Pontificia Universidad Católica de Chile
Departamento de Ingeniería Matemática y Computacional
IMT2112 Algoritmos paralelos en computación cientifica
Profesor: Elwin Van T'Wout
Estudiante: Williams Medina

Tarea 4

2. El algoritmo que utilice para calcular el área del fractal fue una implementación en python del método pyxel counting, en donde a partir de los números complejos $c = a_c + ib_c$ (en el apartado 3 especificaré como los cree) calculo la serie $x_0 = 0$, $x_n = x_{n-1}^2 + c$ para $n\_iteraciones$ términos, es importante notar que debido a la forma de la sucesión, el nuevo número complejo creado tendrá la forma $x_n = (a_{n-1}^2 - b_{n-1}^2 + a_c) + (2a_{n-1}b_{n-1} + b_c)i$, y como todos los valores $a_c, b_c \in [-2, 2]$ tenemos que el área del fractal debe estar sobre la superficie que genera estos puntos, así si la serie $x_n = x_{n-1}^2 + c$ converge, podemos calcular el área del fractal como la proporción de complejos que convergen en la serie , dicho de otra forma área $= 16 \cdot \frac{c\_conver}{n}$, donde c_conver = cantidad de complejos convergentes y n = cantidad total de complejos . De aquí tenemos que si al valor esperado 1,50659177 le restamos el valor de aproximación obtenemos el error de aproximación, el cual disminuye en función de la cant de iteraciones..
Es importante notar que para ver si la sucesión x_n es convergente solo basta ver que la distancia del punto c (presente en x_n) al origen debe ser menor que el intervalo en el que se encuentra, en otras palabras para nuestros complejos $c = a_c + ib_c$ con $a_c, b_c \in [-2, 2]$ se debe cumplir que $a_c^2 + b_c^2 \leq 2^2$ para que sea convergente, en caso contrario la serie divergerá. A partir de aquí, es evidente ver entonces que dependiendo de la cantidad de iteraciones puede que una serie sea o no convergente, con lo que aumentar la cantidad de iteraciones, solo podrá aumentar la cantidad de complejos no convergentes.

3.En primer lugar, todo el código fue realizado en python utilizando la libre de pyopencl. Así,para generar los valores de c utilice la función $np.random.uniform(low = -2, high = 2, size = (n,))$ tanto para a_c como para b_c, generando así n complejos con $a_c, b_c \in [-2, 2]$.
En cuanto a la generación de números aleatorios debemos recordar que los números aleatorios, en realidad son pseudo aleatorios y estos siguen una secuencia. Esta secuencia cumple con distribuir uniforme entre 0 y RAND_MAX, pero cada número depende del anterior. Con lo que se comienza con una semilla inicial y en cada llamado, se actualiza la semilla para generar el siguiente número aleatorio. De esta manera, la generación de números aleatorios debe realizarse secuencialmente. Si se paralelizara, todos los núcleos generarían los mismos números aleatorios y se perdería la aleatoriedad. Como la GPU funciona en con varios núcles en paralelo, habría que desaprovechar muchos núcleos para generar lo números aleatorios de forma secuencial, haciendolo poco eficiente en cuánto a rapidez.

4. La condición $z_n > 4$ corresponde a una condición $if - else$, esta para que no fuera ineficiente fue implementada de manera que todos los núcleos de la GPU no tengan tiempos de espera demasiado distintos entre ellos, ya que como sabemos todos los núcleos ejecutan la misma instrucción, con lo que algunos pueden estar haciendo nada, mientras otros trabajan. Esto se debe a que si una rama del $if - else$ es más larga que la otra, los procesadores que les tocó esa rama, tendrán que trabajar por más tiempo, mientras los que les tocó la más corta estarán esperando, haciendo que este proceso sea ineficiente en cuanto a tiempo.

Impotante: Por alguna razón, al aumentar los valores $n > 100$ y $m > 100$ la matriz d_h que contiene la cantidad de complejos que divergieron deja alguna columnas en 0, pero de igual forma el resultado coincide con el método secuencial que implemente a modo de comparación.