CSCI 551 – Numerical and Parallel Programming: Exercise #3

MPI and Numerical Integration of Smooth Functions

(Model inspiration: Beijing to Tianjin Intercity Rail) – Story on High Speed Rail

DUE: As indicated on Canvas

Please thoroughly read Week-6 and Week-7 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 to do reading, come to office hours, etc.

You will need an ECC system (***ecc-linux.csuchico.edu***) 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). If you have issues with Global Protect VPN on your system, seek help from IT Support. If this can't be resolved please use **Wildcat Desktop**!

Note that you can download the starter code from the class web-site 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 Pacheco book MPI companion materials useful as well as use of the online Kaw and Kalu numerical reference book for numerical methods. For MPI analysis and design work, you will find the **starter code hello_cluster 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 ECC server (ecc-linux) 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 by boxing them). 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.

Exercise #3 Objectives:

For the following four problems, **use the following sine curve acceleration profile**. **You should write your code using SPMD MPI method of integration using collective communication functions such as MPI_Reduce and segmentation of the overall functions with time based upon comm_sz, for MPI scaling. Please be sure to test your MPI program for 1 worker and up to S # of workers.**

Test your code for one process and for multiple processes to make sure you get the similar results for each problem for each level of precision and method of integration. Using the required method of integration and floating-point precision for each problem, compute the final position of the train for the given acceleration profile. If you see differences between the final positions, comment on which method is the most accurate and precise and what you believe contributes to error (inaccuracy for each). Note that the antiderivative of sin(x) is -cos(x)+C. You can generate the acceleration function and the velocity function from the known definite integral solutions as well as numerically for both acceleration and velocity. The time and amplitude have been scaled, so take a careful look at the Ex-3-Smooth-Function-Train-profile.xlsx spreadsheet (download) for guidance and this more in-depth error analysis. It is also possible to use interpolation, but the goal here is to explore accuracy and precision, so math library functions are simpler - use Wolfram Alpha or similar math tools can be used to check work. ***Note that for this exercise, you are not only allowed to use the anti-derivatives of the acceleration and velocity function given, but in fact encouraged to use them, but you must numerically integrate each. Using the anti-derivatives allows you to avoid dealing with the issue of a loop-carried dependency between acceleration, velocity, and position computations. In Exercise #4, you are not allowed to use anti-derivatives.***

1) [40 points total] For this exercise, you must **implement and compare the Riemann and the Trapezoidal numerical integration methods below as sequential programs and test the well –** use a fixed step size of dx=0.001 and compare accuracy of the methods using "double" precision for all calculations and flexible "function to ingrate" capability.

   Use methods below to compare accuracy for integration of sin(x) numerically over an interval with known area based on calculus (note that hello_integrators may help).

   a) Riemann sum (Left, Right, or Mid-point can be used as you wish)
   b) Trapezoidal sum (2nd method you must implement and test)

[10 points] **Show that your 2 methods work by integrating a simple CONSTANT function such as f(x)=10.0 over the interval 0 to 10.0 using a step size small enough to compute the expected result of 100.0**, with 7 digits of precision or more if possible.

[10 points] **Show that your 2 methods work by integrating a simple function such as f(x)=sin(x) over the interval 0 to Π using a step size small enough to compute the expected result of 2.0**, the diameter of a circle, with 7 digits of precision or more if possible.

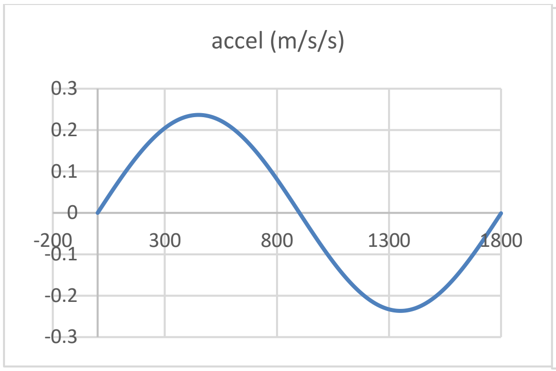
[20 points] Convert the most accurate integrator from those you tested into an ***MPI implementation that breaks the problem down into area under a curve in 4 or more segments and re-test it by integrating f(x)=10.0 over the interval 0 to 100000.0 to verify the expected result of 1000000.00***. You should design your program so that sub-intervals are added together at the end of your program to speed up overall integration time. Describe which method gets you the best answer and why?

