Programming Assignment 1: Shared Memory Programming

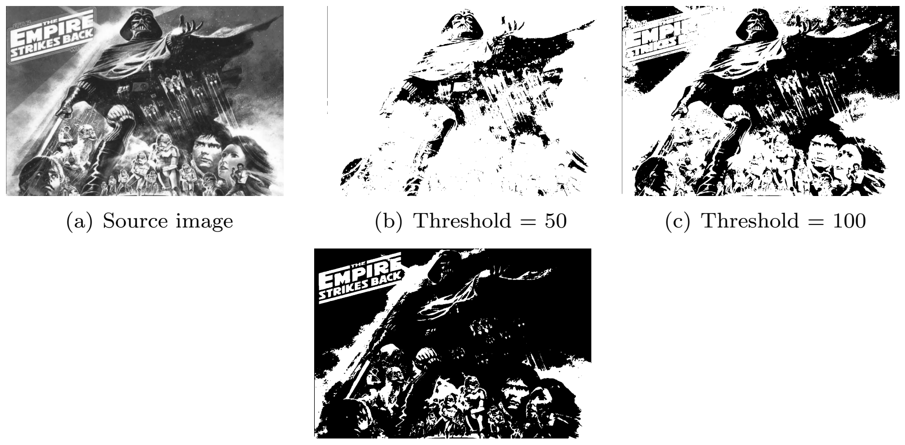
ECSE 420 – Parallel Computing – Fall 2015

Released: September 23, 2015
Due: September 30, 2015 at 11:59 pm

1 An "Embarrassingly Parallel" Task

In this programming assignment, you will parallelize some common greyscale image processing algorithms using multithreading.

The first algorithm you will accelerate is binarization. Binarizing an image consists in setting all pixels above a certain integer threshold value to 255 and all pixels below the threshold to 0. This is an "embarrassingly parallel" algorithm (i.e., a task which is very easy to parallelize because every operand can be assigned to a single subtask and no communication between subtasks is necessary). The result of performing binarization on a test image for a few threshold values is shown in Figure 1:



(a) Source image (b) Threshold = 50 (c) Threshold = 100 (d) Threshold = 200

Figure 1: A source image and binarized versions of that image. Note how a threshold of 50 extracts the main figure fairly well. Binarization is sometimes used to preprocess images before more sophisticated kernels are applied.

Fill in the TODOs in binarize_sequential.c, binarize_pthreads.c, and binarize_openmp.c. A Makefile is included- you can use “make” to compile everything, and “make clean” to delete all object files and linked executables. The executables have the same names as the source files (without the .c extension). For the sequential executable, the usage is “./< executable name> <input PNG name> <output PNG name>”. For the multithreaded executables, the usage is “./<executable name> <input PNG name> <output PNG name> <number of threads>”.

Run the sequential version, then run both parallel programs using 2, 4, 8, 16, and 32 threads. Repeat this trial 100 times to get 100 + 500 + 500 = 1100 runtimes. Calculate the average speedup over the sequential version for each thread count (10 averages).

Plot the average speedup for the Pthreads version and the OpenMP version for each thread count. (Put all 10 data points on your graph so that it’s easy to compare each version visually for each thread count.)

Is the speedup greater than 1 for all five thread counts? Does the speedup peak at a certain thread count? Explain your speedup graph with reference to your computer’s architecture.

Report the size of each executable in bytes and the length (number of lines of code) of each source file. Which executable is smaller, the Phreads executable or the OpenMP executable? Come up with a hypothesis as to why.

Finally, include your test PNG picture and the binarized version. Bonus points if your picture amuses the grader!

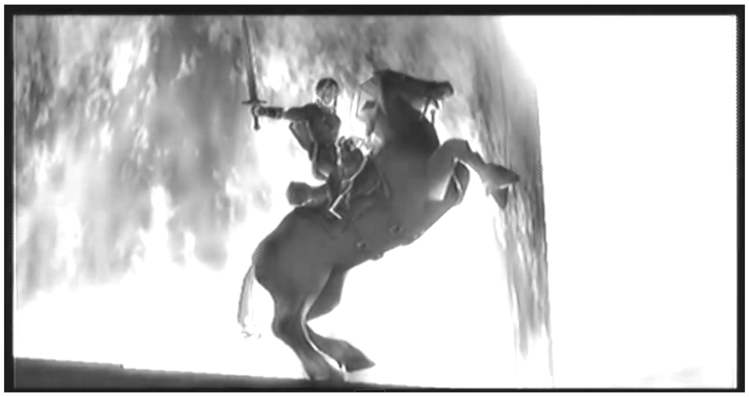
2 A More Challenging Parallelization Problem

The Sobel filter is defined as follows:

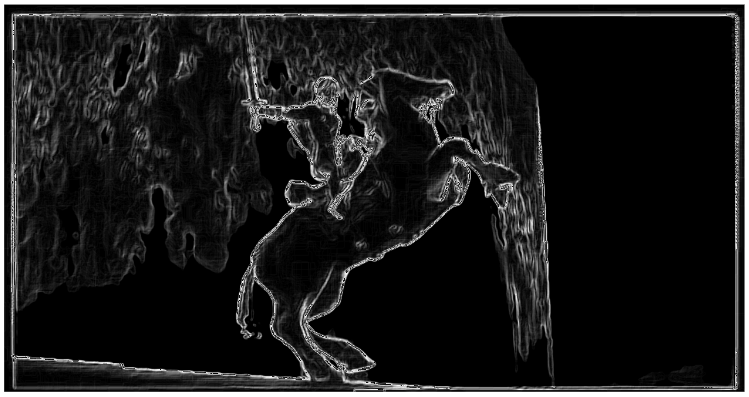
```
0 < i < image height - 1, 0 < j < image width - 1:

output(i,j) = |(input(i-1,j-1) + 2*input(i-1,j) + input(i-1,j+1)
             - (input (i+1,j-1) + 2*input(i+1,j) + input(i+1,j+1))|
            + |(input (i-1,j+1) + 2*input(i,j+1) + input(i+1,j+1))
             - (input (i-1,j-1) + 2*input(i,j-1) + input(i+1,j-1))|
```

When applied to a source image, the Sobel filter generates an image of the edges in the source image. More information on this operation can be found at http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm. An image with the Sobel filter applied is shown in Figure 2:



(a) Source image



(b) Filtered image

Figure 2: A source image and the result of applying the Sobel filter.

Sobel filtering is not quite "embarrassingly parallel" because a thread operating on a pixel at the edge of that thread's domain needs to access neighboring pixels which "belong" to a neighboring thread. The beauty of shared memory is that no messages between threads are required to perform this access!

Fill in the TODOs in sobel_sequential.c, sobel_pthreads.c, and sobel_openmp.c. Repeat the testing procedure you performed in the first part with these programs. **Graph, interpret the results, report code size, and show the Sobel-filtered version of a test image as before**.

Deliverables

A single zip file with:
-binarize_sequential.c
-binarize_pthreads.c
-binarize_openmp.c
-sobel_sequential.c
-sobel_pthreads.c
-sobel_openmp.c
-PDF report

Note the architecture of the computer you are using to test your programs (e.g. "64-bit dual core 2.4 GHz Intel Core i5").