arreglos estará establecido por constante `N`.

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

```

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

```

En la siguiente porción de código se inicia el proceso de paralelización a través de la librería **OpenMP**, con la cual el bucle `for` sera el que se esté paralelizando. De igual manera, se establece la información que se compartirá entre las diversas particiones a través de la función `shared` la cual recibirá como parámetros los arreglos y la variable `pedazos` a la cual se le asigna como valor la constante `chunk`, esta variable se encargará de guardar el valor de cómo será repartida la información a través de los hilos que sean creados. Es importante resaltar que la variable `i` se debe mantener privada para que los ciclos no se puedan mezclar con otros hilos. De igual manera establecemos el `schedule` como estático para que siempre sea ejecutado de la misma manera.

```

int pedazos = chunk;

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

```

A continuación hacemos uso de la función `imprimeArreglo()` para imprimir los arreglos con el número de elementos que se definió en la constante `mostrar`.

```

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;
}

```

Finalmente se tiene la función que imprime cada uno de los elementos del arreglo que recibe como puntero.

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

Código refactorizado

Como se comentó en la sección anterior se realizó un código en el cual se efectuaron algunas refactorizaciones para tener a través de una macro de ejecución el poder ejecutar los valores de los parámetros `N`, `chunk`, `mostrar` de forma aleatoria dentro de un rango definido o de una forma establecida. De igual manera se separaron los prototipos para ser definidos en un archivo de cabecera y se buscó modularizar el código para en futuro de ser reutilizado poder más fácil poder depurarlo al tener funciones separadas.

Otro cambio importante es que tuvimos que cambiar la definición de los arreglos como vectores para poder tener arreglos dinámicos y evitar errores en la compilación y ejecución del código.

```
#ifndef PRUEBA_OMP_H
#define PRUEBA_OMP_H

void imprimeArreglo(const std::vector<float>& d);
void sumaArreglosParalelo(const std::vector<float>& a,
                          const std::vector<float>& b,
                          std::vector<float>& c,
                          int size,
                          int chunk);
int numeroAleatorio(int min_num, int max_num);

#endif
```

```
#include <iostream>
#include <omp.h>
#include <stdio.h>
#include <vector>
#include "PruebaOMP.h"
```



```

// Comentar para usar valores predefinidos
#define USE_RANDOM_VALUES

#ifdef USE_RANDOM_VALUES
const int N = numeroAleatorio(50000, 60000);
const int CHUNK_SIZE = numeroAleatorio(700, 1000);
const int MOSTRAR = numeroAleatorio(20, 40);
#else
const int N = 1000;
const int CHUNK_SIZE = 100;
const int MOSTRAR = 10;
#endif

int main()
{
    std::cout << "Sumando Arreglos en Paralelo!\n";
    std::cout << std::endl;
    std::cout << "N = " << N << std::endl;
    std::cout << "CHUNK_SIZE = " << CHUNK_SIZE
                << std::endl;
    std::cout << "MOSTRAR = " << MOSTRAR
                << std::endl;
    std::cout << std::endl;

    std::vector<float> a(N), b(N), c(N);

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

    sumaArreglosParalelo(a, b, c, N, CHUNK_SIZE);

    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(const std::vector<float>& d)
{
    for (int x = 0; x < MOSTRAR && x < d.size(); x++)
        std::cout << d[x] << " - ";

    std::cout << std::endl;
    std::cout << std::endl;
}

void sumaArreglosParalelo(const std::vector<float>& a,
                          const std::vector<float>& b,
                          std::vector<float>& c, int n,
                          int chunk)
{
    #pragma omp parallel for \
        shared(a, b, c, chunk) \
        schedule(static, chunk)
    for (int i = 0; i < n; i++)
        c[i] = a[i] + b[i];
}

int numeroAleatorio(int min_num, int max_num)
{
    // Ensure a valid range
    if (min_num >= max_num) {
        std::swap(min_num, max_num);
    }

    static bool seed_initialized = false;
    if (!seed_initialized) {
        srand(static_cast<unsigned>(time(nullptr)));
        seed_initialized = true;
    }

    return rand() % (max_num - min_num + 1) + min_num;
}

