

Programming Assignment #2

Due November 12, 2009

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, although we will give a majority of points for a functioning implementation. We will look at the code for this assignment so your program should be well-commented.

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 the following as input from a file: Two integers m and n specifying the size of the grid ($m \times n$ grid), an integer specifying the number of generations, 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. The program should run the specified number of generations and print the resulting grid. You are not allowed to make any assumptions on m and n . You can assume, however, 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, make the input and output parts of the program sequential. i.e., one processor reads the input file and sends the required information to all the processors. Similarly, at the end of the running, 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.
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.
5. Use derived datatypes to communicate the leftmost and rightmost columns of the subgrid of life assigned to a processor.

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.

To submit the programming assignment, turn in the following:

1. A printed copy of the program.
2. Present an analysis that derives the following:
 - (a) sequential run-time
 - (b) parallel run-time showing the computation and communication times separately
 - (c) an analysis of the scaling of the algorithm, i.e., keeping the efficiency fixed, how can we scale the number of processors as the problem size increases? Can we scale the number of processors such that the parallel run-time remains the same as the size of the problem is increased? Please be brief in answering these questions.
3. Fix the number of processors to 16. Plot the speedup of a single generation as a function of the problem size.

I will set up individual appointments with you to make sure that your program is running correctly and to discuss any problems. You should be able to demonstrate the running of your program on test files that I create and your program should generate the correct answer. As such, it is important that you strictly adhere to the input format described in the programming assignment.