CSE 6220 Introduction to High Performance Computing
Spring 2014
Programming Assignment 2
Due Friday, April 25

The goal of this programming assignment is to learn to use several advanced programming features of MPI − virtual grid topology, creating communicators for subgrids, persistent communication, non-blocking communication to overlap computation and communication, and derived data types. All these features must be used to receive full points on this programming assignment. Your programs must also reflect a good style of programming and well-commented in order to receive full points.

Write a parallel program to play Conway's game of life in parallel. The game takes place in a grid (or board) of cells (or squares). Each cell can be empty or occupied. The occupied cells change from one iteration (or generation) to the next. To go from one generation to the next, each cell in the grid is examined to see if it will be occupied or not in the next generation. This determination is made according to the following rules.

- If an occupied cell has less than 2 neighbors, it will become empty.
- If an occupied cell has more than 3 neighbors, it will become empty.
- If an empty cell has exactly 3 neighbors, it will become occupied.

The neighbors of a cell are the 8 cells adjacent to it along the directions N, NW, W, SW, S, SE, E, NE. Cells along an edge or at a corner have fewer neighbors.

Your program should take input from a file in the following format: The first line consists of two integers m and n specifying the size of the grid ($m \times n$ grid). The second contains a single integer specifying the number of generations. This is followed by an initial description of the status of each grid cell (empty/occupied) given one row of the grid per line. An empty cell is indicated with a '0' and an occupied cell is indicated with a '1'. Consecutive entries in a row are separated by a single space. The program should run the specified number of generations and print the resulting grid in the same format as the input grid. You are not allowed to make any assumptions on m and n. However, you can assume that the number of processors is either a perfect square ($4, 9, 16, 25, 36$ etc.) or a power of two ($2, 4, 8, 16, 32$ etc.). If the number of processors is not a perfect square, the virtual grid topology you create should be as close to a square as possible (Ex: $32 = 8 \times 4$).

The grid of life is partitioned on to the virtual grid of processors such that each processor is responsible for a subgrid of the grid of life. The subgrids assigned to processors must be as close in size as possible. Notice that the computational work involved in each generation is proportional to the area of the subgrid assigned to a processor. Communication is needed to exchange the perimeter of the subgrid with neighboring processors. For simplicity, you may make the input and output parts of the program sequential. i.e., it is allowed that one processor reads the input file and sends the required information to all the processors. Similarly, at the end of execution, it is

permissible to have all processors send their subgrids to one processor which then prints the final grid of life.

Your program must have the following features:

1. Use MPI-functions to create a virtual grid topology on the set of processors.

2. Create communicators for each row and each column and use them for communicating the edges of the subgrid.

3. Use persistent communication because the same arguments are used in communication in each generation. This part uses MPI functionality that is not taught in class, and you are expected to learn this on your own. The persistent communication functions that you will need include MPI_Send_init, MPI_Recv_init, MPI_Start, and MPI_Request_free. You can study how these functions work using material available at the following URL:
   `http://www.mcs.anl.gov/research/projects/mpi/www/www3/`

4. Use non-blocking communication to schedule requests for communication, work on internal part of the subgrid (everything excluding the edges), wait to finish the communication, and then work on updating the edges. This way, communication can be overlapped with computation to the maximum extent possible. For this part, you should use MPI_Isend, MPI_Irecv, and MPI_Wait.

5. Use derived datatypes to communicate the leftmost and rightmost columns of the subgrid of life assigned to a processor. Note that derived datatypes are not needed for top and bottom rows because those elements are stored contiguously in C/C++.

Also, create a sequential program to run the game of life. This will not take additional effort because the part of the parallel program where a subgrid of life is updated on a processor is identical to the sequential program. The sequential program is used to measure how well we are doing in parallel.

In addition to the program, turn in the following:

1. Speedup as a function of the number of processors for a fixed problem size for a single generation.

2. Speedup for a fixed number of processors as the problem size is varied, again for a single generation.

It is important that you strictly adhere to the input and output formats described in the programming assignment so that we can run your program on test files and verify output. Your executable should take as command line argument the input file name followed by the output file name.

Submit a zip/tar file containing the following

1. Text file containing names of team members and their percentage contributions.

2. Source code with instructions to compile.

3. Report in pdf format. Your report should include:

   (a) Short design description of your program.

   (b) State all computation and communication steps, and memory allocations that are proportional to the problem size.

   (c) Runtime plots by varying problem size and number of processors (keeping one fixed while varying others).

   (d) Observations on how your algorithm is behaving on varying parameters mentioned in previous point.