CSCI 551 – Numerical and Parallel Programming: Exercise #4

SPMD (MPI and CUDA) Parallel Scaling, Interpolation and Methods of Integration for Simulation

From the cover of "An Introduction to Error Analysis – The Study of Uncertainties in Physical Measurements", by John R. Taylor, University Science Books, 2022.

DUE: As indicated on Canvas

Please thoroughly read Week-5 and Week-6 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.

You will need a "cscigpu" system (***cscigpu.csuchico.edu***) and ECC-Linux account or access to a native Linux system that is multi-core to work on this assignment. 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 download the starter code from the class website as a gzip at http://www.ecst.csuchico.edu/~sbsiewert/csci551/code/, which is the last distribution. For analysis and design work, you will find the **starter code hello_cluster and hello_cuda useful,** as well as use of the Pacheco textbook and reference book for methods of numerical integration and outside tutorials on MPI (https://hpc-tutorials.llnl.gov/mpi/).

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 the "cscigpu" server (cscigpu.csuchico.edu) or a cluster node where MPI is available with the Intel Parallel Studio XE installation. 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).

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.

Note for MPI first use: To use ***mpirun*** with a host file and multiple nodes, ***you must set up SSH, so you have a key-based trusted login*** to all cluster nodes you intend to use. Please see the following documentation on the cluster and note that you should "ping" and test "ssh" before using MPI on nodes: http://www.ecst.csuchico.edu/~sbsiewert/csci551/README-cluster.html. For MPI scaling on single node (good for basic testing), you can use ***mpiexec*** and no host list to create multiple processes on just one of the cluster nodes.

Note for CUDA first use: ***To use CUDA, you must log into "cscigpu" for which all CSCI 551 students should have an account.***

Note that for problem #3 in this exercise, you are not allowed to use the anti-derivatives of the acceleration function given. Using the anti-derivatives allows you to avoid dealing with the issue of a loop-carried dependency between acceleration, velocity, and position computations, but a major goal of this exercise is dealing with loop-carried dependencies and in the "real world", you may have a data defined function where it is in fact impossible to determine the anti-derivatives!

Exercise #4 Objectives:

1) [**40 points**] Introduction to SPMD parallel programming with CUDA. Make sure you have a "cscigpu" account and work on this code on that system which has an A100 NVIDIA GP-GPU.

a) [15 points] Read Pacheco and Malensek text pages 291-302 and build, run, and test the two versions of CUDA hello programs (found in csci551/code/hello_cuda/). How many threads can print hello using "hello_cuda.cu"? With the use of blocks and threads, how many threads can print hello using "hello_cuda1.cu"? Can you scale bigger using blocks and threads, and if so why?

b) [25 points] Read Pacheco and Malensek text pages 303-320 and build, run, and test the cuda_trap1.cu (found in csci551/code/cuda-integrators/) and compare it to the sequential trap.c code. Verify trapezoidal integration of the known function sin(x) over an interval of 0 to π (should be 2.0) using the supplied starter code (noting carefully how the steps, blocks, and threads must scale). Explain the original command line arguments and how they work. Then modify the program to make it your own and simplify it so you only need to specify steps and have it automatically determine blocks and theads per block. Compare the speed-

up of 1,000,000 steps using sequential trap.c and your modified cuda_trap1.c and provide the parallel percentage.

2) [**20 points**] For this exercise, download the function generation example code (csci551/code/functiongen/). Adapt this code or write your own to interpolate between given data points (Ex-4-Segmented-Function-Train-profile.xlsx) to produce a CSV file (downloads) that you can plot with Excel (or other tool like MATLAB if you wish) that provides the acceleration profile 10 times per second. Overlay the 1 second original data to see if you can see any differences between the plot generated by interpolation and the original data. **Provide a screen shot of your plots and describe differences if any**. Note that you will need to use interpolation (or a math library function generator) for simulation by integration.

3) [**50 points total**] For this problem, **use your favorite method of integration that you created above to integrate a train's acceleration profile – using a single MPI program to solve this problem is most efficient, but this is a significant challenge due to loop-carried dependencies between acceleration, velocity and position for each required integration**. If you have trouble creating just one MPI program, explain why and complete the program by breaking it into 2 separate MPI parallel programs, one that integrates acceleration, and another that uses velocity results to integrate and to find position by using a table from the first program (e.g., ex4accel.h & ex4vel.h). To get a good idea for how this could be done in a single program, which is much more professional, look at hello_cluster/compare.c and notice that it simply passes arrays between ranks as MPI_Send and MPI_Recv messages. The csci551/code/functiongen/ code will also be helpful to learn linear interpolation and a single program OpenMP parallel implementation of a Riemann sum is provided. Using your best method, determine the velocity and position of the high-speed train that runs over a straight and flat frictionless track for 122 Km. You must adhere to constraints provided.

   a) [10 points] Use the acceleration profile provided that adheres to all constraints and integrate it to determine velocity and position over time using dt=0.001 seconds (note that you will need to use a look-up table with linear interpolation or a functional model to get A(t) for any value of t). Report the final position and the steady state peak velocity.
   b) [20 points] Using the acceleration profile, integrate with dt=0.0001 seconds and compare to the first profile by comparing the final position and the steady state velocity and note impact of smaller step size. Time the sequential program for both step sizes and report.
   c) [20 points] Show that your MPI code can scale to 2, 4, 8 or 16 processes on the ECC cluster and plot the completion time for dt=0.0001 as a function of number of processes.

