	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Jesús Cruz Navarro
Asignatura:	Estructuras de Datos y Algoritmos 2
Grupo:	1
No. de Práctica(s):	11
Integrante(s):	Ugalde Velasco Armando
No. de Equipo de cómputo empleado:	
No. de Lista o Brigada:	32
Semestre:	2021-1
Fecha de entrega:	23 de enero de 2021
Observaciones:	

CALIFICACIÓN: _____

PRÁCTICA 11: INTRODUCCIÓN A OPENMP

Objetivo: El estudiante conocerá y aprenderá a utilizar extensiones de un lenguaje o algún lenguaje de programación paralela que le permita llevar a la programación algunos algoritmos paralelos.

Actividad 6.1

Se implementó el programa mostrado en la práctica, cuya función es sumar dos arreglos de forma secuencial.

```
void actividad6_1()
{
    int *a, *b, *c;
    a = malloc( _Size: sizeof(int) * N);
    b = malloc( _Size: sizeof(int) * N);
    c = malloc( _Size: sizeof(int) * N);

    llenaArreglo(a);
    llenaArreglo(b);
    printf( _Format: "Suma serial\n");
    suma_serial(a, b, c);
}
```

Función principal

```
void llenaArreglo(int *a)
{
    for (int i = 0; i < N; i++)
    {
        a[i] = rand() % N;
        printf( _Format: "%d\t", a[i]);
    }
    printf( _Format: "\n");
}
```

Función llenaArreglo

```c
void suma_serial(int *A, int *B, int *C)
{
    for(int i = 0; i < N; i++)
    {
        C[i] = A[i] + B[i];
        printf( _Format: "%d\t", C[i]);
    }
    printf( _Format: "\n");
}
```

Función suma_serial

A continuación, se muestra la salida del programa:

```
Actividad 6.1
1       7       4       0       9       4       8       8       2       4
5       5       1       7       1       1       5       2       7       6
Suma serial
6       12      5       7       10      5       13      10      9       10
```

Salida del programa

Actividad 6.2

Se implementó el programa mostrado en la práctica, cuya función es sumar dos arreglos de forma paralela. Además, se definieron las constantes **N = 10**, que representa el número de elementos en el arreglo, **NUM_THREADS = 2**, el número de hilos a utilizar.

```c
void actividad6_2()
{
    int *a, *b, *c;
    a = malloc( _Size: sizeof(int) * N);
    b = malloc( _Size: sizeof(int) * N);
    c = malloc( _Size: sizeof(int) * N);

    llenaArreglo(a);
    llenaArreglo(b);
    printf( _Format: "Suma paralela\n");
    suma_paralela(a, b, c);
}
```

Función principal

```c
void suma_paralela(int *A, int *B, int *C)
{
    int i, tid, inicio, fin;
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel private(inicio, fin, tid, i)
    {
        tid = omp_get_thread_num();
        inicio = tid * (N / NUM_THREADS);
        fin = (tid + 1) * (N / NUM_THREADS) - 1;
        for (i = inicio; i <= fin; i++)
        {
            C[i] = A[i] + B[i];
            printf( _Format: "Hilo %d calculo C[%d] = %d\n", tid, i, C[i]);
        }
    }
}
```

Función suma_paralela

A continuación, se muestra la salida del programa:

```
Actividad 6.2
1       4       2       3       2       2       1       6       8       5
7       6       1       8       9       2       7       9       5       4
Suma paralela
Hilo 0 calculo C[0] = 8
Hilo 0 calculo C[1] = 10
Hilo 0 calculo C[2] = 3
Hilo 0 calculo C[3] = 11
Hilo 0 calculo C[4] = 11
Hilo 1 calculo C[5] = 4
Hilo 1 calculo C[6] = 8
Hilo 1 calculo C[7] = 15
Hilo 1 calculo C[8] = 13
Hilo 1 calculo C[9] = 9
```

Salida del programa

Actividad 7

Se implementó el programa solicitado en la práctica, que consiste en realizar la misma función que en la actividad **1**, pero utilizando el constructo **omp for** para dividir las iteraciones realizadas entre los hilos utilizados.

```
void actividad7()
{
    #pragma omp parallel
    {
        printf( _Format: "Hola mundo\n");
        #pragma omp for
        for (int i = 0; i < N; i++)
        {
            printf( _Format: "Iteracion: %d\n", i);
        }
    }
    printf( _Format: "Adios\n");
}
```

