

INF-384: Taller de Modelos y Métodos Cuantitativos

Taller 1

Promediando Imágenes

Prof. Álvaro Salinas

15 de Octubre de 2020

1. Descripción y Marco Teórico

En el siguiente taller, usted realizará un análisis del paralelismo ofrecido por CUDA al resolver un determinado problema. Para lograrlo, deberá implementar CUDA kernels que permitan ejecutar un procesamiento paralelo sobre un conjunto de datos, para posteriormente comparar su desempeño con una versión secuencial que se ejecute en la CPU. Los objetivos principales de esta experiencia consisten en evaluar sus conocimientos sobre los fundamentos básicos de la programación en CUDA, tales como la elaboración de kernels, manejos de memoria, distribución de trabajo en CUDA threads y mediciones de tiempo.

Promedio de Imágenes

En el presente taller, vamos a promediar varias imágenes en una única resultante. Para lograr esto, tendremos que promediar cada canal (R, G y B) del mismo píxel de las distintas imágenes, siendo el resultado de esta operación el respectivo canal de dicho píxel en la imagen resultante.

Consideremos el siguiente ejemplo:

$$\begin{array}{|c|c|} \hline 0 & 1 \\ \hline 0,5 & 0,25 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 1 & 0,5 \\ \hline 0,5 & 0,1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 0,5 & 0,75 \\ \hline 0,5 & 0,175 \\ \hline \end{array}$$

Imagen 1 (R) Imagen 2 (R) Imagen Resultante (R)

donde $*$ representa nuestra operación promedio. En el ejemplo, cada matriz representa el mismo canal, en este caso R, de cada imagen y es posible apreciar como el primer píxel de la Imagen 1 se promedió con el primer píxel de la Imagen 2, dando como resultado el primer píxel de la Imagen Resultante ($\frac{0+1}{2} = 0,5$). Lo mismo ocurre para cada uno de los píxeles restantes. Realizando esta operación completa para cada canal, obtenemos los tres canales de la Imagen Resultante y podemos recrearla. Un ejemplo del resultado de esta operación en imágenes reales es:



Imagen 1

$*$

Imagen 2

=

Imagen Resultante

2. Archivos de Entrada

Los archivos que su programa recibirá tienen el siguiente formato:

- La primera línea contiene tres enteros L , M y N , en dicho orden. L corresponde al número de imágenes que se promediará. M y N son las dimensiones de cada imagen (todas las imágenes utilizadas tienen la misma resolución). En este caso, M corresponde al número de filas de píxeles en la imagen y N el número de columnas, es decir, el largo de cada una de las M filas.
- Luego de la primera línea, existen $L \times 3$ líneas. Cada una de ellas contiene $M \times N$ flotantes de 0 a 1, correspondientes a la concatenación de las filas de cada canal de cada imagen. En otras palabras:
 - La segunda línea del archivo contiene los $M \times N$ píxeles del canal R de la primera imagen.
 - La tercera línea del archivo contiene los $M \times N$ píxeles del canal G de la primera imagen.
 - La cuarta línea del archivo contiene los $M \times N$ píxeles del canal B de la primera imagen.
 - La quinta, sexta y séptima línea del archivo contienen los canales R, G y B respectivamente de la segunda imagen.
 - Y así sucesivamente en grupos de tres líneas para las L imágenes.

Por ejemplo, considerando el primer ejemplo mostrado en la sección anterior y asumiendo que los canales no mostrados (G y B) son nulos, el archivo de entrada tendría la siguiente información:

2 2 2	2 imágenes de 2×2 píxeles
0.000 1.000 0.500 0.250	Filas concatenadas del canal R de la Imagen 1
0.000 0.000 0.000 0.000	Filas concatenadas del canal G de la Imagen 1
0.000 0.000 0.000 0.000	Filas concatenadas del canal B de la Imagen 1
1.000 0.500 0.500 0.100	Filas concatenadas del canal R de la Imagen 2
0.000 0.000 0.000 0.000	Filas concatenadas del canal G de la Imagen 2
0.000 0.000 0.000 0.000	Filas concatenadas del canal B de la Imagen 2

Junto a este enunciado, encontrará los archivos `imagesN.txt`, con N del 1 al 6 que siguen el formato explicado y deberá usar en el desarrollo de este taller. Cada uno de estos archivos tienen 2764800 píxeles repartidos en imágenes de distinto tamaño desde las más pequeñas hasta las más grandes:

- `images1.txt` posee 691200 imágenes de 2×2 .
- `images2.txt` posee 172800 imágenes de 4×4 .
- `images3.txt` posee 10800 imágenes de 16×16 .
- `images4.txt` posee 300 imágenes de 72×128 .
- `images5.txt` posee 75 imágenes de 144×256 .
- `images6.txt` posee 3 imágenes de 720×1280 .

