Laboratorio del curso GPGPU

Introducción

Las imágenes resultan distorsionadas en mayor o menor medida durante su adquisición por la introducción de cierta cantidad de ruido. Non Local Means (NLM) es un método muy popular, propuesto por Buades et. al. [1] para reducir el ruido de una imagen dada su efectividad y simplicidad. Su principal desventaja es el alto costo computacional que implica. El trabajo consiste en construir una versión en GPU del algoritmo siguiendo las pautas que se detallan en las secciones siguientes.

A continuación se brindará una breve descripción del algoritmo. Luego, se describirán las ideas principales para construir la versión paralela. Por último se mencionarán algunos consejos para resolver las distintas etapas del obligatorio.

Algoritmo

Para el caso de las imágenes en escala de grises, el método NLM consiste en sustituir el valor de intensidad de cada pixel por un promedio ponderado de la intensidad de pixels similares de la imagen (por cuestiones de complejidad computacional se restringe esto a un área de búsqueda centrada en el pixel). La diferencia fundamental con otros métodos radica en que la similitud entre dos pixels se calcula comparando un vecindario centrado en cada pixel y no únicamente sus valores de intensidad.

La ecuación que representa este algoritmo es la siguiente:

$$\tilde{I}(p) = \frac{1}{Z(p)} \sum_{(k)\in S_p} e^{-\frac{\|N_p - N_k\|^2}{h^2}} I(k) \tag{1}$$

donde S_p es la ventana de búsqueda de tamaño $K \times K$ centrada en el píxel p, N_p yN_k son vecindarios de tamaño $W \times W$ centrados en los pixel p y k, $Z(p) = \sum_{(k)\in S_p} e^{-\frac{\|N_p - N_k\|^2}{h^2}}$ es una constante de normalización y h es un parámetro del algoritmo que debiera ser proporcional a la varianza del ruido de la imagen.

```
for (px = 0 ; px < imagen.width ; px++)
   for (py = 0 ; py < imagen.height ; py++)

      for (sx = px - K/2 ; sx < px + K/2 ; sx++)
         for (sy = py - K/2 ; sy < py + K/2 ; sy++)

            for (wx = - W/2 ; wx < W/2 ; wx++)
               for (wy = - W/2 ; wy < W/2 ; wy++)

                  dist += ( imagen(px + wx, py + wy) - imagen(sx + wx, sy + wy) )^2
               end
            end

            peso = exp(-dist/h^2)

            suma += imagen(sx,sy) * peso
            const += peso
         end
      end

      out(px, py) = suma / const;
   end
end
```

Figura 1: Código secuencial del algoritmo Non Local Means.

De la descripción del algoritmo en la sección anterior se desprende inmediatamente que el procesamiento de cada pixel depende solamente de los valores de la imagen ruidosa y no de los valores de pixels ya procesados.

Aprovechando las características de las GPU, es natural paralelizar el problema de forma que cada thread procese un pixel de la imagen en paralelo. La imagen se divide en bloques de $N \times M$ pixels que serán procesados concurrentemente por bloques de $N \times M$ threads en GPU. El cálculo de la distancia entre dos vecindarios se basa en sumar el cuadrado de la diferencia elemento a elemento de ambos vecindarios, por lo que se implementa como una recorrida secuencial de $W \times W$ elementos. En una misma ventana de búsqueda se comparan los vecindarios de cada pixel de la ventana con el del pixel central.

Consigna

Debe producirse una versión secuencial y una versión en CUDA-C que transfiera la imagen al dispositivo, aplique el filtro y luego la trasfiera al host para poder presentarla.

La versión en CUDA-C **debe** utilizar la memoria compartida y funcionar para cualquier tamaño de imagen (aunque pueden asumirse tamaños de imagen múltiplos de 32).

Cada bloque **debe** trabajar de forma independiente sobre una zona de la imagen.

Compare los tiempos de ejecución obtenidos para las tres imágenes proporcionadas con tamaños de bloque (ventana de búsqueda) de 16×16 y 32×32.

Se proporciona un proyecto en Visual Studio el cual utiliza la biblioteca CImg para la carga y la presentación de las imágenes.

