I OBJETIVOS

- Medir y comparar la ejecución de procesos que requieren uso intensivo del CPU en arquitecturas Multi-Core.
- Utilizar la API OpenMP para la gestión de hilos, y de esta forma optimizar una aplicación que requiere uso intensivo del CPU para procesadores Multi-Core.

II BIBLIOGRAFÍA

- Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation). *Barbara Chapman, Gabriele Jost, Ruud van der Pas, and David J. Kuck* (Paperback - Oct 31, 2007).

III RECURSOS

- Una estación con Visual Studio y el compilador Intel C/C++.
- Computadoras con procesadores Multi-Core.

IV ACTIVIDADES

1 Ordenamiento de números de punto flotante en paralelo

El programa que se muestra en el Ejemplo 1 debe generar números aleatorios para almacenarlos en un arreglo, ordenarlos, y posteriormente comprobar si los números en el arreglo están debidamente ordenados. Este programa debe ser rápido y eficiente con un volumen de datos grande, por ejemplo 20 millones de números y la función `sortArr` no está implementada.

Lo que tú debes hacer en este programa es implementar un algoritmo de ordenamiento paralelo utilizando las directivas de OpenMP en la función `sortArr`.

1.1 Verificación del algoritmo implementado

Utiliza las herramientas de Parallel Studio para asegurarte que:

a) No haya errores ocasionados por condiciones de concurso o interbloqueo en la paralelización.
b) El rendimiento de la función `sortArr` sea el óptimo y no existan problemas ocasionados por esperas innecesarias en objetos de sincronización y balanceo de carga.

Guarda los resultados que obtengas de Parallel Studio y documenta los cambios que hiciste para hacer correcciones y mejoras en el rendimiento.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define ELEMS 20000000 // Elementos a ordenar

int errors(int *numbers,int elems);
void initArr(int *numbers,int elems);
void SortArr(int *numbers);

int arr[ELEMS];

int main()
{
	int i;
	int n;
	clock_t start,stop;

	initArr(arr,ELEMS);

	start=clock();

	SortArr(arr);

	stop=clock();
	if((n=errors(arr,ELEMS)))
		printf("Se encontrarin %d errores\n",n);
	else
		printf("%d elementos ordenados en %1.2f
segundos\n",ELEMS,((float)stop-(float)start)/1000.0);
}

void initArr(int *numbers,int elems)
{
	int i;
	for(i=0;i<elems;i++)
		numbers[i]=rand()*rand()%ELEMS;

}

int errors(int *numbers,int elems)
{
	int i;
	int errs=0;
```

```
	for(i=0;i<elems-1;i++)
		if(numbers[i]>numbers[i+1])
			errs++;
	return(errs);
}


void SortArr(int *numbers)
{
	// Aquí va tu función para ordenar los números en paralelo
}
```

Ejemplo 1. `Sort.c`. Ordenamiento de números en paralelo.

V ENTREGA

1 Equipos

Esta práctica se hará en equipos (máximo 2 integrantes), es necesario que en la revisión esté el equipo completo ya que el integrante que no se presente no tendrá calificación en la práctica.

2 Entrega

Subir en el apartado correspondiente en Moodle hasta el día Jueves 12 de Marzo a las 11:59 PM un archivo en formato **.zip** que contenga:

1. El archivo **programa fuente** que se pide en la actividad 1
2. El documento en formato .PDF de los resultados obtenidos con Parallel Studio de la actividad 1.1

No incluya líneas de código en sus programas de las cuales desconozca su funcionamiento. El código no conocido será anulado en el funcionamiento de la práctica.

3 Evaluación

Puntualidad en las revisiones	El equipo estuvo completo y puntual en todas las sesiones de revisión.	Si hubo dos o más sesiones con el equipo, el equipo estuvo completo y puntual en casi todas las sesiones de revisión	Si solo hubo una sesión de revisión, el equipo no estuvo completo o no fue puntual. Si fueron dos o más sesiones de revisión, en más de una sesión el equipo no estuvo completo o fue puntual
	+10	+5	0
Funcionamiento	El producto cumple con todas las especificaciones indicadas en el documento y no tiene fallas	El producto muestra una falla no esperada o el producto está casi completo, puede funcionar excepto la parte no completada	El producto muestra más de una falla inesperada o no funciona, esto puede ser debido a que no esté completo.
	+75	+37.5	0
Interfaz con el usuario	El producto funciona y pudo ser utilizado sin necesidad de recibir indicaciones por el desarrollador, tiene instrucciones claras para ser utilizado.	El producto funciona, pero hubo necesidad de recibir alguna indicación para su uso por parte del desarrollador del producto	El producto carece de instrucciones claras para ser utilizado y requiere que alguno de los desarrolladores esté presente para su utilización o no puede utilizarse debido a que no está completo
	+10	+5	0
Claridad en el código	El código es claro, usa nombres de variables adecuadas, está debidamente comentado e indentado. Puede ser entendido por cualquier otra persona que no intervino en su desarrollo.	El código carece de claridad, puede ser entendido por cualquier persona ajena a su desarrollo pero con cierta dificultad.	El código carece de comentarios, está mal indentado, usa nombres de variables no adecuadas.
	+5	+2.5	0
Defensa del producto	Todos los integrantes son capaces de explicar cualquier parte del producto presentado	Alguno de los integrantes muestra dudas sobre alguna parte del desarrollo del producto presentado	Más de un integrante, o si el trabajo fue individual , el desarrollador duda sobre cómo está desarrollado producto.
	x 1(puntos se multiplican por 1)	x 0.5 (puntos se multiplican por 0.5)	x 0 (puntos se multiplican por 0)
Sobresaliente 20 %	Tiene 1 en todos los puntos anteriores. El producto entregado es sobresaliente, muestra tener la calidad para ser expuesto como un producto representativo de la carrera Hay evidencia de que los desarrolladores se documentaron y muestran aprendizajes más allá de lo esperado		No tiene 1 en todos los puntos anteriores, o el producto entregado no es sobresaliente y no muestra tener la calidad para ser expuesto como un producto representativo de la carrera o no hay evidencia de que los desarrolladores se documentaron y muestran aprendizajes más allá de lo esperado
	+20		0