Homework #2

ME 471/571

All of your work for this assignment should be turned in using Jupyter Notebooks. Please submit the an HTML version as well as the original notebook.

1. (**Trapezoidal rule - using MPI**) Approximate the following definite integral using the Trapezoidal Rule. Your code should run on 1,2,4, and 8 processors.

$$I = \int_{-1}^{1} (x-1)^2 e^{-x^2} \, dx$$

   (a) Use Wolfram Alpha (`www.wolframalpha.com`) to find the exact expression for definite integral above. The correct expression will involve e, π, and the function `erf(x)`.

   (b) Verify that your code is correct by running it on a sequence of intervals $N = 2^p$, for $p = 10, 11, 12, \ldots, 28$. Verify that your results are correct by showing numerical convergence, and show performance results by showing plots of strong and weak scaling and efficiency. You may use the notebooks on the course website to get started with this.

   Your code should be written in C and use the MPI functions we have learned in class including `MPI_Bcast` to broadcast details of the domain, i.e. domain endpoints and N, and `MPI_Reduce` to collect the sums from each processor.

2. (**Integral equation**) You have probably learned that a standard approach to solving the second order differential equation

$$u''(x) = f(x) \tag{1}$$

   with appropriate boundary conditions for a given function $f(x)$ is to subdivide the domain into N intervals of width h, and then, using a finite difference discretization, construct and solve a linear system $A\mathbf{u} = \mathbf{f}$. The advantage of this approach is that the matrix A is tridiagonal matrix and so the linear system can solved very efficiently (using, for example, the *Thomas* Algorithm). The chief disadvantage of the finite difference approach is that ill-conditioning in the matrix A eventually leads to round-off error which swamps the truncation error in the discretization, making the solution unusable for very large N. Also, it isn't obvious how to parallelize Gaussian elimination.

   Below, we will use a different approach to solving the above differential equation which parallelizes in an obvious way, and which solves the round-off error problem.

   We avoid the ill-conditioning by using an explicit formula for the solution to (1). Over the interval $[0, 1]$, the exact solution to (1) can be written in terms of two integrals as

$$u(x) = \left(a - \int_0^x \xi f(\xi) \, d\xi\right)(1-x) + \left(b + \int_x^1 (\xi - 1) f(\xi) \, d\xi\right) x \tag{2}$$

   where $u(0) = a$ and $u(1) = b$ are the prescribed boundary conditions. This method also parallelizes in an obvious way, as we saw in Problem 1.

   For this problem, Use the trapezoidal rule to numerically evaluate (2) on a subdivided mesh with $N = 2^p$ subintervals. Choose a range of p large enough to see good scaling results. Your code should run on 1,2,4,8 and 16 processors. Set $f(x) = -(2\pi)^2 \sin(2\pi x)$ and $a = b = 0$.

   (a) (**Check the solution**) Differentiate (2) twice to convince yourself that (2) is an exact solution to (1). You do not need to turn this in, but you should be convinced before you move to the next problem.

(b) (**Verify.**) For a choice of N, plot solution you get evaluating (2) using the Trapezoidal method along with an exact solution on the same graph. Plot the error for several values of N to show that your solution is converging to the true solution.

(c) (**Scaling results**) Show weak scaling, strong scaling and efficiency results for this approach to solving the second order differential equation.

(d) (**Simpson's Rule**) Develop a fourth order solver by using Simpson's rule to evaluate the integral equation. How does the scaling for the higher order integral compare to the lower order method?

(e) (**Difference matrix.**) Compare the error that you get from solving the integral equation with the error you get from solving the $A\mathbf{x} = \mathbf{b}$ problem. You should see that the error for the integral solution decreases even for very large N, unlike the solution resulting from inverting the matrix A.

Note : This is not a parallel question, but more a question about numerics.

(f) (**Discussion.**) Please discuss what you see as the potential pros and cons of using this integral approach for high performance computing as opposed to inverting the matrix A. What are the parallel implications of solving the integral equation in parallel?