Laboratorio GPGPU 2013

Este informe muestra los resultados obtenidos al desarrollar algoritmos en c++ y CUDA (Non Local Means) para reducir el ruido de una imagen.

Nota: En el main.cu se encuentra la función de kernel nlm_kernel y la función nlm_CPU. Luego desde el programa principal invocamos a ambos (GPU y CPU) y medimos los tiempos de ejecución de ambos. Definimos como constantes en el cabezal el tamaño de bloque, W, K, etc.

Parte (a): Explicación detallada de la solución, incluyendo optimizaciones que haya hecho.

Como se pidió en la letra del problema, diseñamos la solución para GPU de forma de que cada pixel sea procesado por un único thread.

Además dividimos la imagen en ventanas de 32x32 o 16x16, de esta forma cada una de las áreas obtenidas es procesada por un bloque de 32x32 (o 16x16) threads.

También cargamos primero los pixeles sobre los que va a trabajar cada bloque en memoria compartida. Para esto fue necesario cargar (32+K+W)x(32+K+W) pixeles en memoria compartida (en el caso de bloques de tamaño 32x32), ya que cada pixel necesita de sus vecinos próximos a una distancia de K+W para procesar el valor de su propio pixel.

En un intento de evitar los ifs en el código del kernel (y por lo tanto la divergencia en ejecuciones con su correspondiente baja en los tiempos de procesamiento), decidimos agregarle a la imagen inicial un marco de K+W pixeles negros, de forma de que los pixeles cercanos a los márgenes de la imagen pudieran utilizar estos pixeles redundantes en la ejecución del algoritmo.

Por lo que, al nlm_kernel, se le pasa como parámetro un arreglo de floats de tamaño (width+K+W)x(height+K+W) donde widht y height son las dimensiones de la imagen inicial.

A su vez, para lograr el mayor grado de paralelismo posible en la carga de los pixeles a memoria compartida utilizamos la idea de dividir el área de trabajo de cada bloque (ventana + vecindario) en ventanas. Por ejemplo, si nuestra área de trabajo es 32+K+W y por lo tanto nuestra ventana es de tamaño 32, dividimos (32+K+W)/32 de forma de que cada thread copie a memoria compartida tantos pixeles como le corresponda según la división.

Luego de cargada toda la memoria compartida sincronizamos los threads de cada bloque y a partir de allí con toda la memoria de trabajo en memoria compartida cada thread pasa a trabajar sobre el pixel que le corresponda.

Otro detalle más a comentar surge de una franja negra en la parte inferior que se da en la imagen procesada, esto sucede ya que la altura de la imagen de 787 pixeles no es múltiplo ni de 32 ni de 16, por lo tanto hay una franja de pixeles a la cual no se le asocia threads al correr el kernel de GPU y por lo tanto quedan sin procesar.

Probamos filtrar con valores distintos de sigma e.g (0.45, 0.15 etc) obteniendo imágenes filtradas mas "nítidas" que las que se obtienen con el valor 0.9 (Ver Anexo I).

Parte (b): Una comparación de los tiempos de las versiones en GPU y CPU para los tres tamaños de imagen proporcionados, distintos tamaños de bloque, tamaño de ventana de búsqueda de radio S=10 y tamaños de vecindario de radio w=3 y w=5.

Siempre compilamos con las siguientes flags (compute_20 sm_20)

Solo medimos los tiempos del procesamiento de la imagen y no tomamos en cuenta los involucrados en las transferencias de datos , normalización y/o desnormalizacion de la imagen .

Nota: Los siguientes valores fueron obtenidos en el equipo PCWIN061

Tam Bloques 16*16 SIGMA = 0,9				
TIEMPOS	S = 10;W = 5		S = 10;W = 3	
IMAGEN	CPU	GPU	CPU	GPU
fing (1024*687)	23,62037	0,219028	10,351684	0,062397
fing_xl (2048*1374)	95,090704	0,884169	41,941215	0,250124
fing_xxl (4000*2684)	349,188949	3,393889	154,897273	0,986065

Tam Bloques 32*32 SIGMA = 0,9				
TIEMPOS	S = 10;W = 5		S = 10;W = 3	
IMAGEN	CPU	GPU	CPU	GPU
fing (1024*687)	23,67591	0,263128	10,68185	0,070025
fing_xl (2048*1374)	95,593818	1,052545	41,916753	0,27914
fing_xxl (4000*2684)	352,06007	4,052026	155,795257	1,074232

Comparación 16x16 vs 32x32 GPU SIGMA = 0,9				
TIEMPOS	S = 10;W = 5; 16x16		S = 10;W = 3; 32x32	
IMAGEN	GPU 16x16	GPU 32x32	GPU 16x16	GPU 32x32
fing (1024*687)	0,219028	0,263128	0,062397	0,070025
fing_xl (2048*1374)	0,884169	1,052545	0,250124	0,27914
fing_xxl (4000*2684)	3,393889	4,052026	0,986065	1,074232

En todos los casos de prueba podemos ver como los tiempos obtenidos en la GPU son inferiores en comparación con los obtenidos en la CPU, aunque aumentamos la cantidad de hilos a 32x32 en la segunda tabla no vemos una mejora significante en los tiempos obtenidos
Como vemos los mejores tiempos en la GPU se logran para la configuración de 16X16 hilos.

Nota: Los siguientes valores fueron obtenidos en el equipo PCWIN048

Tam Bloques 16*16 SIGMA = 0,9				
TIEMPOS	S = 10;W = 5		S = 10;W = 3	
IMAGEN	CPU	GPU	CPU	GPU
fing (1024*687)	17,717866	0,263505	7,793918	0,070019
fing_xl (2048*1374)	70,765826	1,064468	31,358505	0,282452
fing_xxl (4000*2684)	263,572609	4,081138	115,297433	1,082136

Tam Bloques 32*32 SIGMA = 0,9				
TIEMPOS	S = 10;W = 5		S = 10;W = 3	
IMAGEN	CPU	GPU	CPU	GPU
fing (1024*687)	17,540942	0,262928	7,711452	0,069826
fing_xl (2048*1374)	70,569408	1,052596	31,585162	0,278967
fing_xxl (4000*2684)	262,715508	4,051911	114,710499	1,074232

En casi todos los casos de prueba podemos ver como los tiempos obtenidos en la GPU son inferiores en comparación con los obtenidos en la CPU y que los tiempos obtenidos con bloques de 32*32 son mejores a los de 16*16 hilos.

Anexo I

Las siguientes imágenes fueron obtenidas con la configuración siguiente:

```
#define K 10
#define W 5
#define DIM_BLOQUE 32
#define SIGMA 0.45
char path[20] = "img\\fing.pgm";
```


Fing - CPU
Fing - GPU