Actividad de programación Multiprocesadores:

Procesando imágenes con el Fork-Join framework

José Ramón Romero Chávez (A01700318)
Tecnológico de Monterrey, Campus Querétaro
A01700318@itesm.mx

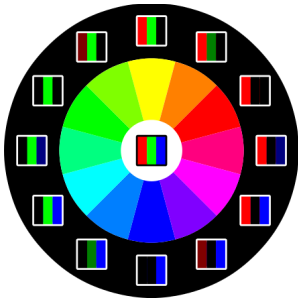
23 de septiembre de 2018

Resumen

Este documento se explicara de manera puntual la implementación del algoritmo de transformación de una imagen de color a escala de grises, haciendo uso de del framework Fork/Join, el cual nos permite sacar ventaja de sistemas multiprocesador, adicional a esto se realizará el mismo procedimiento en modo monoprocesador con el fin de analizar las diferencias y comparar su rendimiento.

1. Introducción

El modelo RGB se basa en lo que se conoce como síntesis aditiva de color. Empleando la luminosidad del rojo, el verde y el azul en diferentes proporciones, se produce el resto de los colores. Los monitores de las computadoras(ordenadores) apelan a la síntesis aditiva de color para la representación de los colores.



Varios lenguajes de programación usan el modelo RGB para representar los colores. Estos lenguajes asignan un valor a los distintos colores: a mayor valor, mayor intensidad en la mezcla. Un color, por lo tanto, se compone de tres valores (uno correspondiente al rojo, otro al verde y el tercero al azul) .

Adicional al los canales RGB, encontramos el canal alpha, el cual nos servirá cuando hablemos de imágenes (.png).

ALPHA								RED								GREEN								BLUE							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

La escala de grises es una escala empleada en la imagen digital en la que el valor de cada píxel posee un valor equivalente a una graduación de gris. Las imágenes representadas de este tipo están compuestas de sombras de grises.

Color		GrayScale
Red = X		Red =Average (X,Y,Z)
Green = Y	-> Average (X,Y,Z) ->	Blue = Average (X,Y,Z)
Blue = Z		Green =Average (X,Y,Z)

Para obtener dicha graduación basta con obtener el promedio de los valores de cada uno de los canales y asignar valor resultante a cada canal

Ahora bien, dado este algoritmo, es posible realizarse de manera MonoProcesador o MultiProcesador, para ello haremos uso del framework de Fork Join de Java para posteriormente explicar y comparar ambas implementaciones con fin de exponer los puntos importantes de cada uno con respecto a su contraparte.

2. Implementación

La implementación fue realizada en una computadora con las siguientes características:

- MacBook Pro (Retina, 13-inch, Early 2015)
  - macOS High Sierra 10.13.6 (17G65)
  - Processor Name: Intel Core i5
  - Processor Speed: 2.7 GHz
  - Number of Processors: 1
  - Total Number of Cores: 2 with hyperthreading
  - L2 Cache (per Core): 256 KB
  - L3 Cache: 3 MB
  - Memory: 8 GB 1867 MHz DDR3
  - javac -version: javac 9
  - java -version:
    - java version "9"
    - Java(TM) SE Runtime Environment (build 9+181)
    - Java HotSpot(TM) 64-Bit Server VM (build 9+181, mixed mode)

Clases compartidas

Con el fin de simplificar el procedimiento de creacion y ejecucion de algunas tareas se crearon algunas clases compartidas.

- *Utils.java (Variables estaticas y de control)*
- *ImageFrame.java (Exposicion de Imagen)*

Utils.java (Variables estaticas y de control);

```
import java.util.Random;
public class Utils {
     private static final int DISPLAY = 100;
     private static final int MAX_VALUE = 10_000;
     private static final Random r = new Random();
     public static final int N = 10;
     public static void randomArray(int array[]) {
          for (int i = 0; i < array.length; i++) {
               array[i] = r.nextInt(MAX_VALUE) + 1;
          }
     }
     public static void fillArray(int array[]) {
          for (int i = 0; i < array.length; i++) {
               array[i] = (i % MAX_VALUE) + 1;
          }
     }
     public static void displayArray(String text, int
array[]) {
          System.out.printf("%s = [%4d", text, array[0]);
          for (int i = 1; i < DISPLAY; i++) {
               System.out.printf(",%4d", array[i]);
          }
          System.out.printf(", ..., ]\n");
     }
}
```

ImageFrame.java (Exposicion de Imagen)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ImageFrame {

     public static void showImage(String text, Image image) {
        JFrame frame = new JFrame(text);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel label = new JLabel(new ImageIcon(image));
                              frame.getContentPane().add(label,
BorderLayout.CENTER);

        frame.pack();
        frame.setVisible(true);
    }
}
```

Versión Mono-Procesador

En esta primera implementación vemos todo el cálculo implementado en una sola clase SingleMain que ejecuta una instancia de SingleGray.

Es importante recalcar que la parte fundamental, es el ciclo iterativo que recorre toda la imagen de una sola vez.

MainSingle.java

