CSCI 551 – Numerical and Parallel Programming: Exercise #1

Familiarity with Matrices and Using Concurrency for Speed-Up

DUE: As indicated on Canvas

Please thoroughly read Week-1 and Week-2 notes and bring your questions to discussion. You should expect to spend about 1 hour per 10 points, and complete spending 5 hours per week. This leaves you 4 hours per week for reading, coming to office hours, etc. Please read **Pacheco Chapter 5 – 5.1 to 5.5** for introductory coverage on OpenMP compiler directives and read **Pacheco Chapter 4 – 4.1 to 4.6** on basic Pthread creation. ***Keep in mind that OpenMP is a compiler feature and Pthreads are an API/library feature***. Reading in Pacheco Chapter 1 and 2 is helpful, but mostly theory that will also be covered in class. Some additional OpenMP compiler and directive documentation you may find useful:

1. https://bisqwit.iki.fi/story/howto/openmp/ - OpenMP HowTo
2. GCC openmp docs - https://gcc.gnu.org/wiki/openmp gory details of compiler directives

You will need an ECC system (***ecc-linux.csuchico.edu***) account to work on this assignment or access to your own home Linux system. Most students can just SSH to the ECC system using MobaXterm, Cyberduck, or Putty or whatever your favorite SSH tool is, but please obtain an account one if you don't already have (https://www.csuchico.edu/itss/index.shtml).

Note that you can cone the tested started code from https://github.com/sbsiewertcsu/numeric-parallel-starter-code (recommended) or download the starter code from the class web-site as a gzip at http://www.ecst.csuchico.edu/~sbsiewert/csci551/code/, which is "sandbox" code that may be in the process of update. ***Github is the best option – use "git clone https://github.com/sbsiewertcsu/numeric-parallel-starter-code.git" to make a copy on ECC-Linux. All github code should just build with "make" and should run and has no known bugs, but should not be considered solution code, just a start in that direction.***

A major goal for CSCI 551 is to write as much code as possible from the ground up without use of libraries beyond basic C library, math, pthreads, and input/output, but limited use of libraries that are not used for parallel coding is acceptable. E.g., reading in and writing out specific file formats (you might use OpenCV or STB image for example), but please keep the minimal library use goal in mind.

All code submissions must be accompanied by a Makefile or similar build automation with documentation on how to build with a "single command" from the command line on either ECC server (ecc-linux.csuchico.edu) or a Raspberry Pi or Jetson Linux system. Please zip up your code when you submit it to Canvas but provide key code snippet examples in your reporting to describe your code and clearly note what you have written, what you have re-used from any other source (must be cited in the code or in your report or both).

Report your results, clearly answering each of the questions below with analysis, example output, code snippets, and/or mathematical analysis and solutions (answers clearly indicated by boxing them).

Exercise #1 Objectives:

1) [50 points] Download the C DCT code examples (omp_dct2), and read the references provided in the code and notes from Week-1-2 to familiarize yourself with the use of the DCT (Discrete Cosine Transform). The basic code is used for testing and benchmarking single thread and multi-thread OpenMP versions and does not process an image file. However, this code can be adapted for image transformation by adding code to read in an image, parsed into macroblocks, and to write out those macroblocks. If you want to test it with real images, you may also, download code and test images as needed from my web page (code/, Comp-Vision-Images/).

