Scientific Computing 502b
Assignment3: Problems on a 2D Mesh
Due: 23 December 2022 (or Jan 3rd)

In this assignment we will solve a problem in 2 spatial dimensions, or one that can posed in an analogous way. It is somewhat up to you which problem to solve, but it would probably be a good idea to run it by me to ensure it is realistic. You do not have to start from scratch. You can start with a *serial* code that you obtain from elsewhere (as long as it is either open-source, or you have legal permission to use the code). Some potential problems you could solve are:

a) You can solve the 2D Poisson Equation directly if you like, but should use the conjugate gradient method instead of Jacobi iteration.
b) You could solve a Poisson-like equation. Examples include things like Stokes-flow, etc.
c) Some other PDE where you have time and two spatial dimensions. In this case the problem can usually be modelled using finite-differences. Explicit models are easiest to parallize, implicit models are possible but discuss this with me first. Examples include the diffusion equation, Schrödinger's equation (which can be modelled as an imaginary time diffusion problem). Note that nonlinear effects are fairly easy to include in such models.
d) Lattice-Boltzmann models for fluid flow are also fairly easy to parallize using the techniques we have covered so far.
e) You do not have to solve a PDE, there are other models you can study on a grid, for example, the Forest-fire model (http://en.wikipedia.org/wiki/Forest-fire_model), or other Cellular automata (http://en.wikipedia.org/wiki/Cellular_automaton).
f) There are a number of numerical linear algebra problems that can be parallized using block algorithms allowing a parallization scheme similar to what we have done in class. If you have one in mind, let me know and we can decide if it would be suitable.

1. Parallize your problem using a 1D grid of processors similar to how we first parallized the Poisson equation solver.
2. Modify what you did in (1) to use a 2d grid of processors. You should use MPI_Type_vector and the MPI_Cart routines, as discussed in the lecture, to handle the decomposition. If you parallize your own problem and a 2d grid doesn't really make sense, then you should make use of a library (like lapack, fftw, or something like that) to solve the local part of the problem (discuss with me to make sure it is reasonable).
3. Time your code for 1D and 2D parallel decomposition for different numbers of processors and sizes of systems (mesh size). This will need to be done on a machine with at least 16 processors, so on a Compute-Canada cluster or on other machines in the department if you do not otherwise have access to such a machine. Time calculation and communication times separately as well as the total run time. Plot speedup, efficiency, and ratio of communication to calculation time as a function of n (nxn grid) and p (number of processors). Try to optimize the test size of the system so that it does not take too long to run (e.g. max 16 minutes on 1 processor and then you could go up to 16 processors, say in the test queue) but still large enough that parallelization makes sense (e.g. not much less than 16 minutes on 1 processor). For small n you will typically find that the program actually slows down when you add more processors. At what n do you break-even (i.e. how big does the system have to be to benefit from parallization, note that this may depend on p in that if the problem gets too small on each processor you become dominated by communication.). Make sure you have n significantly bigger than this before you run on to

many processors. Do you ever see “super-linear” speedup? Is 1D or 2D decomposition faster? Any idea why?

What to hand in: Same format as in previous assignments.