Programación Concurrente

Trabajo Práctico

Se desea implementar en Java un programa que grafique el *conjunto de Mandelbrot*. Este conjunto es lo que se conoce como un *fractal*, una estructura geométrica que se repite cuando es observada a diferentes escalas.

El conjunto lleva el nombre de Benoit Mandelbrot, matemático polaco que lo estudió en los años 70 y 80, y se lo suele tomar como un ejemplo de una estructura compleja derivada de reglas sencillas. Además, se lo suele considerar visualmente atractivo (Figura 1).

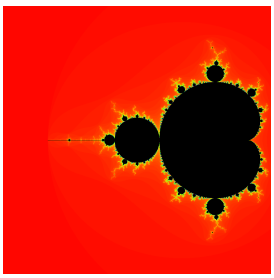

Figura 1: Conjunto de Mandelbrot, en este caso mostrando el rango $(-2{,}5; 0{,}5)$ en el eje x y $(1{,}5; -1{,}5)$ en el eje y.

Los elementos del conjunto de Mandelbrot son números complejos. Recordemos que un número complejo $c \in \mathbb{C}$ es de la forma $x + iy$, donde i es la *unidad imaginaria*, un valor que es solución de $x^2 + 1 = 0$. A x e y se los llama la *parte real* de c y su *parte imaginaria*, respectivamente. Un número $c \in \mathbb{C}$ pertenece al conjunto de Mandelbrot si la sucesión

$$z_0 = 0$$
$$z_n = (z_{n-1})^2 + c$$

no diverge, es decir, si $|z_k| < K$ para algún valor real K. El número $z_n^2 = (x_n + y_n)^2$ se puede calcular como $(x_n^2 - y_n^2) + i2x_n y_n$.

Cuando se representa al conjunto de Mandelbrot en una imágen, cada píxel representa un número complejo c donde la posición en el eje x representa la parte real de c y la posición en el eje y su parte imaginaria. Luego, para cada pixel se evalúa si su valor complejo c correspondiente pertenece al conjunto de Mandelbrot. Para hacerlo, se fija un valor N y se calcula $z_N = x_N + iy_N$ para c. Si $(x_N^2 + y_N^2) > 4$, entonces es posible deducir que z_n va a diverger para c, por lo que c no pertenece al conjunto. En cambio, si $(x_N^2 + y_N^2) \leq 4$, se considera que c está en el conjunto. Esto se representa pintando al píxel correspondiente a c de color negro. Se puede colorear todo píxel no perteneciente al conjunto de Mandelbrot de blanco, pero también es posible obtener más información (y una imágen más bonita) si se usan distintos tonos en los puntos que no pertenecen al conjunto de Mandelbrot. En particular, si usamos distintos tonos de grises para indicar el $n < N$ a partir del cual $(x_n^2 + y_n^2) > 4$, se comienzan a ver más detalles. Típicamente, en vez de tonos de grises se utiliza una paleta de colores para realizar el dibujo. La más común (y que utilizaremos en este TP) es la basada en el modelo de color HSV. Se puede

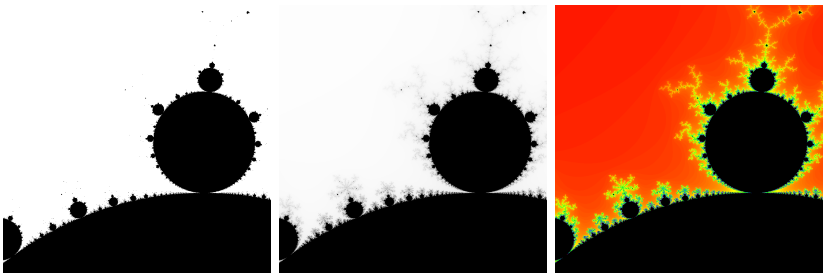

Figura 2: Gráfico del conjunto de Mandelbrot en la región $(-0{,}5; 0) \times (0{,}5; 1)$. A la izquierda, en blanco y negro, en el centro en escala de grises y a la derecha con coloreado utilizando el modelo de color HSV.

ver un ejemplo de cada una de estas tres alternativas en la Figura 2. También se puede explorar un gráfico interactivo en `https://sciencedemos.org.uk/mandelbrot.php`.