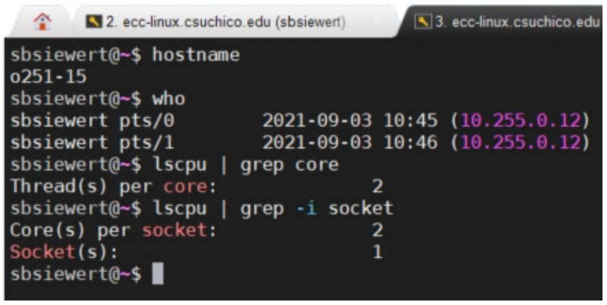
    a) [10 pts] First download and test a very simple OpenMP program "hello_omp.c", (csci551/code/hello_openmp/hello_omp.c), and make sure you understand the output and describe what you see when you run it. Read through the basic methods of using OpenMP directives in Pacheco, Chapter 5, section 5.1, and section 5.5, and determine if there are alternative ways to use the "omp" directives for programs that call functions in loops. Based on your understanding, come up with a way to add a loop to this basic code to print out the message 16 times with 4 omp threads that you believe is most simple and efficient.
    b) [10 pts] Build and run DCT dct2.c single thread code and report on results when you build and run it on a cluster node. You can use the Linux "time" function (e.g., "time ./dct2") to time the run or you can add in ***clock_gettime*** calls (see "man clock_gettime") to time the execution of this code. Determine how many 1280x960 frames you can process per second on your test system (ECC, Raspberry Pi, or other) and document how you determined.
    c) [10 pts] Build and run DCT ompdct2.c multi-thread code and report on results when you build and run it on a cluster node. You can use the Linux "time" function (e.g., "time ./dct2") to time the run or you can add in ***clock_gettime*** calls (see "man clock_gettime") to time the execution of this code. Determine how many 1280x960 frames you can process per second on your test system (ECC, Raspberry Pi, or other) and document how you determined. How much did the "omp" directive in this code speed-up this version compared to dct2.c?
    d) [10 pts] Using ompdct2.c, vary the number of threads from 1, 2, 3, up to 4 and time each and report on the time to complete each run (real time) and plot as a function of the number of threads using Excel or your favorite graphing tool.

e) [10 pts] Based upon the maximum speed-up you observe between dct2.c and ompdct2.c, determine the parallel and sequential portion of the code in ompdct2.c based on Amdahl's law.

2) [50 points] For this exercise, download the ECC tested code (http://www.ecst.csuchico.edu/~sbsiewert/csci551/code/psf-convolution-threaded.zip) and compare to an OpenMP version of the same code (openmp-sharpen.zip).
   a) [10 pts] Modify both versions (sharpen.c with #pragma for OpenMP & sharpen_grid.c using Pthreads) so that they sharpen the 1280x960 test image multiple times (e.g., 3 or more) long enough to see activity on "htop" and report the frame rate for application of the PSF sharpen. Document and explain the changes you make. Read Chapter 24 of the DSP Engineer's Handbook, and briefly describe the use of PSF convolution.
   b) [10 pts] Single thread compared to Pthread - Test the PSF sharpen code for 90 iterations with the 1280x960 pixel test image using both the single thread (sharpen.c) and the multi-threaded (sharpen_grid.c) and compare time to complete (using built in timestamps or the Linux "time" command). Report your results.
   c) [10 pts] OpenMP sharpen compared to Pthread - Test the PSF sharpen code for 90 iterations with the 1280x960 pixel test image using both the OpenMP modified sharpen.c code and the multi-threaded (sharpen_grid.c) and compare time to complete (using built in timestamps or the Linux "time" command). Report your results.
   d) [20 pts] Compare all 3 versions (single thread sharpen.c, OpenMP sharpen.c, and Pthread sharpen_grid.c) and compute the speed-up of the Pthread and the OpenMP compared to the single thread version. Compare this to Amdahl's law and linear ideal speed-up on a graph for the 4x3 thread gridding for the system you test it on (the "S" in Amdahl's law can be assumed to be # of cores).

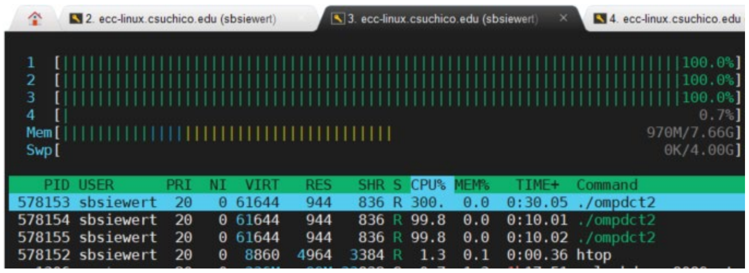
      More background on "S" - Note the system that you test on and capture the CPU info with "lscpu" or "cat /proc/cpuinfo" to determine hardware scaling factor "S" (see Figure 1). For S used for Amdahl's law, set S to # of cores and multiply by 2x if your machine has SMT (hyperthreading). Setting S to # of cores is always safe, but some systems have SMT (Symmetric Multi-Threading micro-parallel features), so check number of virtual cores (use "htop" for example) and double S if necessary (see Figure 2). The number of threads should be at least as large as S, but ideally 2x or 3x the value of S.

