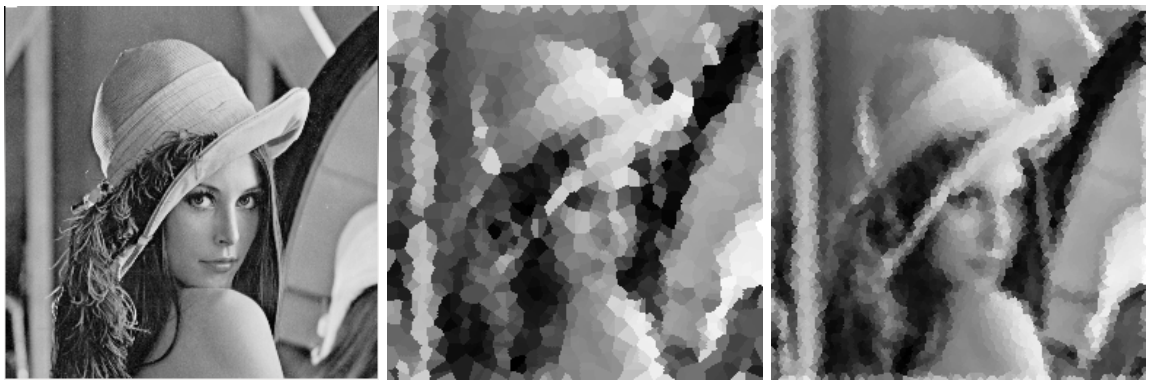
Laboratorio del curso GPGPU

Introducción

Dado cierto conjunto de puntos en un plano a los que llamaremos *centros*, el diagrama de Voronoi correspondiente divide dicho plano de acuerdo con la regla del vecino más cercano: Cada punto del plano pertenece a una región asociada con el centro más cercano.

En los últimos años se ha despertado un creciente interés por esta particular construcción geométrica. Al parecer, existen tres razones principales: En primer lugar, los diagramas de Voronoi aparecen en la naturaleza en varias situaciones. De hecho, algunos procesos naturales pueden ser utilizados para definir clases particulares de diagramas de Voronoi. En segundo lugar, los diagramas de Voronoi tienen propiedades matemáticas muy interesantes, y están relacionados con muchas estructuras geométricas conocidas. Por último, son una poderosa herramienta para resolver diversos problemas computacionales [1] en áreas como robótica, sistemas de información geográfica, sistemas operativos, etc.

Por estas razones, es necesario desarrollar herramientas para computar eficientemente este tipo de estructuras. En este obligatorio utilizaremos los diagramas de Voronoi para filtrar una imagen creando un efecto artístico. El efecto consistirá en deformar la imagen para convertirla en polígonos de colores que se asemejen a la imagen original.



Algoritmo

Una imagen en escala de grises puede representarse como una matriz en la que cada entrada contiene el valor de intensidad de un pixel.

El efecto que se busca lograr consiste en sustituir la imagen original por un conjunto de polígonos convexos que cubra la imagen en su totalidad. Para delimitar estos polígonos utilizaremos el concepto de diagrama de Voronoi, donde la imagen de entrada se interpretará como una matriz de tamaño $m \times n$, donde cada entrada a_{ij} de la matriz representa un punto de coordenadas (i, j) en un espacio eucídeo. Para construir el diagrama se eligen k centros de forma aleatoria, y luego se asigna a cada entrada de la matriz el valor que identifique al centro que se encuentra a menor distancia.

La distancia euclídea entre el punto (i, j) y el centro $c = (c_i, c_j)$ se define como:

$$\sqrt{(i - c_i)^2 + (j - c_j)^2} \tag{1}$$

Para que el diagrama resultante se asemeje a la imagen original el color de cada polígono será el promedio de los colores de un conjunto de pixels cercanos al centro del mismo.

El algoritmo para resolver el obligatorio puede entonces separarse en dos partes:

- En una primera etapa se construirá una imagen *promedio* en la cual se sustituye el valor de cada pixel por un promedio del valor de los pixels pertenecientes a una ventana centrada en el mismo.
- En la segunda etapa se elegirán aleatoriamente las coordenadas de una cierta cantidad de centros y se construirá un diagrama de Voronoi en el cual la celda de Voronoi correspondiente al centro de coordenadas (c_i, c_j) sea del color del pixel (c_i, c_j) en la imagen *promedio*.

Consigna

Parte 1

Realizar una versión secuencial del algoritmo. El resultado debe ser una imagen en la que cada pixel se encuentre pintado con el color correspondiente al centro más cercano, el cual debe ser un promedio de los pixels pertenecientes a una ventana de tamaño 5×5 centrada en dicho centro.