Una de las propiedades interesantes del conjunto de Mandelbrot es que si se grafican regiones muy pequeñas de plano complejo, es decir, "haciendo zoom" sobre el conjunto original, se pueden encontrar diversas figuras que incluso tienen formas similares a la del gráfico del conjunto original, como se puede ver por ejemplo en la Figura 3.

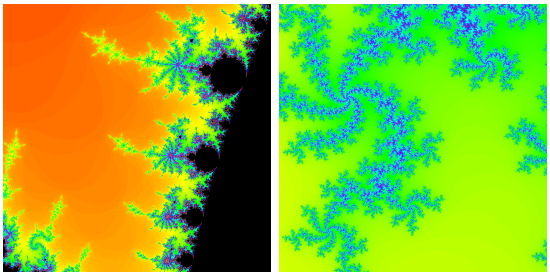

Figura 3: Ejemplos de figuras que surgen graficando el conjunto de Mandelbrot en escalas pequeñas. El gráfico de la izquierda coresponde a la región $(-1{,}28; -1{,}23) \times (0{,}1; 0{,}05)$, y el de la derecha a la región $(-0{,}55; -0{,}5498) \times (-0{,}55; -0{,}5498)$.

El nivel de detalle obtenido al graficar el conjunto de Mandelbrot depende de la cantidad de iteraciones, como se puede ver en la Figura 4. A mayor grado de acercamiento, mayor es la cantidad de iteraciones que hay que realizar para conseguir detalle.

Figura 4: Distintas cantidades de iteraciones para le región $(-0{,}5; 0) \times (0{,}5; 1)$. A la izquierda, 10, en el centro 20 y a la derecha 50.

La tarea de calcular el color que corresponderá a cada píxel de la imágen es independiente de las demás y por lo tanto nos interesará realizar estos cálculos de manera

concurrente.

Objetivo del Trabajo Práctico

Se pide implementar un programa en Java que genere la imágen correspondiente al conjunto de Mandelbrot para una región dada por parámetro, donde la cantidad de iteraciones y cantidad de *threads* a usar sea configurable. Se pedirá tomar los tiempos que lleva generar una misma imágen con distintas configuraciones para determinar el impacto de utilizar la concurrencia para generarla.

Elementos a entregar

Programa Java

Se debe presentar un programa escrito en Java que tome los siguientes parámetros: `alto`, `ancho`, `x_inicial`, `y_inicial`, `x_rango`, `y_rango`, `cant_iteraciones`, `cant_threads`, y `tamano_buffer`. El programa deberá tener la siguiente estructura:

1. Una clase `Main` con el punto de entrada del programa, que tiene la responsabilidad de inicializar las estructuras mencionadas a continuación y producir unidades de trabajo consistentes en subregiones de la región a graficar para que sean generadas por múltiples thredads. El programa debe terminar únicamente cuando se haya generado la imágen de salida, y debe informar el tiempo transcurrido desde el inicio de la ejecución.

2. Una clase `Buffer` (implementada como un monitor utilizando métodos synchronized) que actúa como una cola FIFO concurrente de capacidad acotada. Es decir, bloquea a un lector intentando sacar un elemento cuando está vacía y bloquea a un productor intentando agregar un elemento cuando está llena. La capacidad del Buffer también debe ser un parámetro configurable desde la clase `Main`.

3. Una clase `WorkerCounter` (implementada como un monitor utilizando métodos synchronized) que evita que `Main` termine su ejecución mientras queden threads trabajando.

4. Una clase `Worker` que extienda de `Thread` y que tome tareas (clase `Task`) de un buffer hasta tomar una señal de terminación (*Poison Pill*).

