

Práctica 2

Mapeo Tonal

Procesadores Gráficos y Aplicaciones en Tiempo Real

Alejandro López Vizuite
Exp - 410

Título	Página
1. Descripción del código	2
1. <i>__global__ void scan</i>	2
2. <i>__global__ void histograma</i>	2
3. <i>__global__ void calcularMinimo</i>	2
4. <i>__global__ void calcularMaximo</i>	2
5. <i>void calculate_cdf</i>	2
2. Análisis de mejoras introducidas	3
1. <i>Mejoras de rendimiento</i>	3
3. Comentarios personales	5
1. <i>Problemas encontrados</i>	5

1. Descripción de código

__global__ void scan

Kernel realizado con la ayuda de las diapositivas del tema 2.7 del temario de PGATR.

Realiza la función exclusive Scan para el histograma pasado como parámetro. Se compone de dos partes diferenciadas, llamadas Reduction y Reverse. Calcula los valores definitivos para el cdf.

__global__ void histograma

Kernel realizado con la ayuda de las diapositivas del tema 2.7 del temario de PGATR y el propio documento explicativo de la práctica.

Permite extraer características de los datos, en este caso la imagen. Contando con los datos sacados de la imagen, podemos aumentar de uno en uno mediante atomicAdd cuando queremos destacar una característica. En este caso, según la función dada en el documento explicativo: $\text{bin} = (\text{Lum}[i] - \text{lumMin}) / \text{lumRange} * \text{numBins}$.

__global__ void calcularMaximo

Kernel realizado con la ayuda de las diapositivas del tema 2.7 del temario de PGATR y videos explicativos del curso Introduction to Parallel Programming de Udacity.

Se realiza la función de reducción, consistente en resumir un conjunto de valores en un único valor, para sacar el valor máximo de luminancia de la fotografía dada.

__global__ void calcularMinimo

Misma función que calcularMaximo, pero en este caso obtenemos el valor mínimo de luminancia.

void calculate_cdf

Función principal a la que se llama desde el “main” del programa. Su función es ir llamando a los kernel explicados anteriormente para devolver la salida del exclusive scan al d_cdf pasado como parámetro.

Debemos crear variables, reservando y liberando memoria correspondiente para cada una, además de manejar cada blockSize y gridSize de cada llamada a funciones kernel.

El orden de llamada viene dado en el propio código, y es necesario realizarlo en ese estricto orden debido a que la salida de cada kernel se utiliza en kernel posterior.

2. Análisis de mejoras introducidas

Mejoras de Rendimiento

Al contrario que la práctica anterior, en este caso al ejecutar el programa de Nvidia, Visual Profiler, he considerado oportuno realizar las comparaciones mediante el tiempo de ejecución resultante en la línea de comandos.

He realizado cuatro configuraciones distintas, para ver cual de ellas tarda menos.

- Sin memoria compartida y tamaño de bloque 32

```
C:\WINDOWS\system32\cmd.exe
El minimo es: -3.122315
El maximo es: 2.350199
El rango es: 5.472514
El numero de bins es: 1024
Your code ran in: 14.331328 msecs.
Presione una tecla para continuar . . .
```

- Sin memoria compartida y tamaño de bloque 16

```
C:\WINDOWS\system32\cmd.exe
El minimo es: -3.122315
El maximo es: 2.350199
El rango es: 5.472514
El numero de bins es: 1024
Your code ran in: 21.617023 msecs.
Presione una tecla para continuar . . .
```

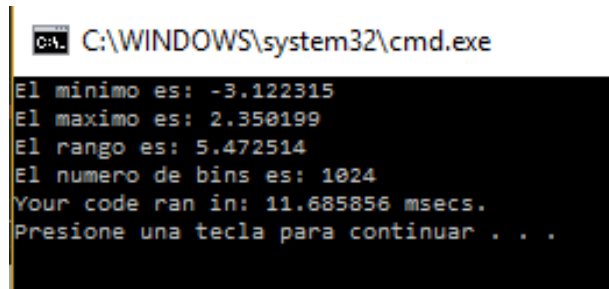
- Con memoria compartida y tamaño de bloque 32

```
C:\WINDOWS\system32\cmd.exe
El minimo es: -3.122315
El maximo es: 2.350199
El rango es: 5.472514
El numero de bins es: 1024
Your code ran in: 14.928608 msecs.
Presione una tecla para continuar . . .
```

- Con memoria compartida y tamaño de bloque 16

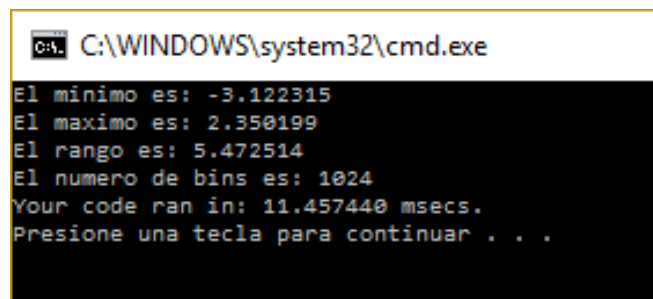
```
C:\WINDOWS\system32\cmd.exe
El minimo es: -3.122315
El maximo es: 2.350199
El rango es: 5.472514
El numero de bins es: 1024
Your code ran in: 22.884544 msecs.
Presione una tecla para continuar . . .
```

