ECE575: Cluster Computing – Homework 7
MPI

Due: Thursday 6 April 2017, 3:30pm

1. **Background**

   - In this homework we will take the sobel code from earlier homeworks and parallelize it using MPI.
   - You may work in a group for this Homework.

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 http://web.eece.maine.edu/~vweaver/classes/ece574_2017s/ece574_hw07_code.tar.gz`
     to avoid the hassle of copying it back and forth.
   - Decompress the code
     `tar -xzvf ece574_hw07_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 (6 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.

   If not using the sample code you will need to make sure your code includes `mpi.h` and that it calls `MPI_init()` at the beginning and `MPI_Finalize()` at the end.

   Edit the file `sobel_coarse.c`

   Be sure to comment your code!

   A suggested first (coarse) implementation is this:

   (a) Get the rank and size parameters.

   (b) Be sure to only load the jpeg in rank 0.

   (c) You will need to send the image parameters (`image.x`, `image.y`, `image.depth`) to all the other ranks so they know how big to allocate new_image, sobel_x, and sobel_y. MPI is optimized for sending arrays of same sized data, so sending an array of 3 INTS might be your best bet.

   (d) You will also need to malloc `image.pixels` in the non rank-0 threads. (Because usually it's load jpeg that does that).

   (e) Use `MPI_Bcast()` to broadcast the entire image data from rank0 to all the other ranks. You want to broadcast "image.pixels", not all of image (remember, MPI you can't send structs, just arrays).

(f) Modify generic_convolve so it takes a range of y to operate on. Then calculate this y range based on the rank and size parameters. (For example, if we detect size of 4, then we are running on 4 nodes and each rank should get 1/4 of the results)

(g) Be sure to run both sobel_x and sobel_y on the subset for that rank.

(h) Use `MPI_Gather()` to get the results and combine them into the result in rank 0. NOTE: `MPI_Gather()` will gather from the *start* of each buffer and put it in the proper place in the result. So you have to modify the convolve routine to store the output starting at offset 0, rather than at offset ystart. If you forget this, the bottom ((N-1)/N)th of your image will be blank.

(i) On rank 0 alone, run combine.

(j) On rank 0 alone, write the output to a file.

(k) Also be sure the code calls `MPI_wtime()` to get the wallclock times for Load, Convolve, Combine, and store much like we did with PAPI in the OpenMP code. You only need to record and print these from rank 0.

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. This will run on the provided `IMG_1733.JPG`.

Slurm is smart enough to mpirun with the right number based on the value you pass to it.

The md5sum of the expected output is `948a681b6c8768b28ac0a3329de5ec2c`.

If (for fun) you want a bigger image to test with, try `/opt/ece574/jan_15_2017_high_res.jpg`

4. **Fine Grained (3 points)**

   Use some method to improve the parallelism and see if it improves the runtime.

   Copy your code to `sobel_fine.c` and edit it.

   Various things that might improve performance:

   - Reading the image from all nodes rather than just rank 0.
   - Scattering the image info (only sending the part needed rather than sending it all). This is tricky as you will need to send 2 extra rows, it's not an even split.
   - Parallelize the combine code.
   - Using some of the more advanced MPI calls. We didn't cover all of them in class.

   Report before and after times for 1, 4, 8, and 16 cores as well as speedup and parallel efficiency.

5. **Pi Cluster (1 point)**

   Run your code on the Raspberry Pi Cluster. Either the fine or coarse, specify which one.

   To log in, first log into the haswell-ep machine as per usual.

   Next you will need to ssh one more time,

   ```
   ssh pi-cluster
   ```

Your password should be the same as it was on the Haswell machine.

Copy your code to the pi cluster. First “make clean” then cd down a directory (“cd ..”) and do `scp -r ece574_hw07_code pi-cluster:`

Run `make clean` and `make` to recompile for the ARM architecture.

Run your code for 1, 4, 8, and 16 cores. `sbatch -n X time_coarse.sh`

Report your timing in the README and comment on the scaling behavior on the Pi cluster.

The md5sum might be different on the pi than on x86. (Not 100% sure why. Different version of jpeg library?).

6. **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 `hw07_submit.tar.gz`. E-mail this file to me.
   - e-mail the file to me by the homework deadline.