Brian Rodriguez Badillo ICOM 4036-096 Prof. Wilson Rivera

**Take Home #2 Report**

This simulation of a server application that applies serial and concurrent implementations of a median filter to an arbitrary image consists of 4 parts. The code begins by defining some vals that will correspond to the input filepath string, importing the image designated by the filepath, and the appropriate Future instantiations.

Next, we apply separate for-comprehensions to independently run both serial and concurrent implementations of the median filter concurrently/side-by-side. These run in parallel asynchronously and print to the console the time each implementation took to finish since the program started. They print just after each process writes the image within the appropriate directory (in the src folder). The serial implementation generates a serial.jpg image, while the concurrent implementation generates a concurrent.jpg file.

The median filter implemented takes in a BufferedImage and then applies a standard median filtering technique to the image. It iterates over most of the image pixels (excluding the right-most and bottom 2-pixel lines) and calculates the median RGB value of the surrounding pixels defined by a 3x3 kernel. Essentially, it collects the RGB values of the neighboring pixels, sorts them, and selects the median or “middle” value of the bunch, then sets the corresponding pixel on the output image to that value. The method then returns the new image contained in a Future container. This is used as the serial implementation, by passing it the whole image to be filtered and returning the whole image filtered.

The concurrent implementation takes the input image and splits it horizontally into 4 equal sized parts (always leaving the bottom part as the biggest for images with an odd number of vertical rows). Each one of these quarters gets filtered by the base/serial implementation of the median filter, but all quarters get filtered concurrently. Once all quarters are properly filtered, they are put together by being drawn onto an output BufferedImage’s Graphics object. This is then returned inside a Future container.

Consistently, the concurrent implementation finishes between 300 and 700 units of time faster than the serial implementation. For testing purposes, each run of the standard median Filter method prints out the time it took to run the filter. Assuming that the 4 runs with the lowest and most similar runtimes (between 200 and 350 for the test image) belong to the 4 concurrent filtering, the concurrent filtering takes approximately 5 times less than the serial filtering (which takes about 1000 units for the test image). Presumably, the overall time for the concurrent processing is bottlenecked by yielding for all quarters to be filtered to then serially write these to the output image, which reduces the gap between the serial and concurrent implementations. Still the concurrent implementation finishes consistently faster than the serial implementation.

These values are heavily dependent on the image being filtered and the computer filtering them, but the trend is not; concurrently filtering an image is more efficient and therefore faster than a serial filtering, and this gap increases as the size of the image being filtered increases.