Repaso Paralelismo

1. Ejercicios OpenMP

1.1. Ejercicio 1

Realice un programa en C que cree tres vectores de tamaño n (solicitar al usuario y comprobar que sea positivo). Los dos primeros vectores se tendrán que inicializar con números entre -5 y 5 (ambos incluidos). Después se llamará a una función suma que se encargará de sumar los valores del primer vector con el segundo y guardarlo en el vector resultado. Por último se llamará a una función imprime con cada uno de los vectores para imprimir los tres vectores. Paraleliza los bucles que sean posibles

1.2. Ejercicio 2

Realice un programa en C que lance una región paralela con 4 hilos cada hilo imprimirá su identificador de hilo, el número de hilos de la región paralela y el resultado de mulitplicar su identifcador por el número de hilos de la región paralela.

1.3. Ejercicio 3

Realice un programa en C que cree un vector de tamaño n (solicitar al usuario y comprobar que sea positivo y múltiplo de 4) y se haga la suma de sus elementos midiendo los tiempos que se tarde de las siguientes maneras:

- Paralelizando un bucle for con reduction
- Paralelizando un bucle for con atomic o critical
- Repartiendo el trabajo entre 4 secciones con sumas parciales

1.4. Ejercicio 4

Completa el código para realizar la suma de los vectores a y b de tamaño n y guardarlo en el vector resultado

```
#pragma omp parallel for
for(i=0;i<n;i++){
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
}
```

1.5. Ejercicio 5

Completa el código para realizar la suma los elementos del vector a y guardarlos en la variable total

```
-------------------------------------------------------------------------
#pragma omp parallel for reduction(+:total)
for(i=0;i<n;i++){
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
}
```

1.6. Ejercicio 6

Completa el código para hacerlo en paralelo con OpenMP

```
-------------------------------------------------------------------------
-------------------------------------------------------------------------
for(i=0;i<n;i++){
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  v[i] = a[i] + b[i]
-------------------------------------------------------------------------
-------------------------------------------------------------------------
}
```

1.7. Ejercicio 7

Completa el código para hacerlo en paralelo con OpenMP

```
-------------------------------------------------------------------------
-------------------------------------------------------------------------
for(i=0;i<n;i++){
-------------------------------------------------------------------------
-------------------------------------------------------------------------
  if (maximo < v[i]){
    maximo = v[i];
  }
-------------------------------------------------------------------------
-------------------------------------------------------------------------
}
```

1.8. Ejercicio 8

Indica la directiva correspondiente para paralelizar el siguiente código con OpenMP, indicando para todas las variables si son privadas, compartidas, etc.

```
int modulo = 5;
-------------------------------------------------------------------------
for(i=0;i<n;i++){
  thread_id = omp_get_thread_num();
```

```
    resto = thread_id%modulo;
    printf(\El hilo%d calcula%d", omp_get_thread_num(), resto);
}
```

2. Ejercicios CUDA

2.1. Ejercicio 9

Realice un programa en C que lance un kernel de CUDA con un tamaño de grid de 10 y un tamaño de bloque de 2x3. Cada hilo que ejecute ese kernel imprimirá su identificador de hilo y el identificador de su bloque.

2.2. Ejercicio 10

Escribe un programa en C que calcule la media de los elementos de un vector de tamaño 1024 utilizando un kernel de CUDA. El vector se tiene que inicializar con números aleatorios entre 5 y 15. **Nota**: podéis usar atomicAdd.

2.3. Ejercicio 11

Realice un programa primos.c en lenguaje C que calcule cuántos primos hay en el intervalo 1 a n repartiendo el trabajo entre distintos hilos de GPU utilizando CUDA. **Nota**: podéis usar atomicAdd.

2.4. Ejercicio 12

Realice un programa en lenguaje C que inicializa una matriz 100x100 con valores aleatorios entre 1 y 10 calcule la suma de los elementos de las columnas repartiendo el trabajo entre distintos hilos de GPU utilizando CUDA de forma que cada hilo calcule la suma de una columna.

2.5. Ejercicio 13

Escribe un programa en C que multiplica dos matrices 100x100 utilizando un kernel de CUDA. Las matrices se tienen que inicializar con números aleatorios entre 10 y 20. El resutlado se tiene que guardar en una tercera matriz. Las matrices se tienen que crear como punteros y cada hilo del kernel tiene que calcular un nuevo elemento de la matriz resultante. **Nota**: tenéis que usar dmi3 para definir el tamaño del grid y de los bloques.