ECE574: Cluster Computing – Homework 6

OpenMP

Due: Thursday 2 March 2017, 3:30pm

1. **Background**

   - In this homework we will take the sobel code from Homeworks #4 and #5 and parallelize it using OpenMP.

2. **Setup**

   - You may work in groups.
   - 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_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). It might make more sense to reuse your HW#4 code or the HW#5 coarse code as a basis rather than having to back out cases from HW#5 fine code. Just copy your un-parallelized code over `sobel_before.c`, `sobel_coarse.c` and `sobel_fine.c`.

3. **Coarse-grained Parallelism (4 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 (`sbatch time_before.sh`)

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

   - Just like HW#5 use PAPI to measure the time various subcomponents take to run. Have your code print to the screen the time taken by:
     (a) `load_jpeg()`
     (b) sobelx
     (c) sobely
     (d) combine
     (e) `store_jpeg()`
   - Calculate the speedup and parallel efficiency compared to the non-parallel version and report the results in your README.

5. **Fine-grained Threading (4 points)**

   For this part, update the code to 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. (the md5sum of the sobel output from IMG_1733.JPG is `948a681b6c8768b28ac0a3329de5ec2c`)
   - Record the total time (using time) as well as the PAPI timing measurements for 1, 2, 4, 8, 16, and 32 threads. The easiest way to change the number of threads is to change the `OMP_NUM_THREADS` variable in `time_fine.sh` before running sbatch, but you can also use one of the other ways of specifying thread numbers.
   - Does changing the thread scheduler from static to dynamic change your performance in the 16-thread case?

6. **Submitting your work**

   - Be sure to edit the README to include your name, as well as the timing results and answers to questions.
   - Run `make submit` and it should create a file called `hw06_submit.tar.gz`.
   - e-mail the file to me by the homework deadline.