Project #3: MPI Programming

Basic Image Processing

For this part of the programming assignment, we will be looking at some simple image analysis and manipulation routines, similar to those used in programs such as Adobe Photoshop to adjust images. In order to keep this manageable and focus on the parallel programming aspects, the actual algorithms used are highly simplified. You will be using MPI to parallelize code that blurs an image.

"Blur" Filter
You are being given sequential code that performs a simple yet common image operation: *blurring*. This just consists of making each pixel in the image a weighted average of itself and its neighbors. Edges are smeared out by this averaging process, resulting in a soft-focus effect. Each pixel of the output image is produced by calculating the following mathematical formula three times for every pixel (once each for the red, green, and blue color components):

$$output(x,y) = \frac{\sum_{i=-(r-1)}^{r-1} \sum_{j=-(r-1)}^{r-1} input(x+i, y+j)(r-|i|)(r-|j|)}{\sum_{i=-(r-1)}^{r-1} \sum_{j=-(r-1)}^{r-1} (r-|i|)(r-|j|)}$$

Real blurring filters often use much more sophisticated weightings than the simple 1/r2 dropoff (as we move away from the original pixel) used here, but this will do for this assignment. Please note that the pixels are composed of three 8-bit *unsigned* chars (R, G, B), with a range of 0-255 each. Intermediate results will require larger integers, however, to avoid overflow. Through the final division, the output pixels are confined to the 0-255 range.

Your task is to parallelize the code currently in the `CS338_function()` using MPI. How you divide up the work among the different MPI processes is up to you. For example, you could choose either a block or cyclic data distribution. You might choose to group work in terms of rows, columns, or rectangular/square blocks when distributing that work to the MPI processes.

Whatever configuration you use, run it using 1, 2, 4, 8, and 16 processes. Provide a brief write-up that

1) Describes your parallelization scheme,
2) shows how well the parallelization speeds up the work over the sequential code as number of processes increases,
3) discusses which parts of parallelization were difficult, and
4) reflects on what you would have had to do differently in your implementation if you had written this code using only Unix processes (e.g., fork and exec) and pipes.

Code
You'll need to download the code from the programming assignment webpage. There is a `Makefile`, the main code file `program3.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.