Pontificia Universidad Católica de Chile
Escuela de Ingenieria
IMT2112 - Algoritmos Paralelos en Computación Científica

Tarea 4

25 de noviembre de 2020

2º semestre 2020 - Elwin van't Wout

Melanie Esmeralda Pacheco Riffo

Pregunta 2

a) **Mencionan como eligen los puntos c.**

La elección y creación de los números complejos c fue por medio de dos vectores aleatorios de n elementos: a, b generados por una distribución uniforme, donde uno contiene la parte real (vector a) del un número que con valores entre -2 y 1 y el otro la parte imaginaria (vector b) con valores entre -1.5 y 1.5. De esta forma el número complejo i se verá de la forma $c[i] = a[i] + i \cdot b[i]$

La elección de los límites fue en base a cómo se iba dibujando la figura y el artículo `https://userswww.pd.infn.it/~lacaprar/Didattica/C++/Complex/Area%20of%20the%20Mandelbrot%20Set.pdf`

b) **Mencionan como se puede determinar que un punto está adentro o afuera el fractal.**

Para determinar si el punto está dentro o fuera se hizo una cantidad de 400 iteraciones bajo la serie $x_0 = 0,\ x_n = x_{n-1} + c$ donde c es el número complejo a verificar si está dentro o fuera. Como se utilizaron vectores (a y b) en representación de números complejos de la forma $c = a + i \cdot b$,en la iteración i la parte real es de la forma $a_i = a_{i-1}^2 - b_{i-1}^2 + a$ y la parte imaginaria es de la forma $b_i = 2 \cdot a_{i-1} \cdot b_{i-1} + b$. Finalmente se calcula la norma de número generado al terminar las iteraciones $c_n = a_n + i \cdot b_n$ dado por $\sqrt{a_n^2 + b_n^2}$ y se compara si este tiene una norma menor a 2, si es así este está dentro del fractal.

c) **Se puede usar un máximo al número de iteraciones x_n, elegido deforma razonable**

Si se pude usar un máximo de iteraciones que entregue un resultado aproximado al área, debido a que al calcular el cuadrado de la sucesión anterior en cada sucesión permite que si el número a verificar está fuera del rango el valor de cada iteración vaya creciendo de forma exponencial y por lo tanto se vaya a infinito. Esto se puede ver ya que si se tiene $|z_n| \geq 2$ y $|x_n| \geq c$ luego

$$|x_{n+1}| - |c| = |x_n^2 + c| - |c| > 2(|x_n| - |c|)$$
$$\Rightarrow |x_{x+m}| > |c| + 2^m(|x_m| - |c|)$$

Por lo que se puede ver que su límite inferior crece al infinito a medida que aumentan las iteraciones. En la tarea de ocuparon 400 iteraciones ya que al aumentar iteraciones no variaba mucho el resultado del área calculada a lo que esperaba (depende también de la precisión deseada).

Pregunta 3

b) **Expliquen por qué no se puede paralelizar la generación de números aleatorios y, por lo tanto, no será eficiente hacerlo en la GPU.Sugerencia:la función es un '*pseudorandom* number generator'.**

No es buena idea hacer un random en la GPU ya que al ser un pseudo código, este toma un *seed* específico de forma que si se ejecuta paralelamente probablemente los números serían iguales o muy parecidos. Hay formas de prevenir ese proceso pero es mucho más sencillo programar en la CPU los ramdon.

Pregunta 4

b) **Calcular el superficie, es decir, la proporción de puntos adentro del fractal, debe ser ejecutado en la GPU. Expliquen como programaron esta operación de reducción**

Para calcular la proporción de puntos que se encuentran dentro del fractal se ocupó un vector auxiliar de 0 y tamaño igual a la cantidad de puntos, en donde se pone un 1, si en la posición del punto a ver tiene norma menor a 2. Luego en otro kernel se realiza una suma en paralelo del vector de 0 y 1 en donde todos los hilos con número designado 0 del grupo al que corresponden todos suman los valores de su grupo y guardan en el valor en un vector con tamaño igual a la cantidad de grupos que será finalmente sumado por la operación *sum* de numpy. Finalmente la proporción será la suma obtenida dividido la cantidad de puntos creados en un inicio.

c) **Expliquen como programaron los if-else statements y si es eficiente hacerlo cada iteración.**

Para hacer la la programación de *if-else statements* se tuvo cuidado de evitar condiciones de carrera, para ello se crearon dos kernel. El primer kernel calcula las iteraciones y la norma, en este paso no hay problemas ya que todos los hilos están haciendo lo mismo al no haber *if-else*,luego se verifica si se el punto se encuentra dentro o fuera del fractal por medio de un *if* por lo que van haber puntos que no pasen por el *if* y terminen antes que lo que si entraron. Debido a lo anterior fue necesario crear otro Kernel que sumase los elementos de un vector de forma paralela (como fue explicado anteriormente) para evitar condiciones de carrera.

Referencias

- https://stackoverflow.com/questions/9912143/how-to-get-a-random-number-in-opencl

- https://userswww.pd.infn.it/ lacaprar/Didattica/C++/Complex/Area

- https://renatofonseca.net/mandelbrotset