Archivos de Salida

Se le recomienda utilizar el formato visto en la clase de ejemplos. De esta forma, no debe realizar un nuevo código para la escritura del archivo ni tiene que crear una nueva herramienta para visualizar el resultado, pues puede utilizar la función `TXTtoRGB` presente en el script que se publicó.

3. Desarrollo

1. Como primeros pasos, cree una función de C/C++ que permita leer los archivos de entrada. También, en el main, genere los arreglos necesarios, reserve la memoria y copie la información necesaria a GPU.
2. Implemente una función secuencial de C/C++ que genere la imagen resultante a partir de las imágenes de entrada.
3. Implemente un CUDA kernel que realice el mismo procedimiento que la función creada en el punto anterior. Preocúpese de seguir buenas prácticas y tener una correcta indexación. Enfoque el paralelismo de su implementación de forma adecuada, recordando que si varias hebras intentan escribir valores en la misma dirección de memoria se pueden producir condiciones de carrera.

[Pregunta] Antes de ejecutar su código, ¿cuál de las dos implementaciones cree usted que presentará un mejor desempeño? ¿Depende de algo? Argumente su respuesta.

4. Ejecute su solución para cada archivo de entrada y mida los tiempos de cada caso (recuerde medir el tiempo correctamente).

[Informe] Presente en su informe las 6 imágenes resultantes obtenidas (usted debe asegurarse de que ambas implementaciones entreguen el mismo resultado, pero no necesita mostrar estos duplicados en el informe).

[Informe] Realice un gráfico o tabla en donde se muestren los 12 tiempos obtenidos (cada una de las 2 implementaciones para cada uno de los 6 archivos de entrada). Si opta por realizar un gráfico, obtendrá 2 curvas distintas (una por cada implementación), conteniendo el eje *x* los valores de $M \times N$, mientras que el eje *y* corresponderá a los tiempos medidos. *Hint: considere utilizar una escala logarítmica si las curvas no se aprecian correctamente.*

[Pregunta] Analice los tiempos obtenidos. ¿Qué puede concluir? Comente sobre el comportamiento observado y relaciónelo al enfoque de paralelismo (trabajo de cada hebra) que utilizó.

4. Reglas y Consideraciones

Entrega

- Se debe realizar una entrega semanal del código al finalizar cada clase. Junto al código, se debe adjuntar un documento en donde se comente brevemente lo que se realizó en la respectiva sesión.
- Al finalizar el taller, se debe realizar una entrega final. Esta corresponde al código terminado más un informe en formato pdf. Se le ruega entregar un código ordenado.
- El informe debe contener:
 - Título y número del taller.
 - Nombre y rol de todos los integrantes del grupo.
 - Modelo y compute capability de la tarjeta gráfica que fue utilizada para ejecutar el código. Si se probó con más de una tarjeta gráfica, incluya los datos de todas y especifique qué tarjeta se utilizó en cada pregunta y resultado reportado.
 - Desarrollo. Preocúpese de incluir cada ítem señalado con los tag [Pregunta] e [Informe] en el enunciado.
 - Conclusiones. Incluya aprendizajes, desafíos, comentarios, observaciones o supuestos que surgieron durante el desarrollo del taller.
- Cada entrega debe realizarse en un archivo de nombre Taller1-X.tar.gz (formatos rar y zip también son aceptados), donde X debe ser reemplazado por el número de su grupo. Diríjase al documento Grupos publicado en el aula para consultar su número.
- En caso de copia, los grupos involucrados serán evaluados con nota 0. No hay problema en generar discusión y compartir ideas de implementación con sus compañeros, pero códigos copiados y pegados no serán aceptados.
- El no cumplimiento de estas reglas implica descuentos en su evaluación.

Consideraciones

- Trabaje con flotantes de precisión simple (`float`).
- Para mediciones de tiempo, `clock()` de la librería `time.h` en caso de mediciones en CPU y `cudaEvent()` o Nvidia Visual Profiler para códigos de GPU. Trabaje con milisegundos [`ms`].
- En caso de optar por la realización de gráficos, puede optar por la herramienta que más le acomode. La librería `matplotlib` de Python siempre es una buena opción.
- Utilice 256 hebras por bloque en su implementacion.