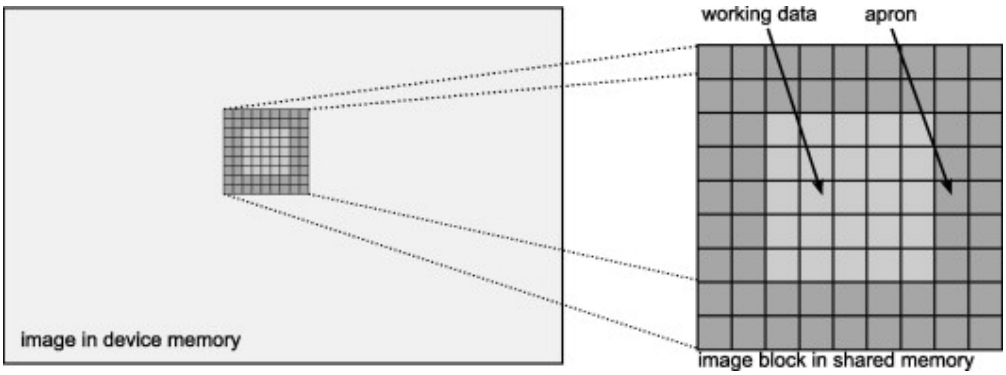

Figura 2: Datos cargados en memoria compartida

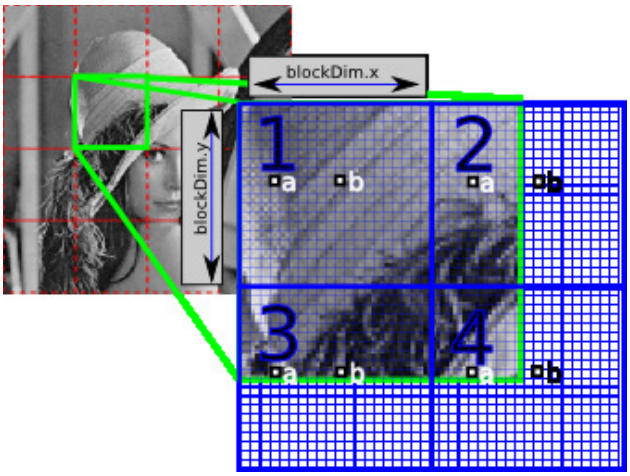

Figura 3: Carga iterativa de los datos. Aquí el thread de la grilla marcado con la letra *a* carga 4 elementos de la zona de la imagen. El thread marcado con *b* carga únicamente 2, quedando inactivo en las iteraciones 2 y 4.

Consejos

Para implementar el algoritmo en GPU se deberá dividir la imagen en bloques con el fin de cargarla en memoria compartida. Sin embargo los threads que operen sobre el borde de cada bloque deberán acceder a pixels (que están dentro de su ventana) pertenecientes a otra zona de la imagen. En esta versión, deberán cargarse en memoria compartida los pixels correspondientes a cada thread y además aquellos pixels que pertenecen a la ventana de los threads del borde (ventana de búsqueda + vecindario), tal como muestra la Figura 2.

La carga a memoria compartida debe llevarse a cabo con el mayor grado de paralelismo posible pero utilizando únicamente la cantidad de threads necesaria para realizar el cómputo, es decir, algunos hilos deberán cargar más de un valor debido a los "bordes". Esta carga puede realizarse iterativamente como se muestra en la Figura 3.

Se recomienda traducir la imagen (la cual está formada por enteros entre 0 y 255) a flotantes entre 0 y 1, antes de aplicar la función del filtro.

Entregar

1. Un archivo nlm.cu que contenga el código de las funciones *nlm_ CPU*, *nlm_ GPU* y *nlm_ kernel*.
2. Archivos de cabecera en caso de existir.
3. Archivo main.cu con la función main() del programa. La solución debe funcionar reemplazandoagregando los archivos entregados en el proyecto de Visual Studio proporcionado (no debe entregarse el proyecto de Visual Studio).
4. Un informe que contenga:
   a) Una explicación detallada de la solución, incluyendo optimizaciones que haya hecho.
   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 $w=5$.

Referencias

[1] A. Buades, B. Coll, J.M. Morel A non local algorithm for image denoising IEEE Computer Vision and Pattern Recognition 2005, Vol 2, pp: 60-65, 2005.