Función principal

```
Actividad 7
Hola mundo
Iteracion: 5
Iteracion: 6
Iteracion: 7
Iteracion: 8
Iteracion: 9
Hola mundo
Iteracion: 0
Iteracion: 1
Iteracion: 2
Iteracion: 3
Iteracion: 4
Adios
```

Salida del programa

Actividad 8

Se implementó el programa solicitado en la práctica, que consiste en realizar la misma función que en la actividad **6.1** y **6.2**, pero utilizando el constructo **omp for**, el cual facilita en gran medida su implementación. En otras palabras, no es necesario dividir las iteraciones entre los hilos manualmente mediante índices, como se realizó en la actividad **6.2**.

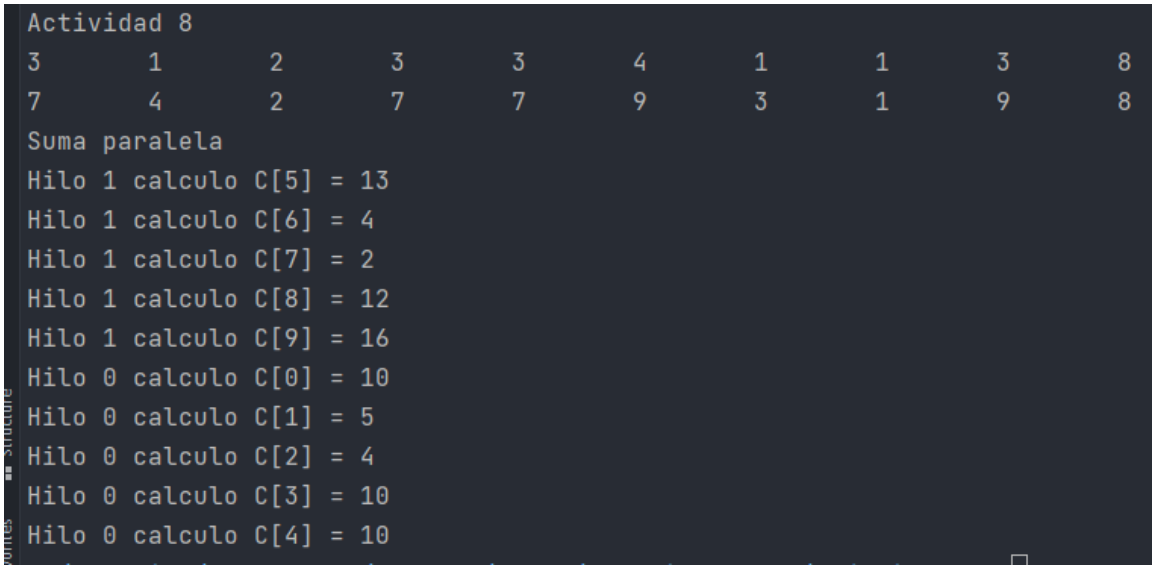
```
void actividad8()
{
    int *a, *b, *c;
    a = malloc( _Size: sizeof(int) * N);
    b = malloc( _Size: sizeof(int) * N);
    c = malloc( _Size: sizeof(int) * N);

    llenaArreglo(a);
    llenaArreglo(b);
    printf( _Format: "Suma paralela\n");
    suma_paralela_for(a, b, c);
}
```

Salida del programa

```
void suma_paralela_for(int *A, int *B, int *C)
{
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel for
    for (int i = 0; i < N; i++)
    {
        C[i] = A[i] + B[i];
        printf( _Format: "Hilo %d calculo C[%d] = %d\n", omp_get_thread_num(), i, C[i]);
    }
}
```

Salida del programa



Salida del programa

CONCLUSIONES

La librería **OpenMP** nos facilita realizar programas concurrentes al proporcionar una serie de constructos simples para cumplir con tareas comúnmente realizadas en este dominio. Un ejemplo de lo anterior es el constructo **omp for,** que, dentro de una región paralela, nos permite dividir las iteraciones entre los hilos a utilizar. Es importante resaltar la expresividad de dichos constructos, lo cual también mejora la legibilidad del código respecto a formas imperativas de realizar la misma tarea.

Es fundamental comprender su funcionamiento y forma de uso, para así aprovechar de forma óptima sus constructos y lograr implementar los algoritmos deseados correctamente.