High Performance Computing
Departamento de Ingeniería en Informática
PEP1

1. (1.0) Explique cuáles son las ventajas del modelo de concurrencia de uC++, respecto de OpenMP o Pthreads.

2. (2.0) Comente sobre las similitudes y diferencias entre el modelo SIMD y SIMT.

3. (1.5) Escriba un kernel en CUDA que encuentre la suma total de los elementos de un arreglo de enteros de largo N. El algoritmo debe estar basado en los pasos descritos en la Figura 1, usando memoria compartida y memoria global. La Figura 1 ilustra el procedimiento en cada bloque de la grilla. El algoritmo procede iterativamente usando la siguiente lógica:

   (a) Se carga en memoria compartida los elementos del bloque

   (b) Fíjese que no todas las hebras realizan trabajo.

   (c) El número de hebras que ejecutan una suma, se reduce a la mitad en cada iteración

   (d) Las sumas parciales se guardan en el mismo arreglo compartido

   (e) Al final de la iteración, la suma parcial se encuentra en la posición 0 del bloque compartido

   (f) Una hebra de cada bloque suma el valor parcial al valor global (memoria global) usando operación atómica

   Asumiedo que B es el tamaño del bloque y es potencia de 2, y que N es potencia de 2 la invocación del kernel es la siguiente.

```
sumreduction<<<N/B, B>>(A, N, &sum);
```

   La siguiente es la estructura del kernel que usted debe completar.

```
__global__ void sumreduction(int *A, int, int N, int *sum ) {

    // Declare memoria compartida para el bloque

    int globaltid = ...;  // ID global de la hebra

    // Cargar bloque de memoria compartida

    // Sincronizar a que todas hayan terminado

    // Reduccion iterativa dentro del bloque

    // Sincronizar a que todas haya terminado

    // Reduccion total a memoria global sum

}
```

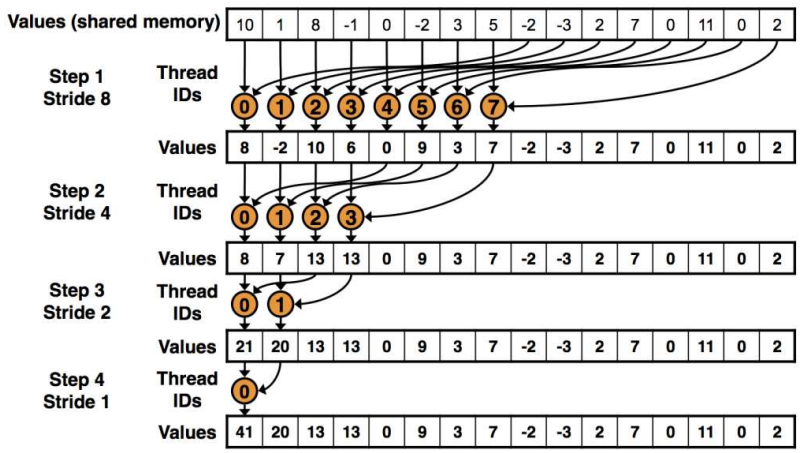

Figure 1: Reducción paralela en CUDA.

4. (1.5) El siguiente código OpenMP (simplificado) intenta que dos hebras compartan variables declaradas como `shared`. Asuma que se crea exactamente dos hebras:

```
main() {
    int yo, vecino, flag[2];
    #pragma omp parallel private(yo, vecino) shared(buffer, flag)
    {
        yo = omp_get_thread_num(); // quién soy yo
        flag[yo] = 0;
        #pragma barrier            // sincronizo
        buffer[yo] = 5*yo;         // produzco un item
        flag[yo] = 1;              // aviso que el dato esta en el buffer

        vecino = (yo == 0 ? 1 : 0);
        while (flag[vecino] == 0);  // espero por el item

        result = buffer[vecino]*buffer[yo]; // computo final
    }
}
```

Explique por qué se produce una condición de carrera. Realice los cambios para eliminar dicha condición.