1 Hola Mundo

Considere el código `hello.cu` visto en clase.

(a) En `hello.cu` haga todas las permutaciones posibles del `printf` en el host, la llamada al kernel y de `cudaDeviceSynchronize`. Compila y corra y vea que pasa en cada caso. Resuelva el misterio. Ayuda: `docs.nvidia.com/cuda/cuda-runtime-api/`.

(b) Cada hilo que se ejecuta tiene asignado un identificador único accesible desde el kernel a través de la variable especial `threadIdx.x` (sólo en el caso de ser una grilla de un solo bloque unidimensional). Modifique el kernel de `hello.cu` usando el índice de hilo para que la salida sea:

```
$ ./hello
Hello World from CPU!
Hello World from GPU thread 5!
```

2 Grillas

En el kernel del programa `grillas.cu` cada hilo es instruído para escribir sus variables identificadoras, así como las dimensiones de los bloques y la grilla a la que pertence.

(a) Experimente con grillas de distintas dimensiones, unidimensionales, bidimensionales y tridimensionales, y verifique la salida.

(b) Suponga que que quiera aplicar una operación matemática sobre cada elemento de un vector X de N elementos, usando un kernel. La grilla lanzada, ¿debe contener necesariamente N hilos?

3 Propiedades del Device

Los datos de las tarjetas gráficas nVIDIA instaladas en un dado servidor pueden obtenerse ejecutando el comando `nvidia-smi` en el nodo donde está la GPU. Sin embargo, es muy útil (¿por qué?) poder acceder a esos datos directamente en "runtime", es decir cuando se ejecuta el programa. Para eso, CUDA Runtime API define una estructura `cudaDeviceProp` que contiene esos datos. Consultando el código `deviceQuery.cpp` del cuda-samples y `docs.nvidia.com/cuda/cuda-runtime-api/structcudaDeviceProp.html` complete el programa `queplacasoy.cu` para que imprima lo siguiente:

(a) El máximo número de hilos por bloque en cada dimensión x, y y z, y el máximo número de bloques por grilla en cada dimensión x, y y z. ¿Explica ésto alguna de las preguntas del ejercicio anterior?

(b) Memoria total, la frecuencia de reloj, y el ancho de banda de memoria de cada una de las tarjetas presentes. Comparar estos valores con los de una CPU actual. Discuta sobre la ventaja del uso de GPUs y su potencial.

(c) El número total de hilos y la memoria de la GPU son finitos. Quiero que cada hilo procese un solo elemento de un array de N enteros, lanzando un solo kernel con una grilla dada. ¿Cuál cree que es el factor limitante para N?

4 SAXPY

SAXPY significa "Single-Precision A·X Plus Y". Es una función de la librería standard de subrutinas de Algebra Lineal Básica, BLAS: toma dos vectores de floats de 32-bits X e Y con N elements cada uno, y un valor escalar A. Luego multiplica cada elemento $X[i]$ por A y suma el

resultado a $Y[i]$. Considere para este ejercicio la opción in-place, donde el resultado se guarda en Y, es decir $Y \leftarrow A * X + Y$. Una matriz de $N \times M$ se guarda usualmente en un vector de NM elementos ordenados por fila o columna. Considere que los vectores X e Y representan a las matrices M_X y M_Y de $N \times N$ elementos ordenados por fila. Adaptando el código de suma de vectores dado en clase, haga lo siguiente:

(a) Con un primer kernel rellene los vectores X e Y en paralelo tal que $M_X(i,j) = 1/(j+i)$ y $M_Y(i,j) = 1/(j-i)$.

(b) Para el punto anterior experimente con (i) una grilla unidimensional de bloques unidimensionales, (ii) una grilla bidimensional de bloques bidimensionales, y (iii) otras grillas o indexados que se le ocurra. Elija luego el indexado más conveniente para continuar.

(c) Con otro kernel, lanzado a continuación, realice la operación SAXPY "in-place".

(d) Traiga los vectores desde device al host y verifique, en el host, la correctitud de la operación realizada en el device.

(e) Agregue, al inicio de su programa paralelo, la impresión de las propiedades de la GPU en la que está corriendo. Consulte el código `queplacasoy.cu` visto en clase.

(f) Escriba un programa serial en CPU que realice la mismas operaciones de llenado de las matrices, SAXPY y verificación de resultado (en cualquier lenguaje).

(g) Analice la aceleración lograda con la placa gráfica en función de N, con respecto a la versión serial en CPU. Analice ambos, el tiempo de la operación SAXPY y el tiempo de las copias de host a device, y de device a host involucradas. Experimente con `nvprof` y los timers vistos en clase. Repita el análisis anterior para vectores de tipo double de 64-bits en GPU (la operación SAXPY se convierte en DAXPY).

5 Transformaciones

Escriba un programa que aloque memoria en device para un array a de N floats y que luego ejecute dos kernels en secuencia.

(a) El primer kernel que llene el vector con una función arbitraria de su índice, por ejemplo $a[i] = \tanh(\cos(\exp(-i * 0{,}01) + 0{,}02))$.

(b) El segundo que invierta el array, generando el array de device $b = (a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$.

(c) Analice cuáles son las dificultades para que el segundo kernel invierta el array a *in-place*, es decir, que devuelva el array a invertido.

(d) Escriba un kernel que fusione en uno solo las operaciones de los dos kernels anteriores.

6 Multiplicación de Matrices

En el template `multmat.cu` encontrará una implementación sencilla de un kernel que multiplica dos matrices A y B y copia el resultado en C. Las matrices no son necesariamente cuadradas, pero la multiplicación $C_{ij} = \sum_k A_{ik} B_{kj}$ tiene que estar bien definida.

(a) Complete el código, siguiendo los comentarios que empiezan con "TODO". Debe compilar y correr correctamente para continuar.

(b) Usando los timers vistos en clase, cronometre y compare los tiempos de la multiplicación en host y device, para distintos tamaños de matriz, en simple y doble precisión. ¿Como "escalea" el tiempo de la multiplicación con el tamaño de las matrices? (por simplicidad, particularice a matrices cuadradas para este item).