**COMP553-18B (HAM) - Extremely Parallel Programming**

**OpenCL Assignment: Turning Photos into Cartoons**

# Step 1: Run initial program, trying to find out bottleneck of performance

$ java -jar -Xprof bin/artifacts/cartoonify\_jar/cartoonify\_01.jar -e 256 -c 3 img\_bucket.jpg img\_bumblebee.jpg img\_dog.jpg img\_pavlova.jpg img\_shenzhen.jpg img\_sunflower.jpg img\_surfers.jpg

Using edge threshold 256

Using 3 discrete colours per channel.

Done img\_bucket.jpg -> img\_bucket\_cartoon.jpg in 3.266 secs.

Done img\_bumblebee.jpg -> img\_bumblebee\_cartoon.jpg in 3.203 secs.

Done img\_dog.jpg -> img\_dog\_cartoon.jpg in 3.251 secs.

Done img\_pavlova.jpg -> img\_pavlova\_cartoon.jpg in 3.154 secs.

Done img\_shenzhen.jpg -> img\_shenzhen\_cartoon.jpg in 3.043 secs.

Done img\_sunflower.jpg -> img\_sunflower\_cartoon.jpg in 3.125 secs.

Done img\_surfers.jpg -> img\_surfers\_cartoon.jpg in 1.329 secs.

Average processing time is **2.910** for 7 photos.

Flat profile of 27.76 secs (1538 total ticks): main

Compiled + native Method

31.4% 482 + 1 com.celanim.cartoonify.Cartoonify.gaussianBlur

30.8% 473 + 1 com.celanim.cartoonify.Cartoonify.sobelEdgeDetect

7.4% 114 + 0 com.celanim.cartoonify.Cartoonify.convolution

5.5% 83 + 1 java.awt.image.BufferedImage.getRGB

3.1% 48 + 0 java.awt.image.BufferedImage.setRGB

1.2% 17 + 1 com.celanim.cartoonify.Cartoonify.reduceColours

0.4% 6 + 0 java.awt.image.WritableRaster.setRect

0.2% 3 + 0 sun.awt.image.ByteInterleavedRaster.putByteData

0.2% 2 + 1 com.celanim.cartoonify.Cartoonify.mergeMask

0.1% 0 + 2 com.celanim.cartoonify.Cartoonify.loadPhoto

0.1% 1 + 0 java.awt.image.Raster.getPixels

0.1% 1 + 0 java.awt.image.SinglePixelPackedSampleModel.createSubsetSampleModel

0.1% 1 + 0 sun.awt.image.ByteInterleavedRaster.setPixels

0.1% 1 + 0 UncommonTrapBlob

0.1% 0 + 1 java.awt.image.DirectColorModel.getDataElements

80.6% 1232 + 8 Total compiled

From the profiling output information, gaussianBlur() and sobelEdgeDetect() are the bottlenecks of this program. These two method need to be implemented in OpenCL GPU .

# Step 2: Unroll for-loops in convolution()

*$ java -jar -Xprof bin/artifacts/cartoonify\_jar/cartoonify.jar -e 256 -c 3 img\_bucket.jpg img\_bumblebee.jpg img\_dog.jpg img\_pavlova.jpg img\_shenzhen.jpg img\_sunflower.jpg img\_surfers.jpg*

Using edge threshold 256

Using 3 discrete colours per channel.

Done img\_bucket.jpg -> img\_bucket\_cartoon.jpg in 2.415 secs.

Done img\_bumblebee.jpg -> img\_bumblebee\_cartoon.jpg in 2.417 secs.

Done img\_dog.jpg -> img\_dog\_cartoon.jpg in 2.378 secs.

Done img\_pavlova.jpg -> img\_pavlova\_cartoon.jpg in 2.328 secs.

Done img\_shenzhen.jpg -> img\_shenzhen\_cartoon.jpg in 2.275 secs.

Done img\_sunflower.jpg -> img\_sunflower\_cartoon.jpg in 2.301 secs.

