CSCI 551 – Numerical and Parallel Programming: FINAL PROJECT

Improvement and Refactoring of Numerical-Parallel Program

OR

Creative Project for Numerical-Parallel Programming

DUE: As indicated on Blackboard

Please propose a final project for a numerical problem you can solve and speed-up using parallel programming methods learned in class. The problem can be one of the problems from exercises #1 to #6, re-designed, re-factored, improved, with minor additional features OR it can be a creative project proposal to solve a new problem, of your own interest, that requires numerical methods and computations and can benefit from parallel programming.

Your programs should run on the ECC system, a Raspberry Pi, Jetson, or other commonly available Linux system, using Intel Parallel Studio XSE compilers and tools and/or GNU open source compilers and tools.

ECC Cluster node use policies – Based on your birthday and year, if your year of birth is even, use "o244" nodes and if you birth year is odd, use "o251" nodes. For POSIX shared memory threading (single node use), login to the node # that is the same as your birthday – e.g. for me, odd year, 14th day of month, so I would use "ssh o251-14". This should help distribute the load as we get into problems that are more CPU, I/O, and memory intensive.

Final Project Objectives:

The learning objectives for the final project are:

- ***Choose a numerical and parallel problem that interests you and that is relevant to the class (studied or related to what we have studied)***, explain the problem well, and describe your proposed numerical method (algorithm), the features your program will have, and how you plan to design and program it using MPI, OpenMP, Pthreads, or some hybrid of the 3 methods (combining two or more), or some new method (e.g. CUDA).

- ***Implement a sequential program that solves your problem*** and time it accurately for variously sized problem scenarios as well as describing the algorithmic complexity.

- ***Design and implement a parallel solution*** using one of the parallel programming methods we studied on a multi-core / multi-node system that provides shared memory or distributed memory processor scaling.

- ***Show that your parallel design and implementation speeds-up solution and that it scales*** with number of processor cores used.

- ***Provide a code walk-through of your parallel solution, demonstrate build, and run, and describe*** how it works.

1) [25 points total] ***Numerical and Parallel Problem Description*** - Please answer each of the following by indicating use and describing your particular use of the numerical method(s) and the parallel programming method(s). If you use more than one, indicated all used and describe how they are used together.

   [10 pts] ***Project description*** in terms of major goals and objectives, and problem solved.

   [5 pts] ***The value of your solution*** and applications it has.

   [5 pts] What ***numerical methods and algorithms*** are used and what type of math is required? Please ***indicate and then describe*** your use and objective for using the method.

Improvement and Refactoring Examples (Check Used Column)

Numerical Method	**Mathematics**	**Description**	**Used**
Vector/matrix	Convolution and transformation	Use of 2-D convolution and transformation functions applied to images (e.g. DCT, rotation, image sharpen/blur, etc.)	No
Prime number searching and testing	Prime number theorem	Use of Sieve of Eratosthenes and more advanced methods to find prime density, largest prime in range and list prime numbers in an interval	No
Integration	Calculus	Use of Riemann sums, Trapezoidal, Simpson's Rule, or advanced Runge-Kutta	Yes
Non-linear integration	Calculus, Accuracy and Precision	Integration of non-linear functions and sources of error compared to definite integrals	Yes
Gaussian Elimination with Partial Pivoting, Gauss Seidel, and LU factorization	Linear systems	Solving systems of equations that describe linear systems (circuits, fluid flow and concentration, etc.) or Linear Programming (optimization), or Regression (over defined)	No
Root solving with Newton Raphson or Regula Falsi	Non-linear systems	Use of root solving algorithms to find the intercept of non-linear functions – e.g. 2 ballistic trajectories	No
Other?	Probability	Other mathematics and numerical methods	No

	and Statistics, Discreate math, other?	used that you learned in this course or pre-requisite courses	
Description Please describe methods you used here.	I used a trapezoidal sum to estimate the definite integral of a non linear function, namely sin.		

[5 pts] What ***parallel programming methods*** are used?

Parallel Programming Methods Used (Check mark Used Column)

Parallel Programming Method	Description	Used
POSIX threads	Shared memory threading within a Linux process	Yes
MPI	Message Passing Interfaces between Linux processes on the same node or network interconnected nodes	Yes
OpenMP	Compiler directives to generate parallel shared memory code for specific parts of a program	Yes
Other	CUDA, DSP, etc.	
Description Please describe methods you used here.	I used all three methods of parallelization that we have studied this semester, including hybrid methods between MPI and Pthreads/OpenMP	

2) [25 points total] ***Sequential solution and computation time***.

