Program #5: Shared Memory Multithreading Programming

Basic Image Processing

For this programming assignment, we will be looking at some simple image analysis and manipulation routines. In order to keep this manageable and focus on the parallel programming aspects, the actual algorithms used are highly simplified. You will be counting the number of pixels with given rgb saturation values using pthreads.

Histogram Analysis
A common image manipulation function is to count the number of pixels in an image that are light, dark, midtones, etc. This is useful in determining how under- or over-exposed a photograph might be, for example. For this, we need to *accumulate* the number of pixels that exist at each level. We will actually accumulate *four* different values: a red histogram (0-255), green histogram (0-255), blue histogram (0-255), and sum histogram (R+G+B = 0-765) for each pixel. For a simple uniprocessor version of the code, you would store the values in histogram variables like these:

```
int rHist[256], gHist[256], bHist[256];   /* RGB histogram "buckets" */
int sHist[766];                           /* R+G+B histogram "buckets" */
```

Using the following code for each pixel:

```
rHist[pixel_red]++;
gHist[pixel_green]++;
bHist[pixel_blue]++;
sHist[pixel_red + pixel_green + pixel_blue]++;
```

When complete, you should save your results to a file with the following loop, running on one processor:

```
for (i=0; i < 256; i++)
      fprintf(outputFile, "%3u: R:%8u G:%8u B:%8u S0:%8u S1:%8u S2:%8u\n",
              i, rHist[i], gHist[i], bHist[i], sHist[i], sHist[i+256],
              sHist[i+512]);
```

2A) First allocate your four histograms as a shared memory object protected by locks for each histogram bucket, on groups of 32 histogram buckets, and on entire histograms (for a total of 3 combinations). Run these on 1, 2, 4, 8, and 16 processors. How well do they speed up? Which ones are better/worse and why?

2B) Now allocate private histograms for each processor with a final reduction to one set of histograms at the end (which you can choose to do serially or in parallel). (Note that locks are not necessary as only one processor will be working on its private histograms at a time.) How well does this speed up on 2, 4, 8, and 16 processors? Plot this with the best results from 2A on the same graph and contrast.

2C) Live dangerously by turning off your locks in the shared histogram code (if you did this right, you can just `#define` them to nothing!) and re-run the 2, 4, 8, and 16 processor runs

without them. Do you notice much difference in the final results when compared with the "correct" results? Do using different photos make a difference? Why or why not?

When turning in your code for part 2, your solution should include code for all 3 cases (i.e. 2A-2C). You can choose either to submit different versions of your code file, one for each of these 3 cases, or to submit one code file where you use `#if/#endif` preprocessor directives to insure only one version of your implementation is compiled into the executable at a time. Please document your approach in your README.

Code

You'll need to download the code from the programming assignment webpage. There is a `Makefile`, the main code file `program5.c`, and a directory of jpeg files to use.

Your code will be written in the `CS338-function()`. For the different configurations within each part, you should comment out configurations you are not currently using.

A Note on Performance Times: Because of effects like interference from other users, OS paging, I/O, and the like you should be very careful with interpreting performance timing information. I recommend always running an application at *least* twice in succession before collecting data for your written reports, in order to "warm up" the system with one run before taking numbers, and that you make several more runs afterwards until the average time per run stabilizes.

Collecting results

You should run your code on the `limia` or `lohani` (which both have 16+ processors) to collect your performance results. However, you should do most of your development on one of the other CS machines so as to not use the processor when someone else wants to collect performance results. You should use more than one input file to collect results for your write-up; I suggest using one small, one medium, and one large file to see if the problem size results in different results.