```

Resultados

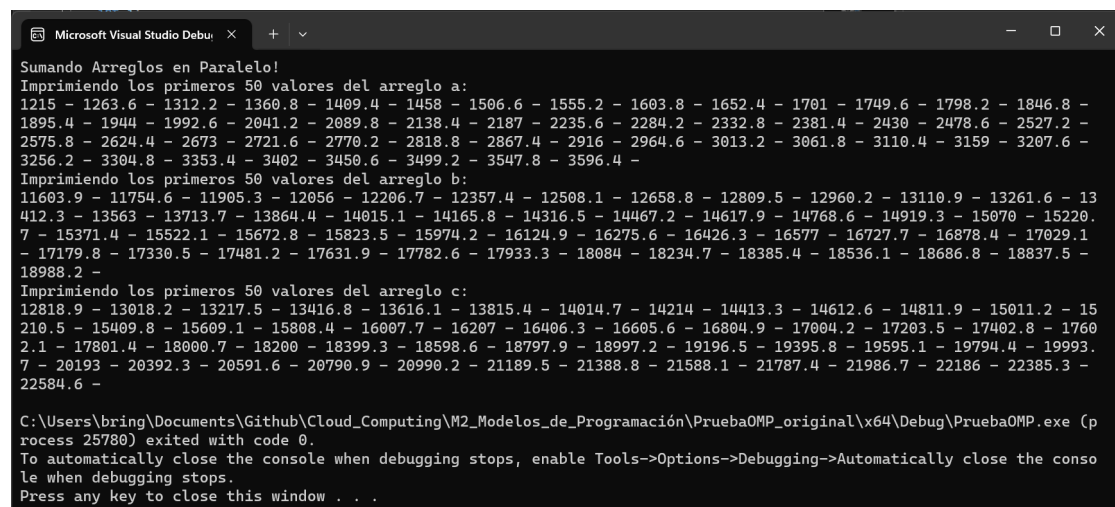
Para experimentar tanto con el código original como con el que se refactorizó, vamos a modificar en ambos las ecuaciones para poblar los arreglos **a** y **b**, en este caso quedan definidos como:

```
for (int i = 0; i < N; i++)
{
    a[i] = (i + 25) * 48.6; // (i + 2) * 10;
    b[i] = (i + 77) * 150.7; // (i + 3) * 3.7;
}
```

Y establecemos de igual manera en ambos códigos las constantes **N**, **chunk**, **mostrar** con los siguientes valores:

```
#define N 1000
#define chunk 500
#define mostrar 50
```

Para el código original obtenemos los siguientes resultados:



```
Microsoft Visual Studio Debu...
Sumando Arreglos en Paralelo!
Imprimiendo los primeros 50 valores del arreglo a:
1215 - 1263.6 - 1312.2 - 1360.8 - 1409.4 - 1458 - 1506.6 - 1555.2 - 1603.8 - 1652.4 - 1701 - 1749.6 - 1798.2 - 1846.8 -
1895.4 - 1944 - 1992.6 - 2041.2 - 2089.8 - 2138.4 - 2187 - 2235.6 - 2284.2 - 2332.8 - 2381.4 - 2430 - 2478.6 - 2527.2 -
2575.8 - 2624.4 - 2673 - 2721.6 - 2770.2 - 2818.8 - 2867.4 - 2916 - 2964.6 - 3013.2 - 3061.8 - 3110.4 - 3159 - 3207.6 -
3256.2 - 3304.8 - 3353.4 - 3402 - 3450.6 - 3499.2 - 3547.8 - 3596.4 -
Imprimiendo los primeros 50 valores del arreglo b:
11603.9 - 11754.6 - 11905.3 - 12056 - 12206.7 - 12357.4 - 12508.1 - 12658.8 - 12809.5 - 12960.2 - 13110.9 - 13261.6 - 13
412.3 - 13563 - 13713.7 - 13864.4 - 14015.1 - 14165.8 - 14316.5 - 14467.2 - 14617.9 - 14768.6 - 14919.3 - 15070 - 15220.
7 - 15371.4 - 15522.1 - 15672.8 - 15823.5 - 15974.2 - 16124.9 - 16275.6 - 16426.3 - 16577 - 16727.7 - 16878.4 - 17029.1
- 17179.8 - 17330.5 - 17481.2 - 17631.9 - 17782.6 - 17933.3 - 18084 - 18234.7 - 18385.4 - 18536.1 - 18686.8 - 18837.5 -
18988.2 -
Imprimiendo los primeros 50 valores del arreglo c:
12818.9 - 13018.2 - 13217.5 - 13416.8 - 13616.1 - 13815.4 - 14014.7 - 14214 - 14413.3 - 14612.6 - 14811.9 - 15011.2 - 15
210.5 - 15409.8 - 15609.1 - 15808.4 - 16007.7 - 16207 - 16406.3 - 16605.6 - 16804.9 - 17004.2 - 17203.5 - 17402.8 - 1760
2.1 - 17801.4 - 18000.7 - 18200 - 18399.3 - 18598.6 - 18797.9 - 18997.2 - 19196.5 - 19395.8 - 19595.1 - 19794.4 - 19993.
7 - 20193 - 20392.3 - 20591.6 - 20790.9 - 20990.2 - 21189.5 - 21388.8 - 21588.1 - 21787.4 - 21986.7 - 22186 - 22385.3 -
22584.6 -
C:\Users\bring\Documents\Github\Cloud_Computing\M2_Modelos_de_Programación\PruebaOMP_original\x64\Debug\PruebaOMP.exe (p
rocess 25780) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Figura 6: Resultados del Código original

Para el código refactorizado obtenemos los siguientes resultados:

