Computación de propósito general de unidades de procesamiento gráfico

Práctico 3

Procesamiento de imágenes en GPU

Integrantes

Nombre	CI	Correo
Renzo Gambone	5.155.486-4	renzo.gambone@fing.edu.uy
Germán Ouviña	4.823.566-1	german.ouvina@fing.edu.uy

Índice

1. Introducción

En el presente informe se detallan los resultados obtenidos durante la evaluación experimental de los ejercicios correspondientes al **Práctico 3**, así como su pertinente análisis. A continuación, se destacan ciertos aspectos a tener en cuenta del análisis y de los archivos entregables:

1. La implementación de los algoritmos se realizó empleando el lenguaje **C** y la API de **CUDA**.
2. La evaluación experimental se realizó por medio de la plataforma ClusterUY. Durante la evaluación experimental los recursos que nos fueron asignados y con los que ejecutamos el algoritmo fueron:
   - **GPU:** Tesla P100-PCIE-12GB
   - **CPU:** Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz
3. Los datos manipulados fueron imágenes representadas por vectores de números decimales en punto flotante (*float* de **C**).
4. Los tiempos registrados al realizar distintas ejecuciones de un mismo script no fueron siempre constantes, sin embargo su variación osciló con un radio lo suficientemente bajo como para que las observaciones y conclusiones presentadas se preserven.

2. Desarrollo

2.1. Ejercicio 1 - Ajuste de brillo

En esta sección se detallan los resultados obtenidos de la experimentación realizada con los algoritmos de ajuste de brillo. Dicha experimentación consistió en registrar los tiempos de cada algoritmo en distintas etapas, así como ejecutar una herramienta de profiling para analizar la performance.

Para ello se trabajó con tres implementaciones de `ajustar_brillo`, una función que altera el valor de cada píxel de una imagen en escala de grises, sumándole un coeficiente $coef \in [-255, 255]$ al valor de cada uno. Las implementaciones fueron las siguientes:

- `ajustar_brillo_cpu`, una versión secuencial del algoritmo pensada para ejecutarse en un núcleo de CPU.
- `ajustar_brillo_coalesced_kernel`, una versión del algoritmo pensada para ejecutarse en GPU, en donde threads con índice consecutivo en la dirección x acceden a píxeles de una misma fila de la imagen
- `ajustar_brillo_no_coalesced_kernel`, una versión del algoritmo pensada para ejecutarse en GPU, en donde threads con índice consecutivo en la dirección x acceden a píxeles de una misma columna de la imagen

2.1.1. Tiempos registrados

En esta sección se adjuntan los tiempos registrados para cada implementación del algoritmo, teniendo en cuenta sus 3 versiones.

Se destaca que la ejecución de un algoritmo en GPU presenta un overhead respecto a la ejecución de un algoritmo equivalente en CPU. Esto se debe a que antes de invocar el kernel, se debe reservar memoria en la GPU y transferir los datos de la memoria del CPU a la memoria global de la GPU. A su vez, después de que el kernel haya finalizado su ejecución, es necesario transferir la salida de la memoria de la GPU para poder trabajar con la misma en CPU. Si bien es importante comparar los tiempos totales entre ambos algoritmos, los overheads anteriormente mencionados suelen representar ruido a la hora de enfocar la comparación en lo que concierne al tiempo del algoritmo matemático (aplicar la función a la estructura dada). Es por esto que se registra por separado los tiempos en cada una de las siguientes etapas:

1. Reserva de memoria
2. Transferencia de datos de entrada
3. Ejecución del kernel
4. Transferencia de datos de salida

A continuación, se adjuntan los tiempos tomados:

Etapas	**CPU**	**GPU (coalesced)**	**GPU (no coalesced)**
Reserva de Memoria	-	0.4049 ms	0.3977 ms
Transferencia de Datos (a Device)	-	1.0556 ms	1.0413 ms
Ejecución del Kernel	-	0.0452 ms	0.1258 ms
Transferencia de Datos (a Host)	-	0.8008 ms	1.1764 ms
Total	**16.7522 ms**	**2.3065 ms**	**2.7411 ms**

2.1.2. Profiling con *nvprof*

Es importante destacar que los resultados obtenidos del profiling en el ClusterUY no fueron los esperados. Tras haber sido este suceso investigado por el cuerpo docente, se determinó que este fenómeno se da debido a particularidades de la arquitectura de la tarjeta de video proporcionada Tesla P100-PCIE-12GB.

En particular, dichas particularidades son en referencia a cómo la tarjeta gráfica trabaja y cachea números en punto flotante. Para obtener los resultados que uno obtendría en una GPU normal (y esperados por el ejercicio) se compiló una versión del programa con la flag `nvcc -Xptxas -dlcm=cg ...`, la cual permite que el programa resultante ejecute con la caché L1 de la GPU deshabilitada.

