Tarea 4 IMT2112

Pablo Rademacher

27 de noviembre de 2020

Como determinar si un punto pertenece al set:

En mi código, yo cree dos arrays de n numeros cada uno. El primer array representa la parte real y el segunda la parte imaginaria de cada numero, como dicen sus nombres. Así, el numero k esta dado por real[k] $+i\cdot$imaginaria[k].

Para ver si un numero pertenece o no al set, le aplico la iteración hasta un maximo de ITER_MAX veces. Si en algún momento me da un resultado de valor absoluto mayor a 2, entonces paro la iteracion de inmediato ya que esto implica que la secuencia divergerá. Si no, asumo que está dentro. Esta decision implica que mi algoritmo sobreestima el area del set, ya que hay puntos que no están en el pero se demoran mucho en diverger.

¿random en GPU?

Una razón por la que no se pueden crear los numeroas aleatorios en la GPU directamente es que estos numeros no son realmente aleatorios: Cada numero es creado a partir de una seed. Así, si dos generadores aleatorios tienen la misma seed, estos generarán la misma secuencia de números.

El problema con esto es que si creamos los numeros directamente en la GPU, la probabilidad de que la seed sea igual entre un thread y otro es muy alta, por lo que ambos van a tener la misma secuencia de numeros. Esto no es muy eficiente, pues nos gustaría que ambos tuvieran distintos numeros, sino no sería necesario paralelizar.

Calculo de la superficie:

Yo hice que mi código retornará la cantidad de iteraciones que le tomo al algoritmo salir del while. Si este número es igual a ITER_MAX, entonces el punto está dentro, si es menor, significa que en algún momento la iteracion produjo un numero de modulo mayor a 2, asi que diverge y no esta en el set.

Una vez tengo este vector, le aplico una operacion filter, la cual me devuelve un objeto cuyas componentes son las del iterable original que cumplan una condición, en nuestro caso ser igual a ITER_MAX. Luego le saco el largo a este vector, lo que me entrega la cantidad de puntos en el set.

Ahora, divido esta cantidad por la cantidad de puntos generados originalmente, lo que me da la proporción del area original de donde saqué los puntos, y multiplico por 16 (Ya que los puntos salen del cuadrado$[-2, 2]^2$).

Fuentes:

Me ayudaron demasiado las siguientes páginas y actividades:

- La ayudantía de pyOpenCL <3.
- `https://github.com/inducer/pyopencl/blob/master/examples/demo.py`: Una forma un poco más fácil (quizá no tan eficiente) de usar buffer.
- `https://www.codingame.com/playgrounds/2358/how-to-plot-the-mandelbrot-set/mandelbrot-set`: Información sobre como programar el set de Mandelbrot.

README:

- Creo importante mencionar que para ejecutar la tarea use Google Colab, por eso está la primera celda donde instalo pyopencl. Esta la puedes ignorar.

- Si llegas a querer hacer variar la cantidad de iteraciones, deberás modificar el for del codigo a entregar en a la GPU (el que está como string) y el filter de la antepenultima celda. Puedes modificar tambien la cantidad de datos, pero a una escala muy grande esto no haceque varíe mucho la cantidad calculada.
- La operacion de reduccion no la hice en la GPU, sino en CPU. En cuantos a los if-else statements, mi solución para evitar sus complicaciones fue simplemente no usarlos