5. Una clase `Task` que implementa `Runnable`, con una subclase `MandelbrotTask` que hace el gráfico del conjunto de Mandelbrot para una región dada. El método `run` de cada `MandelbrotTask` toma los píxeles de la región y evalúa el color que deben tener. Se recomienda usar regiones que sean tiras de píxeles consecutivos de la misma fila (por ejemplo, los primeros 100, luego los segundos 100, etc. hasta terminar la fila).

6. Una clase `PoisonPill` que hereda de `Task` y que hace que el `Worker` que la tome termine su ejecución (puede ser por medio de una excepción, como en la práctica).

7. Una clase `ThreadPool`, que se encarga de instanciar e iniciar la cantidad de `Workers` pedida por un usuario.

8. Cualquier otra clase que considere necesaria.

Al iniciar el programa la clase `Main` debe delegar la iniciación de los threads necesarios en la clase `ThreadPool` y luego introducir de a uno las regiones a procesar en el `Buffer`. Cada `Worker` en funcionamiento debe tomar las tareas de a una del `Buffer` y generar los píxeles que le correspondan. Cabe destacar que es inadmisible utilizar una cantidad de threads menor a la solicitada por el usuario.

El programa, además, debe estar correctamente documentado. En particular, debe contener las instrucciones para configurar los parámetros, y el nombre del archivo de salida. Idealmente hacer que el programa se pueda ejecutar por línea de comandos y tome estas opciones por parámetro o por entrada estándar.

No es incorrecto que el programa haga impresiones por pantalla intermedias para informar su progreso, pero se pide evitar la impresión de mensajes de *debug*.

Informe

Además del código, se requiere un informe corto en formato `pdf`, donde se deben mostrar los resultados de la aplicación del programa sobre una región a elección (que debe estar documentada). Se debe probar generar imágenes de tres tamaños: *pequeño* (cercano a 64×64), *mediano* (cercano a 512×512) y *grande* (cercano a 1024×1024).

Para cada uno de los tamaños, registrar el tiempo de ejecución del programa utilizando distintas combinaciones de cantidades de threads y cantidades de iteraciones.

- **Threads:** 1, 2, 4, 8 y 16
- **Cant. de iteraciones:** 10, 100, 500, 1000

El informe, que es **condición necesaria para la aprobación del TP**, debe respetar el siguiente formato:

1. **Autoría:** Nombres, e-mail y número de legajo de quienes hicieron el trabajo (máximo 3 personas salvo excepciones acordadas con los docentes).
2. **Introducción:** Sección explicando el dominio del TP, el diseño del código y cualquier consideración adicional que considere relevante para la correcta interpretación de los resultados.
3. **Evaluación:** Sección explicando el proceso de evaluación, incluyendo los detalles del hardware donde se ejecutan las pruebas (modelo de microprocesador con cantidad de núcleos, memoria disponible y versión del sistema operativo). En esta sección deben incluirse tres tablas reportando los tiempos en los que se generaron las imágenes para cada combinación de cantidad de threads y cantidad de iteraciones, y dos gráficos: El primero deberá ser para cantidad de iteraciones 500, que en el eje x tenga las distintas cantidades de threads, y en el y el tiempo demorado, con una curva por imagen procesada. El segundo, deberá ser para la cantidad de 8 threads, y tener en el eje x las distintas cantidades de iteraciones (una vez más, en el eje y estarán los tiempos de demora y deberá haber una curva por imagen). Si para una o varias de estas combinaciones el tiempo de ejecución es muy pequeño y produce mucha variabilidad, tomar un promedio de 10 ejecuciones. Esto debe estar documentado.
4. **Análisis:** Sección en la que se debe discutir los datos obtenidos en la evaluación, en particular destacando la combinación de parámetros con la que se obtuvo el tiempo de ejecución mínimo y la combinación que obtuvo el tiempo máximo, más alguna conclusión (Por ejemplo, responder a ¿Es verdad que aumentar la cantidad de threads siempre mejora el rendimiento? ¿Cuánto influye la cantidad de iteraciones? ¿Los valores de cantidad de threads y cantidad de iteraciones óptimos cambian con el tamaño de la imágen?)