```
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;


public class MainSingle {
     public static void main(String args[]) throws Exception
{
          long startTime, stopTime;
          double acum = 0;


          if (args.length != 1) {
               System.out.println("usage:   java   MainSingle
image_file");
               System.exit(-1);
          }


          final String fileName = args[0];
          File srcFile = new File(fileName);
        final BufferedImage source = ImageIO.read(srcFile);


          int w = source.getWidth();
          int h = source.getHeight();
          int src[] = source.getRGB(0, 0, w, h, null, 0, w);
```

```
            int dest[] = new int[src.length];

            SingleGray e = new SingleGray(src, dest, w, h);
            acum = 0;
            for (int i = 0; i < Utils.N; i++) {
                 startTime = System.currentTimeMillis();
                 e.doMagic();
                 stopTime = System.currentTimeMillis();
                 acum += (stopTime - startTime);
            }
            System.out.printf("avg  time  =  %.5f\n",  (acum  /
Utils.N));
            final    BufferedImage    destination    =    new
BufferedImage(w, h, BufferedImage.TYPE_INT_ARGB);
            destination.setRGB(0, 0, w, h, dest, 0, w);

            javax.swing.SwingUtilities.invokeLater(new
Runnable() {
              public void run() {
                  ImageFrame.showImage("Original - " + fileName,
source);
              }
          });

            javax.swing.SwingUtilities.invokeLater(new
Runnable() {
              public void run() {
                     ImageFrame.showImage("Gray  -  " + fileName,
destination);
```

```
            }
        });
    }
}
```

SingleGray.java

```
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;

public class SingleGray {
    private static final int BLUR_WINDOW = 15;
    private int src[], dest[], width, height;

    public SingleGray(int src[], int dest[], int width, int
height) {
        this.src = src;
        this.dest = dest;
        this.width = width;
        this.height = height;
    }

    private void grax_pixel(int ren, int col) {

        int side_pixels, i, j, cells;
        int tmp_ren, tmp_col, pixel, dpixel;
        float r, g, b, avg;
        r = 0; g = 0; b = 0;
        pixel = src[(ren * width) + col];
```

```
                    r = (float) ((pixel & 0x00ff0000) >> 16);
                    g = (float) ((pixel & 0x0000ff00) >> 8);
                    b = (float) ((pixel & 0x000000ff) >> 0);
                    avg=(r+g+b)/3;

          dpixel = (0xff000000)
                    | (((int) (avg)) << 16)
                    | (((int) (avg)) << 8)
                    | (((int) (avg)) << 0);
          dest[(ren * width) + col] = dpixel;
     }

     void doMagic() {
          int index, size;
          int ren, col;
          size = width * height;
          for (index = 0; index < size; index++) {
               ren = index / width;
               col = index % width;
               grax_pixel(ren, col);
          }
     }
}
```

Versión Multiprocesador

En la versión multi-thread existen los siguientes puntos claves dentro de la implementación:

El algoritmo sigue siendo el mismo, sin embargo, es posible implementar la técnica de divide y conquista dentro del arreglo que tiene los valores de colores, ya que al encontrarse con un segmento muy grande se decide fraccionar en partes más pequeñas de manera recursiva.

Similar a la versión Single, consta de un main, que crea y ejecuta una instancia de ForkJoinPool.

MainMulti.java

```
import java.awt.image.BufferedImage;

import java.io.File;

import javax.imageio.ImageIO;

import java.util.concurrent.ForkJoinPool;


public class MainMulti {

     private     static     final     int     MAXTHREADS     =
Runtime.getRuntime().availableProcessors();


     public static void main(String args[]) throws Exception
{

          ForkJoinPool pool;

          long startTime, stopTime;

          double acum = 0;

          if (args.length != 1) {

               System.out.println("usage:    java    MainMulti
image_file");

               System.exit(-1);

          }

          final String fileName = args[0];
```

```
          File srcFile = new File(fileName);
        final BufferedImage source = ImageIO.read(srcFile);

          int w = source.getWidth();
          int h = source.getHeight();
          int src[] = source.getRGB(0, 0, w, h, null, 0, w);
          int dest[] = new int[src.length];

          acum = 0;
          for (int j = 1; j <= Utils.N; j++) {
               startTime = System.currentTimeMillis();

               pool = new ForkJoinPool(MAXTHREADS);
               pool.invoke(new ForkGray(src, dest, w, h, 0,
src.length));

               stopTime = System.currentTimeMillis();
               acum +=  (stopTime - startTime);
          }
          System.out.printf("avg  time  =  %.5f\n",  (acum  /
Utils.N));

          final    BufferedImage    destination    =    new
BufferedImage(w, h, BufferedImage.TYPE_INT_ARGB);
          destination.setRGB(0, 0, w, h, dest, 0, w);
```