- Sin memoria compartida y tamaño de bloque numBins



```
C:\WINDOWS\system32\cmd.exe
El minimo es: -3.122315
El maximo es: 2.350199
El rango es: 5.472514
El numero de bins es: 1024
Your code ran in: 11.685856 msecs.
Presione una tecla para continuar . . .
```

- Con memoria compartida y tamaño de bloque numBins



```
C:\WINDOWS\system32\cmd.exe
El minimo es: -3.122315
El maximo es: 2.350199
El rango es: 5.472514
El numero de bins es: 1024
Your code ran in: 11.457440 msecs.
Presione una tecla para continuar . . .
```

Como se puede apreciar, hay una mejora notable en utilizar tamaño de bloque de numBins con respecto a 16 o 32 como utilizaba en la práctica anterior.

3. Comentarios personales

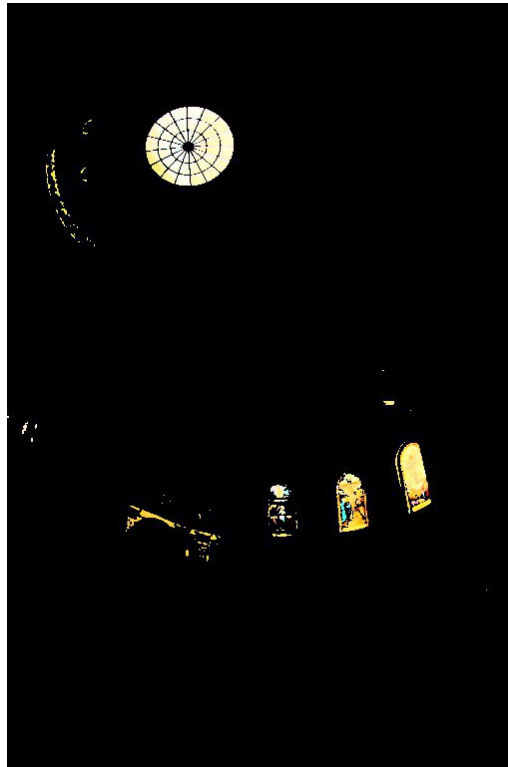
Problemas encontrados

El primer problema importante lo he tenido con los primeros dos kernel de conseguir el máximo y mínimo valor de luminancia.

Primero me salía siempre el mismo valor tanto mínimo como máximo, en este caso fallaba que guardaba la salida del kernel calcularMinimo tanto en mínimo como en máximo .

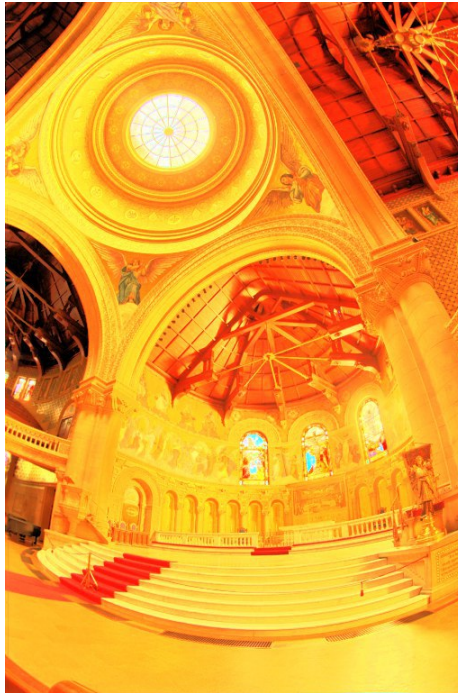
Más tarde, me salían ambos valores negativos. Resulta que estaba llamando al kernel una vez únicamente, sin hacer un bucle e ir reduciendo a la mitad el numero de bloques, por lo que nunca iba reduciendo hasta llegar a un único valor.

Con estos problemas, la imagen me salía prácticamente oscura



Otro problema he tenido ha sido en el histograma, ya que me salía la imagen demasiado iluminada. El fallo que estaba cometiendo era pasarle al kernel el tamaño del histograma y no el de la imagen, por lo que al realizar la comprobación del thread fuera de la imagen, lo hacia incorrecto.

El resultado de este fallo era esta imagen:



Por último, comentar los dos fallos que más me han causado problemas

Ambos fallos están en la función principal, a la hora de llamar al kernel de calcular el máximo y mínimo valor de luminancia.

El primero de ellos es a la hora de liberar memoria de las variables `d_LumIMax` y `d_LumIMin`, que son una copia de `d_LogLuminance` para ir realizando el calculo de los valores. A la hora de realizar `cudaFree` sobre estas variables, me devuelve el fallo *“invalid device pointer cudaFree”*. Este error aún no he conseguido solucionarlo.

El segundo fallo es a la hora de meter una imagen que no sea *memorial_raw_large*. Cuando ejecuto, me devuelve el fallo *“invalid configuration argument cudaGetLastError”*. Finalmente lo solucioné poniendo el tamaño de bloque a `numBins` a todos los kernel, ya que he utilizado unas variables `gridSize` y `BlockSize` diferentes para cada kernel, y sin querer puse la variable `tam_block` del kernel de calcular valores máximos y mínimos a 32.