Práctica 4.- Iniciación a CUDA

Arquitectura y Computación de Altas Prestaciones. Universidad de Granada. Curso 2023/24

Objetivos: En esta práctica se pretende que los estudiantes se familiaricen con los conceptos principales de la programación en CUDA y la medición de rendimiento.

Bases: Esta práctica se realizará individualmente y los programas desarrollados estarán pensados para funcionar en un entorno Linux con tarjeta gráfica compatible con CUDA. Habrá que entregar los archivos de código fuente desarrollados y una memoria (en formato PDF) que explique el trabajo realizado. La entrega se realizará mediante la actividad que se creará en SWAD para tal efecto. Las entregas tardías supondrán una penalización en la nota (0.5 puntos por día hasta alcanzar el límite de 5).

Entorno: La mayor parte del trabajo se puede realizar en equipos personales. Sin embargo, según la actividad, habrá que acceder al cluster GenMagic. Cada estudiante tendrá un usuario y contraseña en dicho cluster. Serán asignados por el profesor de forma individual e intransferible. Tampoco se podrán cambiar las claves recibidas. El acceso al cluster puede requerir estar conectado por VPN a la red de la universidad.

Ejercicio 1 (4 puntos)

Lee atentamente el tutorial que se encuentra públicamente disponible en https://developer.nvidia.com/blog/how-implement-performance-metrics-cuda-cc/. Realiza una traducción reemplazando sus resultados de medición por los que tú obtengas, ya sea en tu propia máquina o en GenMagic. Lógicamente, en el proceso de traducción y replicación ten en cuenta la tarjeta que estés usando (por ejemplo, respecto a valores pico).

Ejercicio 2 (6 puntos)

Trabajaremos en el cálculo del histograma de una imagen. Éste representa la frecuencia relativa de sus niveles de gris. Esta información tiene aplicaciones reales para el tratamiento de imágenes, como el aumento del contraste de imágenes con histogramas muy concentrados. Puedes encontrar más información en https://es.wikipedia.org/wiki/Histograma.

A nivel de cálculo podemos resumir así el procedimiento:

```
for(int i = 0; i < BIN_COUNT; i++)
    result[i] = 0;

for(int i = 0; i < dataN; i++)
    result[data[i]]++;
```

Podríamos plantearnos una primera aproximación CUDA definiendo un buffer con los datos y un solo histograma para el dispositivo. Todos los hilos actualizarían el conteo sobre la misma estructura con *atomicAdd*. Con un solo bloque y 1024 hilos tendríamos algo así:

```
HistoKernel_v1 <<<1 , 1024>>> (buffer ,  histo);
```

(...)

```
__global__ void
HistoKernel_v1 (const char *const buffer , int
*histo)
{
      int i = threadIdx.x ;
      atomicAdd (&histo[buffer[i]] , 1 );

}
```

Claro, que los datos no podrían ser más de 1024… En este mismo contexto podríamos plantearnos usar bloques de hilos mapeando el trabajo entre todos:

```
HistKernel_v2<<<N/1024 , 1024>>>(buffer ,  histo);

HistKernel_v2b<<<(N+1023)/1024 , 1024>>>(buffer ,  histo);
```

(...)

```
__global__ void
HistoKernel_v2(const char *const buffer , int *histo){
    int id = blockidx.x * blockDim.x + threadIdx.x ;
    atomicAdd (&histo[buffer[id]] , 1 );
}

__global__ void
HistoKernel_v2b(const char *const buffer , int *histo){
    int id = blockidx.x * blockDim.x + threadIdx.x ;
    if( id < lenBuffer)
        atomicAdd (&histo[buffer[id]] , 1 );
}
```

Repartimos en bloques de 1024 hilos, pero podemos vernos desbordados fácilmente (alcanzar el límite dimensional) con muchos datos. Podríamos pues trabajar en dos dimensiones:

```
int mitadN = N/2 ;
dim3 blocks (16, 16);
dim3 grid(mitadN / 16 , mitadN/16) ;
HistoKernel_v3 <<<grid , blocks>>> (buffer , histo, mitadN ) ;
```

```
__global__ void
HistoKernel_v3(const char *const buffer , int *histo, int mitadN)
{
      int id_x = blockidx.x * blockDim.x + threadIdx.x ;
      int id_y = blockidx.y * blockDim.y + threadIdx.y ;
      int absoulteId = id_y*mitadN + id_x ;

      atomicAdd (&histo[buffer[absoulteId]] , 1 );

}
```

No obstante, también habría una situación limitante potencial.

Bien, se proporciona una implementación sencilla del cálculo del histograma en el archivo “pr4.cpp” (tiene esa extensión en lugar de “cu” para poder subirse sin problemas a SWAD, pero corrígela una vez descargado). Dicha implementación, que trabaja sobre una imagen sintética, intenta tener en cuenta los problemas previos e incluye además una medición del tiempo empleado. Empieza por estudiar con detenimiento el código dado y entender su funcionamiento.

Seguidamente, ten en cuenta que el código dado arrastra algunos problemas: El número de hilos por bloque no se tiene en cuenta correctamente, pues asume un número adecuado de tamaño de bloque. Empieza por **generalizarlo**. (3 puntos)

Posteriormente, realiza un estudio sobre cómo afecta la variación del tamaño de bloque considerando al menos 4 valores distintos. (1 punto)

A continuación, haz el mismo tipo de estudio sobre el impacto del tamaño grid (número de bloques) tras haber fijado el tamaño de bloque al mejor valor encontrado anteriormente. (1 punto)

Finalmente, sobre la versión general y mejor configuración, calcula la aceleración sobre el cálculo en CPU. Ésto hazlo de dos formas, primero limitándote a comparar sólo tiempos de cálculo, y en última instancia teniendo en cuenta las transferencias a (y desde) GPU en los registros de tiempo. (1 punto)