Estos ejercicios constituyen la evaluación de CUDA y su peso en la nota final será de 4 + 0.5 puntos. El código se ha de documentar y hay que explicar el método seguido por los algoritmos implementados en un pequeño informe (ver ejercicio 6).

1. (0.1 puntos) Explica usando la función de CUDA *cudaGetDeviceProperties(...)* las características más relevantes de la o las GPU en las que realizarás los siguientes ejercicios.

2. (0.4 puntos) Desarrolla un programa en C y la correspondiente versión CUDA para realizar la siguiente operación matricial:

   $$D = A \times B + C$$

   donde A, B y C son matrices de tamaño arbitrario que se especifica por parámetros. El programa deberá comprobar que las dimensiones de las matrices son válidas para poder resolver la operación. La generación de matrices se realizará implementando una función auxiliar *gen_matrices(...)* que dados 3 números enteros, genere las matrices A, B y C de manera aleatoria. En los 3 casos, el tipo de datos a utilizar es *float*. El prototipo de la función es el siguiente:

   - void **gen_matrixes**(int a, int b, int c, float *A, float *B, float *C);

   y devuelve 3 matrices de *floats*: A ($a \times b$), B ($b \times c$) y C ($a \times c$).

3. (1 punto) Implementa el programa descrito en el ejercicio anterior utilizando memoria compartida (*__shared__*).

4. (1 punto) Implementa el ejercicio 2 utilizando las funciones expuestas en la API Warp Matrix Multiply Accumulate (WMMA) que utiliza los Tensor cores de la GPU.

5. (1.5 puntos) Ahora, suponemos que A, B y C son matrices gigantes que no entran en la memoria de nuestra GPU. Desarrolla e implementa un algoritmo que supere esta limitación y que nos permita realizar la misma operación que en el ejercicio 2 procesando las matrices en fases. El algoritmo que implementa la multiplicación puede ser tanto el del ejercicio 3 como el del ejercicio 4.

6. **(0.5 puntos) Redacta un informe que *analice* y *compare* el rendimiento de los algoritmos implementados en los ejercicios 2, 3 y 4 utilizando matrices de diferentes tamaños.**

En todos los casos, hay que medir el tiempo necesario para resolver las operaciones usando el algoritmo secuencial del ejercicio 2 (con la función ***clock_gettime***(CLOCK_MONOTONIC,...)) y de manera paralela usando CUDA. Para ello, el código deberá poder compilarse utilizando un *flag* llamado *DEBUG* para realizar la medición o no.

Una vez que se implemente el código CUDA, será necesario medir el rendimiento. Para hacer esto, utilizaremos un servidor que dispone de 2 tarjetas gráficas Nvidia RTX A4000. La dirección de ese servidor es: U107949.ehu.es. Ese servidor usa el gestor de colas SLURM para organizar las ejecuciones de los trabajos de los usuarios. Para usar este software, puede encontrar el script que usaremos en Egela (slurm.sbatch). Lo único que hay que configurar en ese script es, por un lado, la opción #SBATCH –gpus=X, para usar 1 o 2 GPU, y, en la última línea, el comando que se desea ejecutar. Una vez configurado el script, se debe usar el siguiente comando para enviar la ejecución a SLURM: sbatch slurm.sbatch. Una vez enviado, el comando *squeue* mostrará el estado del trabajo enviado y el comando *scancel* jid (job id) lo elimina.