Tarea 4: Introducción a la computación de alto rendimiento (6%)

Marlon Brenes*

El propósito de esta tarea es practicar los conceptos de paralelismo de memoria compartida y distribuida. La tarea involucra entregar código fuente de C++ (utilizando bibliotecas especializadas) **con su respectivo reporte detallando el análisis**. Usted debe entregar los siguientes archivos para ser evaluados:

- `mandelbrot.cpp`
- `dot_product.cpp`
- reporte4.pdf/reporte4.doc/reporte4.docx/reporte4.ipynb (dependiendo de cual formato escoja para su reporte)

PARTE I: MEMORIA COMPARTIDA (3.0%)

El conjunto de Mandelbrot es un conjunto de dos dimensiones que exhibe comportamiento caótico y fractal. A pesar de la sencillez de la regla que genera el conjunto, la complejidad asociada al conjunto es altamente magnificada. El conjunto está definido en el plano complejo como el conjunto de números c tales que la función

$$f_c(z) = z^2 + c \tag{1}$$

no diverge al infinito cuando se inicia la iteración con $z = 0$. Esto es, cuando la secuencia $[f_c(0), f_c(f_c(0)), \cdots]$ permanece acotada en su valor absoluto.

Al visualizar los números c que satisfacen esta condición en el plano complejo (Fig. 1), se obtiene un **fractal**. Note que $\mathrm{Re}[c] \in [-2, 1]$ y $\mathrm{Im}[c] = [-1, 1]$.

- Se le ha proporcionado una aplicación en `C++` que genera el conjunto de Mandelbrot e imprime los resultados en formato `ASCII` en la terminal
- La aplicación es muy rudimentaria, dado que no se imprimen distintos símbolos dependiendo del valor de acotación mencionado anteriormente. Lo que hace es imprimir '#' o '.' dependiendo si c da lugar a iteraciones acotadas después de cierto valor de corte. Para visualizar el conjunto en la terminal, asegúrese de que la terminal tenga el tamaño de al menos 155x50 caracteres

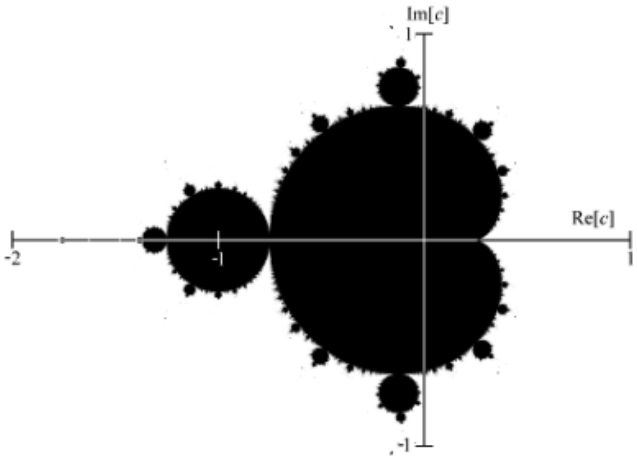

FIG. 1. Conjunto de Mandelbrot

* marlon.brenes@utoronto.ca

- No obstante, la aplicación es suficiente para observar los detalles mínimos del set de Mandelbrot
- **Su tarea** es encontrar una metodología para paralelizar el procedimiento que genera el conjunto utilizando OpenMP en el paradigma de memoria compartida.
- Sus tareas son las siguientes:
  - Reproducir el conjunto de Mandelbrot mediante símbolos en la terminal utilizando la aplicación que se le ha entregado y entender la funcionalidad (0.25%)
  - Diseñar una estrategia de paralelización. Que parte del trabajo se puede hacer de forma concurrente? Explique su estrategia en su reporte (0.75%).
  - Una aplicación en paralelo debe dar lugar a los mismos resultados que una aplicación en serie: su aplicación acelerada con OpenMP **debe generar el mismo gráfico en la terminal al que genera la aplicación en serie**. Explique en su reporte que estrategia utilizó para lograr este resultado (1%).
  - Código `mandelbrot.cpp` paralelizado: (1%)

No se preocupe por la escalabilidad. En general, este algoritmo permite aceleración que escala muy favorablemente, sin embargo, para observar estos resultados se deben estudiar grillas más grandes. Si desea observar el beneficio de la aceleración del algoritmo, puede modificar los parámetros `width` y `height` en `mandelbrot.cpp` para estudiar grillas más grandes. Este ejercicio no es requerido para la tarea.

PARTE II: MEMORIA DISTRIBUIDA (3.0%)

Su tarea es implementar el producto interno de dos vectores de tamaño N utilizando paralelismo de memoria distribuida. La operación corresponde a:

$$k = \mathbf{a}^T \cdot \mathbf{b} = \begin{bmatrix} a_1 & a_2 & \cdots & a_N \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \tag{2}$$

Para implementar la operación, siga las siguientes instrucciones:

- Inicialice el ambiente de `MPI` como vimos en clase (0.25%)
- Declare 4 vectores usando `std::vector` como arreglos de datos: `a`, `b`, `local_a` y `local_b` (0.25%)
- En el proceso 0, utilice el método `resize()` para asignar suficiente memoria para N elementos para los vectores `a` y `b` (0.25%)
- Para todos los procesos, utilice el método `resize()` para asignar suficiente memoria para `nlocal` elementos para los vectores `local_a` y `local_b`. Defina `nlocal` como lo hicimos en clase. Puede asumir que el número de elementos N es un múltiplo del número de procesos `size` (i.e., no hay `rest`) (0.5%)
- En el proceso 0, asigne distintos valores a las entradas de los vectores `a` y `b`. Asigne estos valores de manera tal que $\mathbf{a} = \mathbf{b} = [1,\ 2,\ 3,\ \cdots\ N]$ (0.25%)
- Investigue y utilice la función `MPI_Scatter` para distribuir las secciones de los vectores `a` y `b` a sus respectivos procesos (0.5%)
- Calcule el producto interno de cada sección de tamaño `nlocal` en cada proceso (0.25%)
- Utilice `MPI_Reduce` para calcular el producto interno total con base en los productos internos de cada proceso (0.5%)
- Imprima el resultado usando el proceso 0 (0.25%)

No se preocupe por la escalabilidad. En general, este algoritmo permite aceleración que escala muy favorablemente, sin embargo, para observar estos resultados se deben estudiar vectores grandes. Si desea observar el beneficio de la aceleración del algoritmo, puede modificar el parámetro N para estudiar vectores grandes. Este ejercicio no es requerido para la tarea.