```
          javax.swing.SwingUtilities.invokeLater(new
Runnable() {
            public void run() {
                ImageFrame.showImage("Original - " + fileName,
source);
            }
        });

          javax.swing.SwingUtilities.invokeLater(new
Runnable() {
            public void run() {
                  ImageFrame.showImage("Gray - " + fileName,
destination);
            }
        });
     }
}
```

ForkGray.java

```
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;
import java.util.concurrent.RecursiveAction;

public class ForkGray extends RecursiveAction {
     private static final int BLUR_WINDOW = 15;
     private static final long MIN = 10_000;
     private int src[], dest[], width, height, start, end;
```

```
    public ForkGray(int src[], int dest[], int width, int
height, int start, int end) {
          this.src = src;
          this.dest = dest;
          this.width = width;
          this.height = height;
          this.start = start;
          this.end = end;
    }

    private void gray_pixel(int ren, int col) {
          int side_pixels, i, j, cells;
          int tmp_ren, tmp_col, pixel, dpixel;
          float r, g, b, avg;

          r = 0; g = 0; b = 0;
          pixel = src[(ren * width) + col];

                    r = (float) ((pixel & 0x00ff0000) >> 16);
                    g = (float) ((pixel & 0x0000ff00) >> 8);
                    b = (float) ((pixel & 0x000000ff) >> 0);
                    avg=(r+g+b)/3;
          dpixel = (0xff000000)
                    | (((int) (avg)) << 16)
                    | (((int) (avg)) << 8)
                    | (((int) (avg)) << 0);
          dest[(ren * width) + col] = dpixel;
    }
```

```
    public void computeDirectly() {
        int index, size;
        int ren, col;
        size = width * height;
        for (index = start; index < end; index++) {
            ren = index / width;
            col = index % width;
            gray_pixel(ren, col);
        }
    }

    @Override
    protected void compute() {
        if ( (this.end - this.start) <= ForkGray.MIN ) {
            computeDirectly();

        } else {
            int middle = (end + start) / 2;

            invokeAll(new ForkGray(src, dest, width,
height, start, middle),
                      new ForkGray(src, dest, width,
height, middle, end));
        }
    }
}
```

3. Resultados.

Nuestro caso inicial está basado en el procesamiento de 2 imágenes (anexo seccion 4), 10 veces, ejecutando ambas versiones.

Resultados:

MacBook-Pro-de-Ramon-2:forkJoin ramonromero$ java MainMulti imagen1.jpg

avg time = 102.00000

MacBook-Pro-de-Ramon-2:forkJoin ramonromero$ java MainMulti imagen2.jpg

avg time = 88.80000

MacBook-Pro-de-Ramon-2:secuential ramonromero$ java MainSingle imagen1.jpg

avg time = 169.30000

MacBook-Pro-de-Ramon-2:secuential ramonromero$ java MainSingle imagen2.jpg

avg time = 216.60000

La evaluación promedio del caso imagen1 arrojó el siguiente speedup:

$S = \frac{T_s}{T_m}$ $S = \frac{169.3\ ms}{102\ ms}$ $S = 1,65$

La evaluación promedio del caso imagen1 arrojó el siguiente speedup:

$S = \frac{T_s}{T_m}$ $S = \frac{216.6ms}{88.8\ ms}$ $S = 2.43$

Es evidente que el uso de Fork Join mejora mucho el procesamiento de la imagen, ya que fracciona de manera puntual el trabajo para cada procesador, adicional a esto es necesario puntualizar que la dimensión de la imagen juega un factor fundamental en el procesamiento.

3. Referencias

[1]"Pi", En.wikipedia.org, 2018. [Online]. Available: https://en.wikipedia.org/wiki/RGB. [Accessed: 17- Sep- 2018].

[2] [BRESHEARS] pp. 31 & 32.

[3]"Amdahl's law", En.wikipedia.org, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Amdahl%27s_law. [Accessed: 15- Sep- 2018].

[4]"Wolfram|Alpha", M.wolframalpha.com, 2018. [Online]. Available: http://m.wolframalpha.com/. [Accessed: 15- Sep- 2018].

[5]"JDK 10 Documentation", Docs.oracle.com, 2018. [Online]. Available: https://docs.oracle.com/javase/10/. [Accessed: 15- Sep- 2018].

4 Anexo - Imágenes