Es así que los resultados obtenidos fueron:

```
==30408== NVPROF is profiling process 30408, command: ./blur img/fing1_ruido.pgm
==30408== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==30408== Profiling application: ./blur img/fing1_ruido.pgm
==30408== Profiling result:
==30408== Metric result:

Device "Tesla P100-PCIE-12GB (0)"
Invocations     Metric Description                Min     Max     Avg

Kernel: ajustar_brillo_coalesced_kernel(float*, float*, int, int, float)
    2           Global Memory Load Efficiency     100.00% 100.00% 100.00%

Kernel: blur_kernel(float*, int, int, float*, float const *, int)
    1           Global Memory Load Efficiency     71.30%  71.30%  71.30%

Kernel: ajustar_brillo_no_coalesced_kernel(float*, float*, int, int, float)
    1           Global Memory Load Efficiency     12.50%  12.50%  12.50%
```

La métrica extraída del análisis (Global Memory Load Efficiency) representa la relación entre el total de memoria global pedida y el total de memoria global requerida por el programa.

2.1.3. Observaciones

A partir de los resultados obtenidos se destacan las siguientes observaciones:

- El tiempo total del algoritmo en CPU fue ampliamente mayor a los algoritmos de GPU. Si se comparan los tiempos de los kernels solamente, esta diferencia es aún mayor.
- El tiempo total del algoritmo coalesced en GPU fue mayor al del algoritmo no coalesced. Particularmente, la mayor diferencia entre los tiempos de ambas etapas puede verse en el tiempo del kernel, el cual para el algoritmo no coalesced fue 3 veces mayor que para el coalesced.
- La eficiencia de carga de memoria global es del 100 % para el algoritmo coalesced, mientras que resulta del 12.50 % para el algoritmo no coalesced.
- La mayoría del tiempo utilizado en los algoritmos de GPU proviene de las etapas de transferencia de datos entre host y device.

2.2. Ejercicio 2 - Filtro Gaussiano

En esta sección se detallan los resultados obtenidos de la experimentación realizada con los algoritmos de filtro gaussiano. Dicha experimentación consistió en registrar los tiempos de cada algoritmo en distintas etapas y realizar observaciones sobre las diferencias apreciadas.

Para ello, se trabajó con dos implementaciones de `blur`, una función que aplica un filtro gaussiano a cada píxel de una imagen, sustituyendo el valor del mismo por un promedio ponderado de los píxeles más cercanos al mismo. Las implementaciones fueron las siguientes:

- `blur_cpu`, una versión secuencial del algoritmo pensada para ejecutarse en un núcleo de CPU.
- `blur_gpu`, una versión paralela pensada para ejecutarse en GPU, la misma divide la imagen en bloques de 32×32 píxeles. Cada bloque es compuesto por 1024 hilos (32×32) y empleando paralelización cada hilo computa el nuevo valor del píxel que le corresponde.

2.2.1. Tiempos registrados

En esta sección se adjuntan los tiempos registrados para cada implementación del algoritmo, teniendo en cuenta sus 2 versiones.

Es importante destacar que, respecto al algoritmo en GPU, se tuvieron las mismas consideraciones que en la etapa anterior (particularmente en relación a las distintas etapas de la función). También se destaca que el kernel de `blur_gpu` fue optimizado cacheando la máscara a aplicar en el promedio ponderado de los píxeles vecinos en memoria caché, haciendo uso de la directiva `const __restrict__`.

A continuación, se adjuntan los tiempos tomados:

Etapas	**CPU**	**GPU**
Reserva de Memoria	-	0.5408 ms
Transferencia de Datos (a Device)	-	1.0503 ms
Ejecución del Kernel	-	0.2271 ms
Transferencia de Datos (a Host)	-	0.8074 ms
Total	**44.9551 ms**	**2.6257 ms**

2.2.2. Observaciones

Se realizaron las siguientes observaciones:

- El tiempo total del algoritmo en CPU fue ampliamente mayor al del algoritmo de GPU. Si se comparan los tiempos del kernel solamente, esta diferencia es aún mayor.
- La mayoría del tiempo utilizado en el algoritmo de GPU proviene de las etapas de transferencia de datos entre host y device.

3. Conclusiones

Como conclusión principal, se pudo observar que los algoritmos corridos en GPU tienen un mejor tiempo de ejecución, independientemente al hecho de que agreguen etapas como la reserva de memoria extra en el device o la transferencia de datos de host a device.

A su vez, se notó que los tiempos de transferencia representaron la mayor parte del tiempo total de los algoritmos en GPU. Esto permite concluir que, al escalar el tamaño de los datos de entrada, el tiempo en CPU escalará más rápidamente que el tiempo en GPU, ya que el primer algoritmo es secuencial y el segundo paralelo.

Finalmente, se observó la importancia de realizar un acceso eficiente a memoria al utilizar algoritmos en GPUs. Si bien las diferencias de tiempo entre el ajuste de brillo coalesced y no coalesced no hizo un gran impacto en el tiempo total del algoritmo, se preveé que al escalar el tamaño de los datos de entrada, dicha diferencia también lo haría.