2) [20 points total] Train problem with sine curve acceleration profile – use a **Left Riemann** sum with "**float**" and step size of **1/100th of second (10 milliseconds)** to integrate until the train comes to a stop and **report the final distance traveled and final velocity**. Use the math.h implementation of "sine" to generate your acceleration values as function of time. Note that for the train problems instead of integrating any function f(x), you are integrating the specific f(t) scaled sine functions fo the Ex #3 smoot train profile. You will find the following sequential integration examples (csci551/code/hello_integrators/) and the following MPI integration examples MPI_Examples/mpi_trap4.c and MPI_Examples/mpi_trap3.c useful as references of working MPI programs that are similar to what you need to implement and test. Compare your numerically integrated results with expected symbolic solution (Ex-3-Smooth-Function-Train-profile.xlsx) for velocity and position. What are the sources of inaccuracy errors in this formulation and are there precision and computational error issues using float (7 digits of precision)?

3) [20 points total] Train problem with **scaled** sine curve acceleration profile – use a **Mid-point Riemann** sum with "**double**" and step size of **1/1000th of second (1 millisecond)** to integrate until the train comes to a stop and **report the final distance traveled and final velocity**. Use the math.h implementation of "sine" to generate your acceleration values as function of time. Compare your numerically integrated results with expected symbolic solution for velocity and position. What are the sources of error in this formulation and are there precision issues using double (15 digits of precision)?

4) [20 points total] Train problem with **scaled** sine curve acceleration profile – use **Trapezoidal sum** with "**double**" and step size of **1/1000th of second (1 millisecond)** to integrate until the train comes to a stop and **report the final distance traveled and final velocity**. Use the math.h implementation of "sine" to generate your acceleration values as function of time. Compare your numerically integrated results with expected symbolic solution for velocity and position. What are the sources of error in this formulation and are there precision issues using double (15 digits of precision)?

Figure 1: High Speed Train Acceleration and 1 second step Velocity Profile



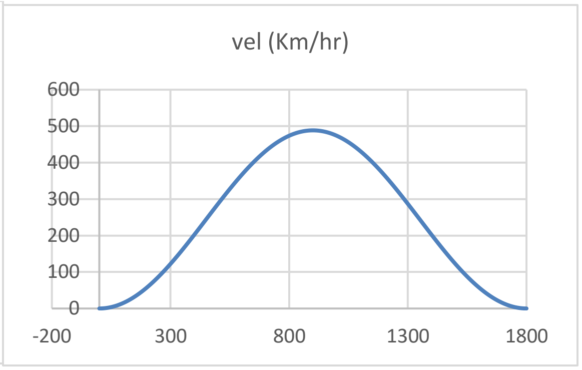
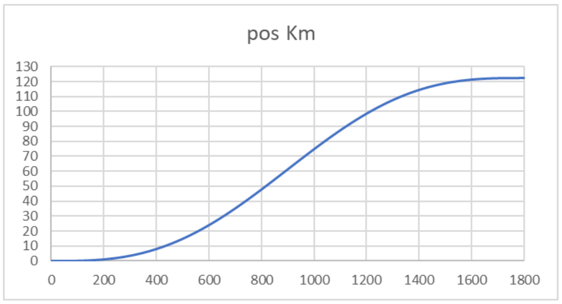


Figure 3: High Speed Train Position from Velocity Profile (Ideally 122 Km travel distance)



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

<u>Grading Checklist for Rubric</u>

[40 points] Code for **Left Riemann & Trapezoidal f(x) integration** with double precision:

Problem	Possible	Score	Comments
Riemann Sequential Code computes **f(x)=10.0 over the interval 0 to 10.0**	5		
Trapezoidal Sequential Code computes **f(x)=10.0 over the interval 0 to 10.0**	5		
Riemann Sequential Code computes **f(x)=sin(x) over the interval 0 to Π**	5		
Trapezoidal Sequential Code computes **f(x)=sin(x) over the interval 0 to Π**	5		
Best MPI parallel method of integration computes ***f(x)=100.0 over the interval 0 to 100000.0 to verify the expected result of 10000000.00***.	20		
TOTAL	20		

[20 points] SPMD MPI code for **Left Riemann "Train" integration** with single precision:

Problem	Possible	Score	Comments
MPI Code divides work up based upon comm_sz and scales well	5		
MPI Code uses **Riemann** with **float** for required step sizes	5		
Program produces credible final position	5		
Report provides **analysis and comments on accuracy and precision**	5		
TOTAL	20		

[20 points] SPMD MPI code for **Midpoint Riemann "Train" integration** with double precision:

Problem	Possible	Score	Comments
MPI Code divides work up based upon comm_sz and scales well	5		
MPI Code uses **Riemann** with **double** for required step sizes	5		
Program produces credible final position	5		
Report provides **analysis and comments on accuracy and precision**	5		
TOTAL	20		

[20 points] SPMD MPI code for **Trapezoidal "Train" integration** with double precision:

Problem	Possible	Score	Comments
MPI Code divides work up based upon comm_sz and scales well	5		
MPI Code uses **Trapezoidal** with **double** for required step sizes	5		
Program produces credible final position	5		
Report provides **analysis and comments on accuracy and precision**	5		
TOTAL	20		