Práctica evaluable – CURSO 2022/2023

Los siguientes ejercicios, relacionados con la programación en CUDA, se utilizarán para evaluar la parte de prácticas de la asignatura correspondiente al 25% de la nota (2.5 sobre 10). Para ello, a lo largo de los diferentes ejercicios se pondrán en prácticas las cuestiones estudiadas a lo largo de las sesiones 1 a 7.

El resultado evaluable consistirá en un informe donde se recogerán las respuestas a las preguntas planteadas. Algunas de las preguntas implican la generación de archivos de código que habrá que entregar junto con el informe. Es importante que los archivos los entreguéis con los nombres indicados para que así durante su corrección sean más sencillos de localizar. La entrega consistirá en un único archivo comprimido, subido en la tarea generada para tal efecto en campus virtual. El nombre del archivo será el nombre completo del estudiante.

La entrega se realizará el lunes 9 de enero de 2023 (hasta las 23:55 horas). Se sacará un listado con aquellos que tendrán que asistir al examen de prácticas bien por no haber entregado el informe o por no haber superado el mínimo de calidad del mismo. Las notas de prácticas se publicarán con las notas del examen de la convocatoria ordinaria.

1. **[0.05 puntos]** Descarga del espacio virtual de la asignatura el archivo `suma-vectores1.cu` Utilizando el profiler (nvvp) o cualquier otro mecanismo (el API de CUDA) rellena la siguiente tabla:

	Tiempo		
Tiempo total de la sesión			
Kernel			
cudaMalloc DA			
cudaMalloc DB			
cudaMalloc DC			
	Latencia	Tiempo dedicado	Ancho de banda
cudaMemcpy DA ←HA			
cudaMemcpy DB ←HB			
cudaMemcpy HC ←DC			

2. **[0.1 puntos]** Sobre el código del ejercicio anterior, cambiando el número de hebras, N, a 600, realiza las modificaciones que se detalla a continuación y renómbralo como `suma-vectores1b.cu`

    a. Comprobar si se genera error al hacer cada una de las llamadas a cudaMalloc() y a cudaMemcpy().

b. Verificar si la llamada al kernel produce error.

c. Que, si en alguno de los casos anteriores se produce error, el programa use la función cudaGetErrorString() para imprimir el tipo de error producido.

d. ¿Has detectado algún error durante la ejecución?

3. **[0.05 puntos]** Modifica el programa suma-vectores1b.cu para obtener el programa `suma-vectores2.cu` que solo utiliza un bloque de una hebra. Modifica en consonancia el *kernel* para que el programa completo siga sumando los vectores. ¿Funciona correctamente?

4. **[0.05 puntos]** Modifica el programa suma-vectores2.cu para obtener el programa suma-vectores3.cu que utiliza N bloques de una hebra. ¿Funciona correctamente?

5. **[0.15 puntos**] Usando `nvvp`, compara los tiempos empleados por el *kernel* en los programas `suma-vectores1b`, `suma-vectores2` y `suma-vectores3`. Busca una explicación a los tiempos resultantes.

6. **[0.25 puntos]** Supongamos que queremos sumar dos vectores de 100.000 componentes:

   a. ¿Podemos usar el código del programa `suma-vectores3`? Explica el porqué y modifica el programa para obtener el programa `suma-vectores4.cu` que permita la suma de dos vectores de 100.000 componentes cada uno.

7. **[0.25 puntos]** Supongamos que queremos sumar dos vectores con un tamaño N mayor que N mayor que 65535*512 componentes. NOTA: En este caso, podrás hacer que en el *kernel* se procesen más de un elemento utilizando un bucle `for`. El resultado de esta modificación será un programa `suma-vectores5.cu` que permitirá la suma de vectores de cualquier tamaño.

8. **[0.05 puntos]** Modifica el programa `suma-vectores.cu` de modo que se reserve memoria *pinned* para los tres vectores del host. Llama `suma-vect-pinned.cu` al programa resultante. No olvides comprobar si las llamadas a `cudaMallocHost` generan o no error. Después, completa la siguiente tabla:

Tiempos cudaMalloc	DA	DB	DC
`suma-vectores`			
`suma-vect-pinned`			

9. **[0.5 puntos]** El programa `traspuesta1.cu` obtiene la traspuesta de una matriz de `F` filas y `C` columnas usando bloques de `H×K` hebras. El grid también es bidimensional.

    a. Explica si hay coalescencia en las lecturas y en las escrituras en memoria global
    b. Si no fuera así, modifica el código para que se aproveche la coalescencia en memoria global y llama al código resultantes `traspuesta1-coalesc.cu`.
10. **[0.5 puntos]** Consideras necesario incluir alguna directiva para la sincronización de hilos en el bloque. Si fuera así, realiza la correspondiente modificación en un nuevo archivo de nombre `traspuesta2.cu`. No olvides comprobar que funciona correctamente.
11. **[0.5 puntos]** Explica si las lecturas o escrituras del warp generan colisiones en los bandos de memoria. Si fuera así, realiza la modificación necesaria para que esto dejara de ocurrir y genera un archivo de nombre `traspuesta3.cu`.
12. **[0.05 puntos]** Compara los tiempos obtenidos por los programas `traspuesta1.cu`, `traspuesta2.cu` y `traspuesta3.cu`. Explica los resultados obtenidos.