ECE575: Cluster Computing – Homework 5

OpenMP

Due: Monday 26 October 2015, 5:00pm

1. Background

   - In this homework we will take the sobel code from Homeworks #3 and #4 and parallelize it using OpenMP.

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_hw5_code.tar.gz
     ```
     to avoid the hassle of copying it back and forth.
   - Decompress the code
     ```
     tar -xzvf ece574_hw5_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). To use your code just copy your un-paralallized code over `sobel_before.c` and your coarse code over `sobel_coarse.c` and your fine code over `sobel_fine.c`.

3. **Coding (6 points)**

   Implement simple two-thread OpenMP parallelism where you run sobel_x and sobel_y in parallel, but it joins before doing the combine step serially.

   To do this, use the OpenMP Sections directives. Remember that OpenMP will automatically do a join at the end of a parallel section.

   - Edit the file `sobel_coarse.c`
   - Convert the code to use OpenMP.
   - You may need to add `#include <omp.h>`
   - Be sure to comment your code!
   - Compare the results generated to make sure they match the output given by previous homeworks.
   - Run your code using
     ```
     sbatch time_coarse.sh
     ```
     which will use the provided `IMG_1733.JPG`.
     Report how long it takes to run compared to the non-parallel code. You can use
     ```
     sbatch time_before.sh
     ```

4. **Performance Measurement (2 points)**

   - Just like HW#4, have your code measure the total convolution time, the combine time, and the `load_jpeg()` and `store_jpeg()` times using PAPI and print the results to the screen.
   - Calculate the speedup and parallel efficiency compared to the non-parallel version and report the results in your README.

5. **Fine-grained Threading (2 point)**

   For this question do some sort of fine-grained parallelism. How you do it is up to you. The most straightforward way of doing this is using an OpenMP `for` directive. The easiest way to do this is to go into your `convolve` and `combine` routines and convert one of the for loops to be parallel.

   - For this exercise modify the `sobel_fine.c` file.
   - Some things to watch out for: remember to mark as private your various loop iterators and other variables (such as sums, etc.)
   - If you don't want to have to keep checking the image to be sure your code is working, an alternate is to use a checksum like `md5sum` to verify the output file matches.
   - Record the total time (using time) as well as the PAPI timing measurements for 1, 2, 4, and 8 threads. The easiest way to do this is by setting the `OMP_NUM_THREADS` variable before running your timing.

     One way to do this is at the shell prompt: i.e. typing `export OMP_NUM_THREADS=1`, running, then `export OMP_NUM_THREADS=2`, running, etc.
   - Does changing the thread scheduler from static to dynamic change your performance?

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 `hw05_submit.tar.gz`. E-mail this file to me.
   - e-mail the file to me by the homework deadline.