ECE575: Cluster Computing – Homework 6
MPI

Due: Tuesday 10 November 2015, 3:30pm

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 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@vincent-weaver-2.umelst.maine.edu`
   - Download the code template from the webpage. You can do this directly via
     `wget http://web.eece.maine.edu/~vweaver/classes/ece574_2015f/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 (5 points)**

   Use MPI to parallelize your code. Use the sample code, or you might want to use your pthreads code as a reference since you will have to split up the loops manually.

   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. 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.

   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, and be sure to only load the jpeg in rank 0.

   (b) Use `MPI_Bcast()` to broadcast the image info (x, y, and depth) to all nodes.

   (c) Allocate an appropriate sized array to hold the image on each node based on the values passed over.

   (d) For sobel_x and sobel_y do the following:

      i. Use `MPI_Bcast()` to broadcast the actual image data.

      ii. On each node convolve on a section of the image data.

      iii. Use `MPI_Gather()` to get the results and combine them into the result in rank 0.

(e) On rank 0 alone, run combine.

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

Run on the Haswell machine for 1, 2, and 4 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`.

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

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

   You can do this on the Haswell machine but it will actually be more meaningful on an actual cluster (the pi-cluster).

   If you want to use the Haswell machine instead that is fine, just report in the README which machine you used.

   Edit `sobel_fine.c`.

   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 and 8 cores as well as speedup and parallel efficiency.

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

   Run your code on the Raspberry Pi Cluster.

   Note, as of posting this I don't have the accounts set up so you can't log in yet. I'll send an e-mail saying when this is available (hopefully by Friday).

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

   Next you will need to ssh one more time, through the reverse-ssh tunnel.

   ```
   ssh -p 19998 localhost
   ```

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

   Copy your code to the pi cluster. You can do something like cd down a directory and do
   `scp -r -P 19998 ece574_hw6_code localhost:`

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

   Run your code for 1, 4, 8, and 16 cores. To do this, modify the `time_fine.sh` code. Change the "number of cores" setting to the number of cores and run it.

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

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 `hw06_submit.tar.gz`. E-mail this file to me.
   - e-mail the file to me by the homework deadline.