Done img\_surfers.jpg -> img\_surfers\_cartoon.jpg in 0.911 secs.

Average processing time is **2.146** for 7 photos.

Flat profile of 22.14 secs (1794 total ticks): main

Compiled + native Method

28.5% 511 + 1 com.celanim.cartoonify.Cartoonify.sobelEdgeDetect

27.5% 492 + 1 com.celanim.cartoonify.Cartoonify.gaussianBlur

7.5% 133 + 1 com.celanim.cartoonify.Cartoonify.convolution

6.6% 118 + 1 java.awt.image.BufferedImage.getRGB

3.7% 66 + 0 java.awt.image.BufferedImage.setRGB

1.6% 27 + 2 com.celanim.cartoonify.Cartoonify.reduceColours

0.3% 4 + 2 com.celanim.cartoonify.Cartoonify.mergeMask

0.3% 5 + 0 java.awt.image.WritableRaster.setRect

0.1% 0 + 2 com.celanim.cartoonify.Cartoonify.loadPhoto

0.1% 0 + 1 java.awt.image.DirectColorModel.getDataElements

76.2% 1356 + 11 Total compiled

By unrolling for-loops in convolution() method, the average processing time improved from 2.91s to 2.15s, the performance improved by 26%.

# Step 3: Alternative wrap() method

The performance didn’t change significantly by modifying the warp() method. The alternative warp() method made it easier to implement corresponding OpenCL kernel method, in that to keep GPU and CPU version generate the same results (same md5sum).

# Step 4: Convert bottleneck methods to kernel

\_\_kernel void gaussianBlur() and \_\_kernel void sobelEdgeDetect() were create. 1D range was used to run the kernel for the sake of simplicity. int3 vector was used to hold rgb in private memory, rather than 3 separate int variables.

# Step 5: Send a sequence of commands to GPU synchronized with queue

*$ java -jar -Xprof bin/artifacts/cartoonify\_jar/cartoonify.jar -g -e 256 -c 3 img\_bucket.jpg img\_bumblebee.jpg img\_dog.jpg img\_pavlova.jpg img\_shenzhen.jpg img\_sunflower.jpg img\_surfers.jpg*

Using edge threshold 256

Using 3 discrete colours per channel.

Done img\_bucket.jpg -> img\_bucket\_cartoon.jpg in 0.586 secs.

Done img\_bumblebee.jpg -> img\_bumblebee\_cartoon.jpg in 0.435 secs.

Done img\_dog.jpg -> img\_dog\_cartoon.jpg in 0.422 secs.

Done img\_pavlova.jpg -> img\_pavlova\_cartoon.jpg in 0.432 secs.

Done img\_shenzhen.jpg -> img\_shenzhen\_cartoon.jpg in 0.416 secs.

Done img\_sunflower.jpg -> img\_sunflower\_cartoon.jpg in 0.401 secs.

Done img\_surfers.jpg -> img\_surfers\_cartoon.jpg in 0.161 secs.

Average processing time is **0.407** for 7 photos.

Flat profile of 10.06 secs (771 total ticks): main

Compiled + native Method

15.0% 115 + 1 java.awt.image.BufferedImage.getRGB

8.8% 67 + 1 java.awt.image.BufferedImage.setRGB

4.0% 27 + 4 com.celanim.cartoonify.Cartoonify.reduceColours

0.4% 3 + 0 java.awt.image.WritableRaster.setRect

0.4% 2 + 1 com.celanim.cartoonify.Cartoonify.mergeMask

0.1% 0 + 1 com.celanim.cartoonify.Cartoonify.loadPhoto

28.8% 214 + 8 Total compiled

The average processing time with OpenCL GPU improved from 2.15s to 0.41s and increased 7.1 times comparing to the original code on my machine. The bottleneck now became image I/O as the profiling information shows above.

Difficulties and improvements:

* Debugging OpenCL kernel code.
* Utilization of local/private memory.
* Building up appropriate execution model.
* Need to come up with better warp() method to improve kernel performance by reduce unnecessary branchings.