[15 pts] ***Sequential program*** (provide source and Makefile), and demonstration of build and run. Code should be well commented and readable.

[10 pts] Use POSIX clock_gettime, ***MPI_Wtime***, or POSIX ***clock_gettime*** functions in your code to ***time and log start and complete of the program*** and run at least 3 times (ideally 10 or more) to get an average run time.

3) [25 points total] ***Parallel design and solution with computation time***.

[15 pts] ***Parallel program*** (provide source and Makefile), and demonstration of build and run. Code should be well commented and readable.

[10 pts] Use POSIX clock_gettime, ***MPI_Wtime***, or POSIX ***clock_gettime*** functions in your code to ***time and log start and complete of the program*** and run at least 3 times (ideally 10 or more) to get an average run time.

4) [25 points total] ***Parallel speed-up analysis***.

[10 pts] ***Determine parameters for Amdahl's Law***: sequential portion of your code **(1-P)**, parallel portion **P** (note that you can use "gcc -S" to generate instructions to count for best results or count C/C++ statements for a more approximate % of each) and then decide upon a range for number of processor cores **S** (must be at least 1 and 2, but ideally also 4 and 8 if possible). Document this in the table below.

Amdahl's Law parameter	**How obtained?**	**Description**
Sequential portion (% of total)	Indirect Method of timing and	adjusting Roughly 10%
Parallel portion (% of total)	Same as previous	Roughly 90%
Number of shared memory cores used and type	lscpu	1-4 (Tested up to 8 threads on a hyperthreaded system)
Number of distributed memory cores used and type	lscpu	1-4

[15 pts] ***Plot Amdahl's Law ideal speed-up and your actual speed-up*** based on timings from problem 3 for at least two data points or more. Describe how well your solution compares to the ideal potential speed-up.

5) [100 points total] ***Code walk-through video or ZOOM live presentation [20 to 40 minutes in length]***.

Present your parallel code and demonstrate build and run followed by a walk-through line by line to describe how it works. You can record this as a video (see instructions for Windows and Macintosh recording). Make sure this is high quality and that it is complete, correct, consistent and clear.

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 can't 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.

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 (adjust for 0…5 and other scales with 0.0 to 1.0)

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

Grading Checklist for Rubric

[25 points] ***Project introduction, significance, and description***:

Problem	Score	Possible	Comments
Goals and objectives with problem statement and solution proposed		10	
Value of your solution (why others should "care")		5	
Numerical methods used		5	
Parallel programming methods used		5	
TOTAL		25	

[25 points] ***Sequential solution and computation time***:

Problem	Score	Possible	Comments
Sequential solution		15	
Time measurement for sequential solution		10	
TOTAL		25	

[25 points] ***Parallel design and solution with computation time***:

Problem	Score	Possible	Comments
Sequential solution		15	
Time measurement for sequential solution		10	
TOTAL		25	

[25 points] ***Parallel speed-up analysis***:

Problem	Score	Possible	Comments
Estimation of Amdahl's law parameters for your parallel solution		10	
Comparison of linear, Amdahl's law and your actual speed-up		15	
TOTAL		25	

[100 points] ***Code Demonstration and Walkthrough***:

Aspect	Score	Possible	Comments
Professionalism		5	
Quality of code submitted – used for walk-through (builds without errors, runs well, commented well, readable, modularized with functions, etc.)		10	
Technical content - (error free, correct use of terminology, concise, but complete)		5	
Interest - motivated and interested in topic, shows passion for work done, concept is clear		5	
Build demonstration – parallel and sequential		10	
Run demonstration – parallel and sequential		10	
Demonstration of speed-up and scaling - of parallel implementation and comparison to sequential		10	
Source code description - function by function and line-by-line as needed		10	
Description of speed-up attained - compared to ideal, original sequential program and Amdahl's law		10	
Speed-up and scaling result - explanation of methods used and whether more improvement is possible		10	
Numerical method(s) used - how well they are explained and used		10	
Significance of solution – why the problem is relevant, of interest to others, and how this solution helps solve the problem		5	
TOTAL		100	