   The operational constraints for the train and estimated resulting velocity and position history are shown in Figure 1, 2, and 3. Here are the key constraints for the train system:

- Maximum speed allowed is 320 Km/hr and the trip must be completed in less 30 minutes.
- The train electric motors can accelerate the train 0.3 meters/sec/sec at most.
- When the braking, decelerate the train no more than -0.3 meters/sec/sec.
- The train must have zero velocity at t=0.0 and zero velocity at t=1800.0
- The train must hit the 122,000 meter distance target with less than 100 meters of error.

Figure 1: Train Acceleration Profile given (Excel)

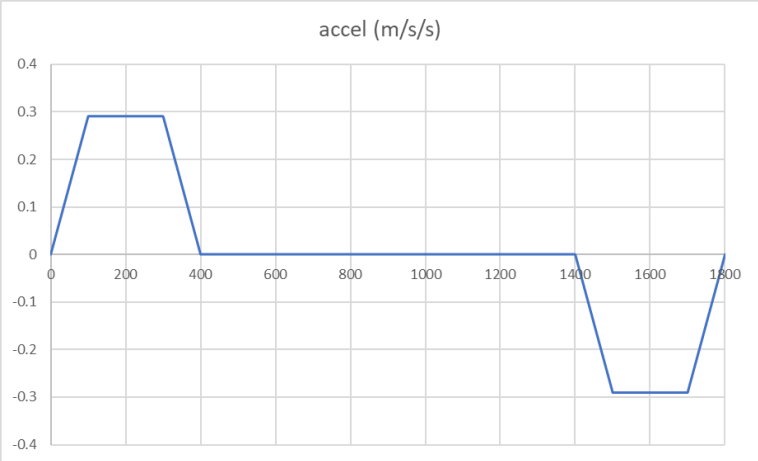


Figure 2: Train Velocity computed from Acceleration Profile

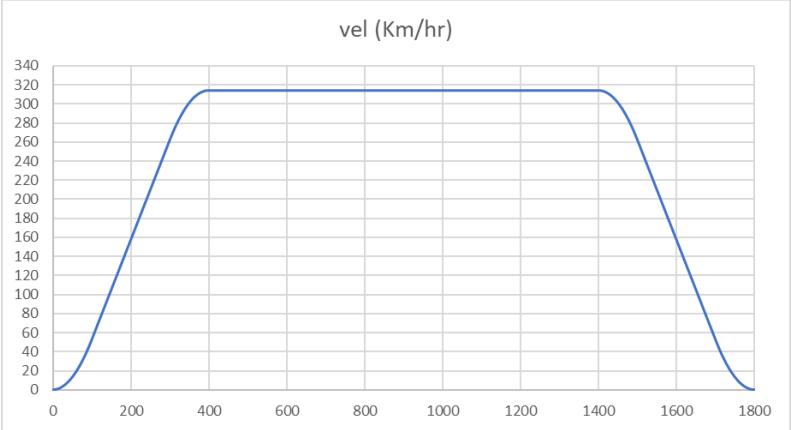
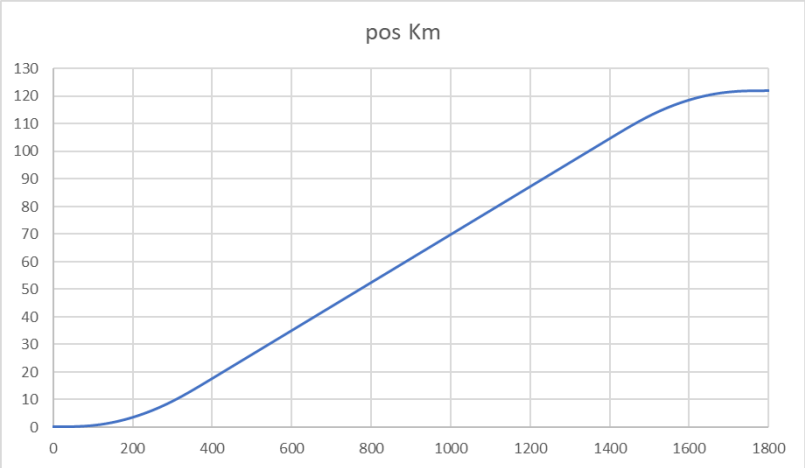


Figure 3: Train Position over time from Velocity Profile



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. 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

Grading Checklist for Rubric

[40 points] Exploring CUDA SPMD Integration:

Problem	Score 0…15	Out of	Comments
Answers to 3 questions asked about cuda_hello code		15	
Explanation of cuda_trap1.cu arguments and how they work		5	
Modified program with comments and simplified arguments		10	
Speed-up and P% comparing cuda_trap1.cu to trap.c		10	
TOTAL		40	

[20 points] Using linear interpolation, compute train function at $1/10^{th}$ second and compare:

Problem	Score 0…10	Out of	Comments
Produce CSV file for $1/10^{th}$ second for 1800 seconds and plot		10	
Describe any differences noted between original file and new?		10	
TOTAL		20	

[40 points] Finding the best integration method to make train simulation parallel with MPI:

Problem	Score 0…10	Out of	Comments
Sequential program with dt=0.001		10	
Problem	Score 0…20		Comments

Sequential program with dt=0.0001 with run time analysis		20	
SPMD MPI program and scaling		20	
TOTAL		50	