Figure 1 – ECC Cluster node o251-15 CPU cores and SMT – Dual SMT, so S=4



```
sbsiewert@~$ hostname
o251-15
sbsiewert@~$ who
sbsiewert pts/0        2021-09-03 10:45 (10.255.0.12)
sbsiewert pts/1        2021-09-03 10:46 (10.255.0.12)
sbsiewert@~$ lscpu | grep core
Thread(s) per core:              2
sbsiewert@~$ lscpu | grep -i socket
Core(s) per socket:              2
Socket(s):                       1
sbsiewert@~$ 
```

Figure 2 – Using "htop" on ECC and Cluster nodes (o251 and o244)



```
  1  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
  2  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
  3  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
  4  [|                                                               0.7%]
  Mem[|||||||||||||||||||||||||||||||||||                      970M/7.66G]
  Swp[                                                            0K/4.00G]

    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 578153 sbsiewert  20   0 61644   944   836 R 300.  0.0  0:30.05 ./ompdct2
 578154 sbsiewert  20   0 61644   944   836 R 99.8  0.0  0:10.01 ./ompdct2
 578155 sbsiewert  20   0 61644   944   836 R 99.8  0.0  0:10.02 ./ompdct2
 578152 sbsiewert  20   0  8860  4964  3384 R  1.3  0.1  0:00.36 htop
```

Overall, provide a well-documented professional report of your findings, output, and tests so that it is easy for a colleague (or instructor) to understand what you have done, what worked, what did not and why (even if you cannot complete to your satisfaction). Provide clear instructions on how to run your programs, including command line arguments required and screenshots demonstrating use and test cases you used to verify your parallel and sequential programs. For all parallel programs please show you have completed the triple challenge (still works, has speed-up, can scale-up and if applicable, can scale-out).

Include any design files or log files, C/C++ source code you write (or modify) and Makefiles needed to build your code. I will look at your report first, so it must be well written and clearly address each problem providing clear and concise responses and example results (e.g. summary analysis and clearly boxed mathematical answers) to receive credit, but I will look at your log files, code and test results as well if I have questions.

Report file MUST be separate from the ZIP file with code and other supporting materials.

Rubric for Scoring for scale 0…10

Score	**Description of reporting and code quality**
0	No answer, no work done
1	Attempted and some work provided, incomplete, does not build, no Makefile
2	Attempted and partial work provided, but unclear, Makefile, but builds and runs with errors
3	Attempted and some work provided, but unclear, build warnings, runs with no apparent error, but not correct or does not terminate
4	Attempted and more work provided, but unclear, build warnings, runs with no

	apparent error, but not correct or does not terminate
5	Attempted and most work provided, but unclear, build warnings, runs with no apparent error, but not correct or does not terminate
6	Complete answer, but does not answer question well and code build and run has warnings and does not provide expected results
7	Complete, mostly correct, average answer to questions, with code that builds and runs with average code quality and overall answer clarity
8	Good, easy to understand and clear answer to questions, with easy-to-read code that builds and runs with no warnings (or errors), completes without error, and provides a credible result
9	Great, easy to understand and insightful answer to questions, with easy-to-read code that builds and runs cleanly, completes without error, and provides an excellent result
10	Most complete and correct - best answer and code given in the current class

<u>Grading Checklist for Rubric</u>

[50 points] Exploring DCT matrix transformations with OpenMP:

Problem #1	Score 0…10	Comments
a		
b		
c		
d		
e		
TOTAL		

[50 points] Parallel PSF Convolution code for ECC:

Problem #2	Score 0…10	Comments
a		
b		
c		
Problem #2	Score 0…20	Comments
d		
TOTAL		