ECE574: Cluster Computing – Homework 6
MPI

Due: Friday, 28 March 2025, 5:00pm

1. **Background**

   - In this homework we will take the sobel code from earlier homeworks and parallelize it using MPI.

2. **Setup**

   - For this assignment, log into the same Haswell-EP machine we used in previous homeworks. As a reminder, use the username handed out in class and ssh in like this
     `ssh -p 2131 username@weaver-lab.eece.maine.edu`
   - Download the code template from the webpage. You can do this directly via
     `wget https://web.eece.maine.edu/~vweaver/classes/ece574/ece574_hw6_code.tar.gz`
     to avoid the hassle of copying it back and forth.
   - Decompress the code
     `tar -xzvf ece574_hw6_code.tar.gz`
   - Run `make` to compile the code.
   - You may use your own code from a previous assignment as a basis for this assignment. (Alternately some really poorly-optimized sample code is provided).

3. **Coarse-grained Code (8 points)**

   Use MPI to parallelize your code. Use the sample code, or you might want to use one of your previous assignments as a basis.

   Note the provided sample code does the following things for you:

   - Includes `mpi.h`
   - `MPI_Init()` is called at the start
   - `MPI_Finalize()` is called at the end
   - Uses `MPI_Comm_size()` to get the total number of ranks
   - Uses `MPI_Comm_rank()` to get the rank number of the currently running code
   - In rank 0 prints a debug message saying how many ranks there are.
   - Uses `MPI_Wtime()` to record the times for load/convolve/combine/store and print these at the end in rank 0.

   Edit the file `sobel_coarse.c`

   Be sure to comment your code!

4. **A suggested first (coarse) implementation**

   (a) **Load and Broadcast the Image Data**

   - Modify the code to only load the jpeg from disk in rank 0
   - Broadcast the image sizes to all ranks
     - Create an integer array with three integers. Set these to the values of `image.xsize`, `image.ysize`, `image.depth`
     - Use `MPI_Bcast()` to send this array from rank 0 to all the other ranks
     - Make sure the other ranks set their values of `image.xsize`, etc, from the array
   - Allocate memory for the image pixel array in all ranks
     - In non-rank 0 you will need to `calloc()` `image.pixels` (usually `load_jpeg()` does this but that doesn't get called on non-rank 0)
   - Broadcast the image data from rank 0 to all other ranks
     - use `MPI_Bcast()` to broadcast the `image.pixels` data from rank0 to all the other ranks.
     - **Note:** You want to broadcast `image.pixels`, not the entire `image` struct as in MPI you can't easily send structs, just arrays).
     - A `MPI_Bcast()` has an implicit barrier, so after this point all ranks should be in the same place, and all should have copies of the full image data.
   - Verify the checksum
     - It is important the pixel data transfers correctly, and this has caused a lot of issues over the years. I've added a checksum that runs on the image data on all ranks. That way you can ensure the data is being broadcast successfully.
     - Assuming your code works, when using the butterfinger input then all ranks should print `0x1edff87` as the checksum found on the input data.

   (b) **Do the Convolutions**

   - Generate the proper values to pass to `generic_convolve()`.
     As with HW#4 you'll have to manually split up the work by rank.
   - Before the convolve calls set `ystart` and `yend` values based on your rank number.
   - **In each rank print the start/end numbers and verify all y values are being calculated**
   - Be sure you handle the special cases of top and bottom to be +1 / -1 for the border
   - Note: instead of convolving into `sobel_x` or `sobel_y` directly it might be helpful to convolve into a temporary result, perhaps using `new_image`. This can make the following gather step easier.

   (c) **Prepare for Gather**

   Gathers by default will gather from the start of an array, whereas your convolve code probably puts results at an offset depending on the rank. There are a few ways you can adjust for this. If you forget to do this, your result will be blank for the bottom part of your output image.

   - The most straightforward way is to adjust your convolve routine to place the output at the start of the array (by subtracting the initial ystart from your y value when setting the output pixels. Note be sure it's the initial ystart, not the one adjusted for the border, or your results will be subtly off),
   - Another way is doing a `memcpy()` to move the results to the start of the array,

- Finally you can adjust the gather source value using pointer math to point to the proper offset in the data array `sobel_x.pixels[rank*total_size/num-ranks]`

(d) **Gather the results**

- Use `MPI_Gather()` to get the results from all the ranks into rank 0
- Note: you only want to gather the pixel data (the array of chars) not the structure containing it. So you want to gather `sobel_x.pixels` not `sobel_x`
- In a previous step we recommended you convolve into a temporary results rather directly into `sobel_x`. This is because MPI by default won't let a gather have the same source and destination (i.e. on rank 0 you can't gather from all `sobel_x` into your own `sobel_x`

(e) **Run Combine**

- After calculating sobel X and sobel Y it's time to combine.
- On rank 0 alone, run the combine.

(f) **Output to File**

- On rank 0 alone, output the data to an image file

(g) You can test your code with a command like: `mpirun -np 4 ./butterfinger.jpg`
For final runs, use slurm as so: `sbatch -n X time_coarse.sh`
where you replace X with the number of cores to use.

5. **Handle tail end data (1 point)**

(a) Get your code working with regular Gather() first.

(b) Your code will likely only work if the image ysize is a multiple of the total rank number.

(c) Copy your `sobel_coarse.c` file over top of `sobel_complete.c` and edit that file for this part (this lets me grade this separately in case things break when you're trying to do this part).

(d) Modify your code to handle ysizes that aren't a multiple of total rank. Use Gatherv() to implement this as discussed in class.

6. **Report your Results (1 point)**

- Run on the Haswell-EP machine for 1, 2, 4, 8 and 16 threads and report the results, as well as reporting the speedup and parallel efficiency for the total time.
- Run your code with:
`sbatch -n X time_coarse.sh`
where you replace X with the number of cores to use.
- If (for fun) you want a bigger image to test with, try `/opt/ece574/jan_15_2017_high_res.jpg`

7. **Some Debugging Hints**

- If you have puzzling results, debug at each step of the way.
- Start by testing the N=1 case, then N=2 case
- Things to watch for:
  - If you get a diagonal pattern in the output, be sure you are gathering in even multiples of `xsize`

  - If only the top part of the image is in the results, make sure you are moving the data to the right place in your `MPI_Gather()`
  - Be sure your limits are set properly. Print the limits out and verify they are being set properly.
  - When broadcast image data, make sure you are sending CHAR not INT
  - If the top of your image is fine but the rest is offset or has weird rainbow patterns, this usually means you are gathering a size of `(ysize*xsize*3)/num_ranks` rather than `(ysize/num_ranks)*xsize*3`
  - If the result looks fine but you're still not matching the expected result, usually it means your edge values are wrong. Remember we go from 1..xsize-1 and 1..ysize-1

8. **Submitting your work.**

   - Be sure to edit the README to include your name, as well as the timing results, and any notes you want to add about your something cool.
   - Run `make submit` and it should create a file called `hw06_submit.tar.gz`. E-mail this file to me.
   - e-mail the file to me by the homework deadline.