Forma de Entrega

Se deben enviar el archivo `.zip` con el código del programa y el `.pdf` con el informe por email a las casillas de correo electrónico de *ambos* profesores con el subject:
`Entrega TP PCONC 2S2023 - (apellido1) - (apellido2) - (apellido3)`
(donde `apellido1`, `apellido2` y `apellido3` son los apellidos de las personas que constituyen el grupo). El mensaje debe ir *con copia* a las otras persona que integre el grupo, pues la devolución del TP se hará en *respuesta a todos* ese correo.

Es posible trabajar en un repositorio *git* compartido por las personas que compongan el grupo. En este caso, el repositorio debe ser **privado**. Al entregar, se deberá agregar a los profesores al repositorio. De todos modos se solicita que envíen el mail con las condiciones especificadas, aunque en vez de tener los archivos adjuntos incluirán el link al respositorio. Se aclara que se espera que el código entregado por el grupo sea **original**. Si se detecta que el mismo fue copiado de alguna fuente online será automáticamente desaprobado sin posibilidad de reentrega.

Fecha de Entrega

El TP debe entregarse antes del **Martes 07 de Noviembre** a las **23:59hs**. En caso de ser necesario, se solicitará una reentrega el **Martes 05 de Diciembre**.

Apéndice: Algunas herramientas de Java

Manejo de números reales

Para operar con números reales se recomienda usar el tipo de datos `double`.

Manipulación de imágenes

Para la manipulación de imágenes píxel a píxel que se require en este TP, se recomienda utilizar las clases `BufferedImage` y `WritableRaster`. En particular, el siguiente código:

```
BufferedImage bi = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
WritableRaster raster = bi.getRaster();
```

Permite obtener un objeto de la clase `WritableRaster`con altura `h` y ancho `w`, al cual se le puede setear el color de un pixel utilizando el método `setPixel(x, y,colores)`, siendo `colores` un array de 3 enteros, representando el color del píxel en sus componentes roja, verde y azul (con valores de 0 a 255).

Para generar el archivo de salida, se puede utilizar la clase `ImageIO`. Se puede hacer:

```
File outputfile = new File("salida.png");
try {
        ImageIO.write(bi, "png", outputfile);
} catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

Medición de tiempos

Para medir el tiempo de ejecución del programa, se puede llamar al método provisto por Java `System.currentTimeMillis()`. Se debe guardar el tiempo actual cuando se va a

comenzar a operar (si se toman parámetros por entrada estándar, debe ser después), y cuando se terminó la ejecución (es decir, luego de que los Workers hayan parado). Luego, basta con hacer la diferencia entre ambos tiempos y dividirla por mil para obtener el tiempo demorado expresado en segundos.

Paleta de Colores HSV

Para generar la paleta de colores a asignar a los píxeles de la imágen, una posibilidad es crear tres arrays de longitud `cant_iteraciones`. Cada uno de ellos contiene el valor en el canal rojo, verde y azul que debe tener un pixel según la iteración n en la que el pixel cumple que $(x_n^2 + y_n^2) > 4$. Para hacer la transformación desde el espacio de color HSV, se debe realizar la siguiente conversión:

```
r = new int[cant_iteraciones+1];
g = new int[cant_iteraciones+1];
b = new int[cant_iteraciones+1];

this.r[cant_iteraciones]=0;
this.g[cant_iteraciones]=0;
this.b[cant_iteraciones]=0;

for (int i=0;i<cant_iteraciones;i++) {
  int argb = Color.HSBtoRGB(i/256f, 1, 1);
  r[i] = argb >> 16;
  g[i] = (argb >> 8) & 255;
  b[i] = argb & 255;
}
```

Así, si por ejemplo $n = 7$, pintaríamos el pixel con los valores `r[7]`, `g[7]` y `b[7]`.