ESCUELA DE INGENIERÍA Y CIENCIAS
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

GPGPU - Laboratorio 1 - Primeros pasos con CUDA

Maxime MARIA

Laboratorio XLIM, UMR CNRS 7252 - Universidad de Limoges, Francia

Marzo 2024

El objetivo de esta asignatura es introducirte en la programación paralela en la GPU utilizando CUDA. CUDA (Compute Unified Device Architecture) es una tecnología patentada por NVidia que aprovecha la enorme capacidad de cálculo paralelo inherente a las GPU.

Deberá consultar la documentación oficial para resolver los distintos ejercicios (`https://docs.nvidia.com/cuda/index.html`).

Para compilar un programa CUDA, puede utilizar el siguiente comando :

```
$ nvcc main.cu [other_files.cu other_files.cpp] -o output_name
```

Los nombres de los archivos .cpp o .cu que contienen las dependencias deben añadirse, separados por un espacio, después del nombre del archivo que contiene la función `main`.

1 Device properties

1. Descargue la carpeta `Lab1-Ex1-Device_properties`.
2. Compile y ejecute el programa dado para mostrar las distintas propiedades de la tarjeta gráfica utilizada. Debería ver información sobre la(s) GPU(s) de su ordenador.

2 Suma de vectores : $\vec{a} + \vec{b} = \vec{c}$

En este ejercicio, va a sumar dos vectores de números enteros en paralelo.

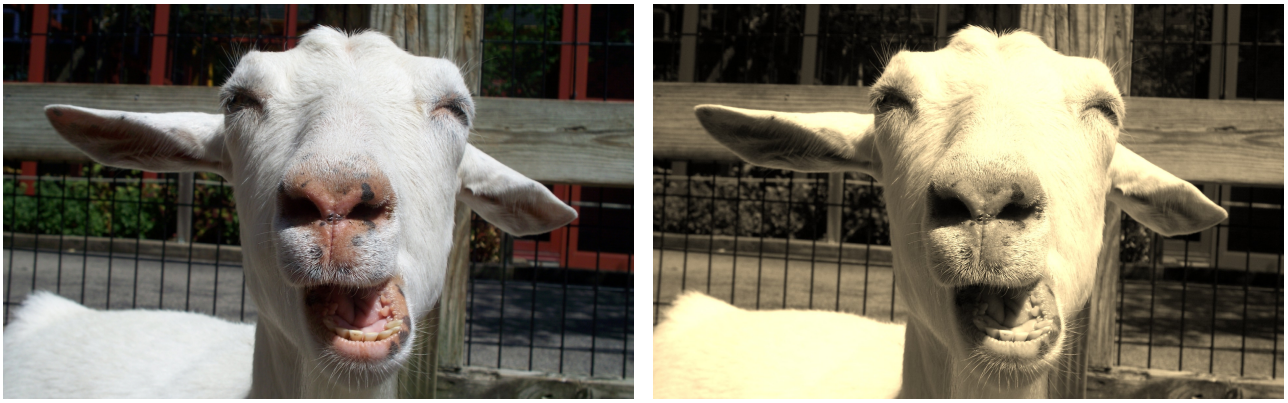
1. Descargue la carpeta `Lab2-Ex2-Vectors_sum`. En esta carpeta, sólo tendrá que modificar los archivos *student.hpp* y *student.cu*. Los archivos *reference.hpp* y *reference.cpp* contienen el cálculo de la suma secuencial de la CPU. El archivo *main.cu* contiene el programa principal que llama a las funciones de cálculo, mide los tiempos de ejecución y compara los resultados de CPU/GPU.
2. Complete el archivo *student.cu* para :
   - asignar memoria en la GPU : `allocateArraysCUDA` ;
   - transferir los datos de la CPU a la GPU : `copyFromHostToDevice` ;
   - configurar la distribución de los *threads* y ejecutar el *kernel* : `launchKernel` ;
   - recuperar los datos de la GPU : `copyFromDeviceToHost` ;
   - liberar la memoria en la GPU : `freeArraysCUDA`.

   Para cada llamada a una función CUDA, utilice la macro `HANDLE_ERROR` disponible en el archivo `commonCUDA.hpp`. Esto le permite recuperar errores CUDA durante la ejecución de sus programas.

3. Implemente el *kernel* `sumArraysCUDA` que calcula la suma.
4. Su *kernel* debería funcionar sea cual sea el número de *blocks* y *threads* elegido. Si este no es el caso, modifique su *kernel* en consecuencia.
5. Compare los tiempos de ejecución entre diferentes versiones (CPU/GPU) variando :
   - el tamaño de los vectores : indique el tamaño del programa como argumento utilizando la opción `-n` (¡tenga cuidado de no superar la capacidad de memoria de su GPU !) ;
   - el número de *blocks* y *threads*.

3 Filtro de imagen sepia

En este ejercicio debe implementar un algoritmo que, a partir de una imagen en formato `.ppm`, aplique un filtro para producir una nueva con aspecto sepia (*cf.* Figura 1).



(a) (b)

FIGURE 1 – Filtro de imagen sepia : (a) Imagen de origen ; (b) Imagen sepia.

La imagen de origen se pasa como argumento al programa utilizando la opción `-f image_path.png`. El algoritmo del filtro es sencillo y consiste en modificar el color de cada píxel de forma que :

```
outRed = min(255, (inRed * .393f + inGreen * .769f + inBlue * .189f));
outGreen = min(255, (inRed * .349f + inGreen * .686f + inBlue * .168f));
outBlue = min(255, (inRed * .272f + inGreen * .534f + inBlue * .131f));
```

1. Descargue la carpeta `Lab1-Ex3-Sepia_filter`. Una vez más, sólo tendrá que modificar los archivos *student.hpp* y *student.cu*.
2. Complete la función `sepiaGPU` (asigne/transfiera/libere la memoria, *etc.*).
   Esta vez **hay que** configurar la distribución de los *threads* en una grilla 2D (podríamos haberlo hecho en 1D, ¡pero no !).
3. Implemente el *kernel* `kernelComputeSepia` y compruebe que funciona correctamente.
4. Su *kernel* debería funcionar sea cual sea el número de *blocks* y *threads* elegido. Si este no es el caso, modifique su *kernel* en consecuencia.
5. Compara y analiza los tiempos de ejecución medidos para imágenes de distintos tamaños y con diferentes números de bloques e hilos. Encontrará algunas imágenes de prueba en la carpeta `img`. Sin embargo, no dude en probar sus algoritmos con otras imágenes (en formato `.png` o `.jpg`). Tenga en cuenta que la imagen `galaxy_16k.png` es muy grande.