MPI: Project - Game of Life

"Game of Life" simulation by John Conway

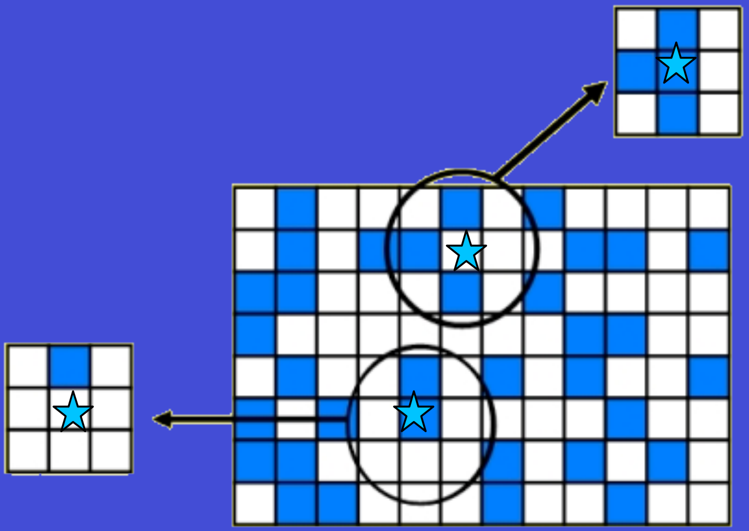
2D array of cells.

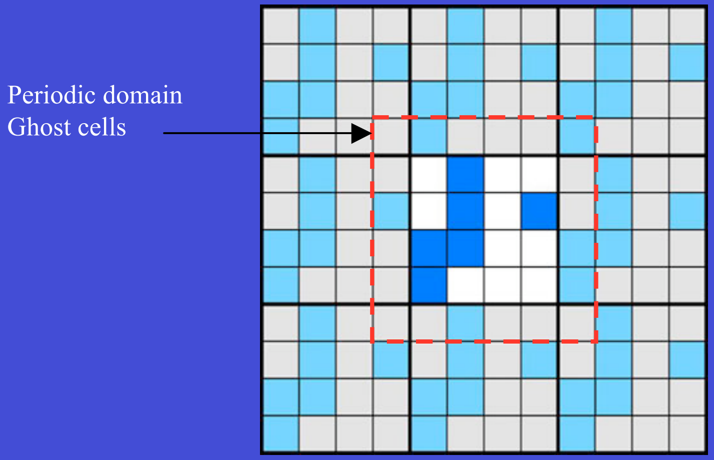
Each cell can have one of two possible states: alive or dead

Initialise with random states and then evolve according to rules:

- If 3 neighbours are alive, cell will be alive (if already alive, remains alive; if was dead, becomes alive)
- If 2 neighbours are alive, no change in cell status.
- All other cases, cell is dead (if was dead, remains dead; if was alive, becomes dead).

Assume periodic boundary conditions (don't forget you need diagonals)





MPI: Project - Game of Life

I suggest you proceed with the following steps:

1. Write an outline of the steps needed to compute the game
2. Write a serial version of the game. To debug, use a small grid (e.g. 4x4) and a single step. You will need this code to check the correctness of your parallel version and to compare speeds. Keep a track (print out) of the total number of alive cells -- this can be your simple metric for whether things are working or not. I suggest you use ghost cells for the boundary conditions.
3. Add the MPI initialisation and finalisation routines to your code and call it a parallel version. Doing this makes a code that runs the same program redundantly on all the processors you specify. Print the rank and the metric to make sure all is working.
4. Do a domain decomposition of the problem to parallelise the problem truly. If you are programming in C, I recommend dividing by rows. If you are programming in Fortran, I recommend dividing by columns. This is because of the way arrays are contiguous in memory in the different languages (e.g. column major in Fortran). Use ghost cells for interior boundaries. Debug with a partition of 2.
5. Mega version: Divide the domain the OTHER way. This means you will need to use derived data types, since these are not the natural ways for the language.
6. Giga version: Decompose the problem in both directions.
7. Tera version: Use a virtual topology. Use MPI_CART_CREATE, and use MPI_CART_SHIFT to create the ghost cells.
8. Peta version: Use parallel I/O to write out the solution grid and visualise it.