EECS 587 Homework: Due at the start of class 5 Oct 2021

> This program is to be written on your own, not in collaboration. However, feel free to talk with others about MPI, problems with using Great Lakes, etc. *Warning: do not put this off until the last minute, and debug using small matrices.*

Using the Advanced Research Computing Great Lakes computer, run an efficient MPI program to use p cores to do a stencil calculation on an $n \times n$ matrix $A(0:n-1, 0:n-1)$ where the entries are long integers (i.e., 64 bits). Do this for $n = 1000$ and $n = 4000$ for p = 4, 16, and 36. The same program must be used for all of the runs, and should work no matter what the entries of A are nor what the function f is (described below), and no matter what $p, n > 1$ are, for p a perfect square such that $p \leq n/2$ (this latter condition is just to insure that you won't have any of the small matrix problems discussed in the handout).

When you submit your program to Great Lakes you must put a time limit of 5 minutes or less, though we might adjust this limit if necessary. Failure to put a time limit of 5 minutes or less may result in losing points on the assignment. You have to use Great Lakes for your timing runs but you can debug on any machine (including your own).

Procedure: For each combination of n and p:

1. Initialize A using the definition below.
2. Use MPI_BARRIER, then have the root process (process 0) call MPI_WTIME.
3. Do 10 iterations, defined below.
4. Compute the verification values (see below) and send them to the root process.
5. Once the root process has all the verification values, have it call MPI_WTIME.
6. Print out the elapsed time (the time between the first and second calls to MPI_WTIME) and the verification values.

Turn in the program code, the timing and verification outputs, and the report.

To initialize A: for all $0 \leq i, j \leq n-1$, $A(i, j) = i + j * n$
Each entry of A can start on whatever processor you choose, but no entry of A can start on more than one processor.

Iterative step: In each iteration, let A_o denote the previous value of A. Then for all $0 \leq i, j \leq n-1$, the new value of $A(i, j)$ is given by:

- $A(i, j) = A_o(i, j)$ if $(i = 0) \vee (i = n-1) \vee (j = 0) \vee (j = n-1)$, i.e., it is unchanged along the border of the matrix

- Otherwise, $A(i,j) = f(A_o(i,j), A_o(i+1,j), A_o(i,j+1), A_o(i+1,j+1))$, where f is a function that we will give you.

Be careful that you use A_o, not values from the current iteration. You are not allowed to make any modifications to the code we give you for the function f.

To verify a run: Print the sum of all of the entries of A after the final iteration, and the number of final entries that are equal to the final value of $A(\lfloor n/3 \rfloor, \lfloor 2n/3 \rfloor)$. Note that this count is at least 1.

The report: Turn in a short report which includes the verification values and a brief description of how you decomposed the matrix, how you did communication, and your timing results. Analyze whether the timing represents perfect speedup and scaling, and, if not, why the program not achieving it (or, if you have superlinear scaling, why that occurred). Note that you have scaling in terms of the size of the matrix, as well as in the number of processors. The report needs to be typewritten, though you can do drawings by hand. You will be graded on correctness, the quality of your report, and achieved performance. Since performance is important, you should make sure you do serial and parallel optimizations discussed in class. For example, you should turn on the compiler's optimization options.

Depending on how long the programs are taking and the turnaround time on Great Lakes, we may adjust the matrix sizes, number of iterations, or details of f.