```
Microsoft Visual Studio Debu x + v - □ x
N = 10000
CHUNK_SIZE = 500
MOSTRAR = 50

Imprimiendo los primeros 50 valores del arreglo a:
1215 - 1263.6 - 1312.2 - 1360.8 - 1409.4 - 1458 - 1506.6 - 1555.2 - 1603.8 - 1652.4 - 1701 - 1749.6 - 1798.2 - 1846.8 -
1895.4 - 1944 - 1992.6 - 2041.2 - 2089.8 - 2138.4 - 2187 - 2235.6 - 2284.2 - 2332.8 - 2381.4 - 2430 - 2478.6 - 2527.2 -
2575.8 - 2624.4 - 2673 - 2721.6 - 2770.2 - 2818.8 - 2867.4 - 2916 - 2964.6 - 3013.2 - 3061.8 - 3110.4 - 3159 - 3207.6 -
3256.2 - 3304.8 - 3353.4 - 3402 - 3450.6 - 3499.2 - 3547.8 - 3596.4 -

Imprimiendo los primeros 50 valores del arreglo b:
11603.9 - 11754.6 - 11905.3 - 12056 - 12206.7 - 12357.4 - 12508.1 - 12658.8 - 12809.5 - 12960.2 - 13110.9 - 13261.6 - 13
412.3 - 13563 - 13713.7 - 13864.4 - 14015.1 - 14165.8 - 14316.5 - 14467.2 - 14617.9 - 14768.6 - 14919.3 - 15070 - 15220.
7 - 15371.4 - 15522.1 - 15672.8 - 15823.5 - 15974.2 - 16124.9 - 16275.6 - 16426.3 - 16577 - 16727.7 - 16878.4 - 17029.1
- 17179.8 - 17330.5 - 17481.2 - 17631.9 - 17782.6 - 17933.3 - 18084 - 18234.7 - 18385.4 - 18536.1 - 18686.8 - 18837.5 -
18988.2 -

Imprimiendo los primeros 50 valores del arreglo c:
12818.9 - 13018.2 - 13217.5 - 13416.8 - 13616.1 - 13815.4 - 14014.7 - 14214 - 14413.3 - 14612.6 - 14811.9 - 15011.2 - 15
210.5 - 15409.8 - 15609.1 - 15808.4 - 16007.7 - 16207 - 16406.3 - 16605.6 - 16804.9 - 17004.2 - 17203.5 - 17402.8 - 1760
2.1 - 17801.4 - 18000.7 - 18200 - 18399.3 - 18598.6 - 18797.9 - 18997.2 - 19196.5 - 19395.8 - 19595.1 - 19794.4 - 19993.
7 - 20193 - 20392.3 - 20591.6 - 20790.9 - 20990.2 - 21189.5 - 21388.8 - 21588.1 - 21787.4 - 21986.7 - 22186 - 22385.3 -
22584.6 -

C:\Users\bring\Documents\Github\Cloud_Computing\M2_Modelos_de_Programación\PruebaOMP_refactorizado\x64\Debug\PruebaOMP.e
xe (process 14828) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Figura 7: Resultados del Código refactorizado

Como podemos observar en ambos códigos el resultado es el mismo, con lo cual se puede corroborar que las implementaciones son adecuadas y no hay discrepancias con el código refactorizado.

5 Reflexión sobre la programación paralela

Se realizó una práctica donde se implementó un ejemplo de programación en paralelo, en este caso anivel de instrucción en el cual se fue ejecutando instrucciones independientes en forma simultánea. Se utilizó la librería OpenMP utilizando el lenguaje de programación C++ y el entorno de desarrollo Visual Studio que facilitaraon en la creación, compilación y ejecución del código.

Aunque a primera instancia el ejemplo pueda parecer trivial, es de suma importancia ya que si se tuviera el caso como el de la suma de dos arreglos que contuvieran miles de elementos se puede mejorar el tiempo de ejecución a través de la paralelización, se podría pensar en otros casos como en el contexto de Inteligencia Artificial donde se emplean algoritmos de Aprendizaje Automático en la cuál se realizan operaciones matemáticas con datos de miles o millones de elementos que no serían posibles de ejecutarlas si no fuera a través de la paralelización.

Es por ello que el uso de la programación paralela es importante en donde se procesan una gran cantidad de datos y donde se tienen que hacer un sinnúmero de

cálculos en los mismos los cuales son posibles a través de técnicas de paralelización, también la paralelización puede ofrecer la oportunidad de optimizar recursos de hardware e incluso ahorrar costos al buscar formas más eficientes de ejecutar el código.