NOMBRE: Diego Castelli PUNTAJE:

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ingeniería Matemática y Computacional

IMT2112 — Algoritmos Paralelos en Computación Científica — 2' 2020

Tarea 4

1.

Tras computar el código opencl-devices.cpp, se imprimió lo siguiente:

1. Device: GeForce MX110
1.1 Hardware version: OpenCL 1.2 CUDA
1.2 Software version: 457.30
1.3 OpenCL C version: OpenCL C 1.2
1.4 Parallel compute units: 2
1. Device: Intel(R) UHD Graphics 620
1.1 Hardware version: OpenCL 3.0 NEO
1.2 Software version: 27.20.100.8935
1.3 OpenCL C version: OpenCL C 3.0
1.4 Parallel compute units: 24

2.

a) Se generan de manera aleatoria que luego se guardaran como dos vectores de largo K, c1 y c2, tal que

$$c = c_1 + ic_2$$

luego basta generar dos float en el rango $[-2, 1]$ para c_1, y $[-1, 1]$ de la forma:

$$\text{largo del intervalo } \cdot \frac{rand()}{RAND\_MAX} + \text{ínfimo del intervalo}$$

que representa el rectángulo en el plano complejo en donde está contenido el conjunto de Mandelbrot.

b) Basta ver que si un elemento pertenece al conjunto, no puede tener ningún valor de la iteración con módulo mayor a 2 o Nan. es decir dado un N, y sea $c \in \mathbb{C}$, luego si definimos c_n como el n-ésimo termino de la iteración, basta verificar que $\forall n \leq N$ si $|c_n| > 2$, si no se cumple para ningún n luego pertenece al conjunto de Mandelbrot, y no pertenece en otro caso.

c) Se escoge como termino razonable 5000 iteraciones en la variable n_it pero que se puede modificar si se desea.

3.

a) Se generan K números complejos usando como semilla srand(time(0)), donde $K = 10000$, pero se puede modificar a placer.

b) No se recomienda generar valores aleatorios en paralelo, pues no es realmente aleatorio, y al generar la semilla que dará origen al valor "aleatorio" será en función del tiempo, luego al correrlo en paralelo se ejecutarán prácticamente al mismo tiempo y luego saldrán valores muy similares lo cual no es deseable para este experimento.

4.

a) Se itera utilizando c_n, y c_{n-1}.

b) El reduce se realiza en grupos guardando las sumas parciales, que luego se suman en la CPU.

c) no se evalúa hasta finalizar la iteración, y en la CPU, donde se revisa que sea de módulo mayor a 2 o Nan. De esta manera se evita por completo en la GPU el realizar *if-else conditions.*

5.