Parte 2

Realizar un kernel en CUDA que tome como entrada una imagen en escala de grises y calcule la imagen promedio para un tamaño de ventana de 5×5. Esta operación es un caso particular de la convolución vista en el práctico del curso, en el cual la máscara está formada por unos y se divide el resultado de la convolución entre el tamaño de la máscara.

Para realizar la operación DEBE utilizarse la memoria compartida.

Parte 3

Construir una función que realice el sorteo de las coordenadas correspondientes a los centros (la función debe recibir la cantidad de centros como parámetro) y luego invoque un kernel en CUDA que tome como entrada la imagen promedio calculada en la parte anterior, y un arreglo con las coordenadas de cada centro; y devuelva otra imagen con el diagrama de Voronoi correspondiente.

Parte 4

Compare los tiempos de ejecución obtenidos por las versiones secuencial (Parte 1) y paralela (Parte 2 + Parte 3) para las tres imágenes proporcionadas con tamaños de bloque de 16×16, 32×32. Para el mejor tamaño de bloque compare los tiempos de ejecución obtenidos al variar la cantidad de centros.

Se proporciona un proyecto en Visual Studio el cual utiliza la biblioteca CImg para la carga y la presentación de las imágenes. Esta biblioteca funciona también en Linux. No es necesario utilizar (ni debe entregarse) la solución en Visual Studio.

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 1.

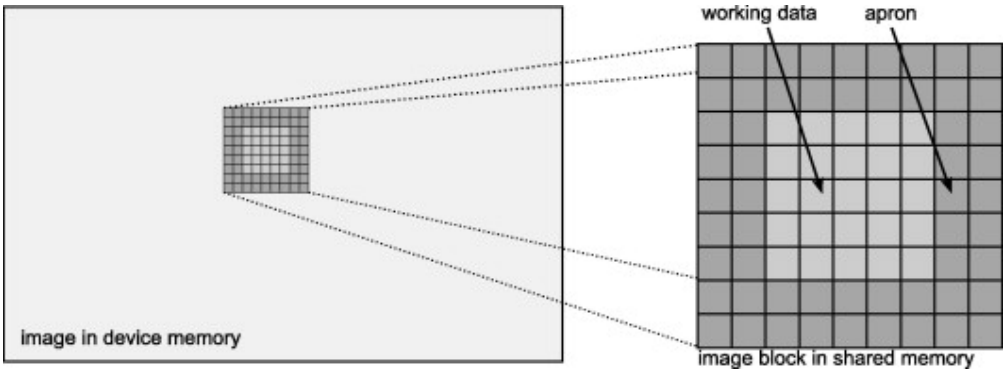


Figura 1: Datos cargados en memoria compartida.

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 2.

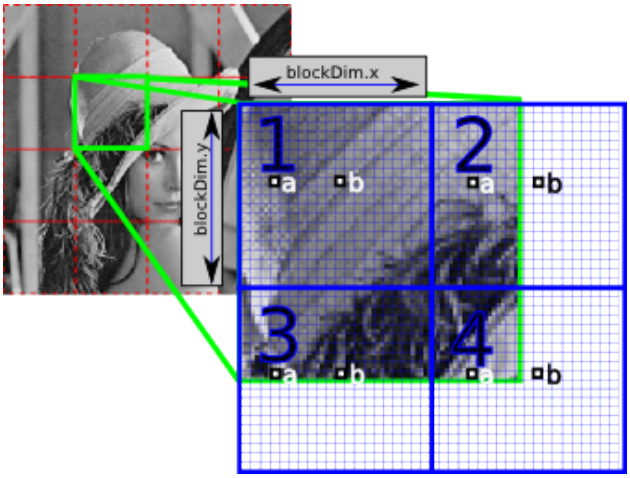


Figura 2: Carga iterativa de los datos. Aquí el thread de la grilla marcado con la letra *a* carga 4 elementos de la zona de la imagen. El thread marcado con *b* carga únicamente dos (en las iteraciones 1 y 3), quedando inactivo en las iteraciones 2 y 4.

Entregar

1. Un archivo voronoi.cu que contenga el código de las funciones *voronoi_CPU*, *voronoi_GPU*, y los kernels en CUDA correspondientes a las Partes 2 y 3.

2. Archivos de cabecera en caso de existir.

3. Archivo main.cu con la función main() del programa. La solución debe funcionar reemplazando/agregando 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 como la que se pide en la Parte 4.

Referencias

[1] Aurenhammer, Franz Voronoi diagrams - a survey of a fundamental geometric data structure, ACM Computing Surveys (CSUR), Vol 23, pp: 345-405, 1991.