Mlife2d – a simple, parallel, implementation of Conway’s “Game of Life”

wgropp.cs.illinois.edu/advmpi-17.tgz

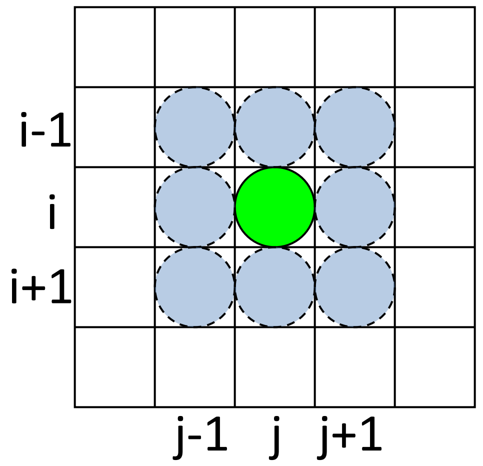
Conway’s Game of Life

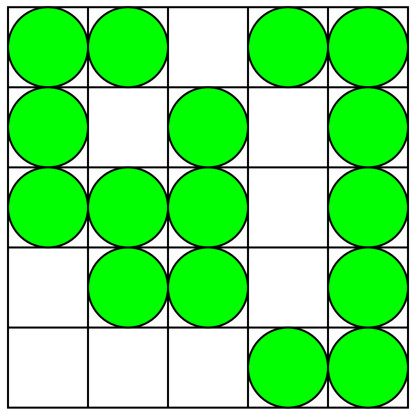
- In this tutorial, we use Conway’s Game of Life as a simple example to illustrate the program issues common to many codes that use regular meshes, such as PDE solvers
  - Allows us to concentrate on the MPI issues

- Game of Life is a cellular automaton
  - Described in 1970 Scientific American
  - Many interesting behaviors; see:
    - http://www.ibiblio.org/lifepatterns/october1970.html

Rules for Life

- Matrix values A(i,j) initialized to 1 (live) or 0 (dead)
- In each iteration, A(i,j) is set to
  - 1 (live) if either
    - the sum of the values of its 8 neighbors is 3, or
    - the value was already 1 and the sum of its 8 neighbors is 2 or 3
  - 0 (dead) otherwise





Implementing Life

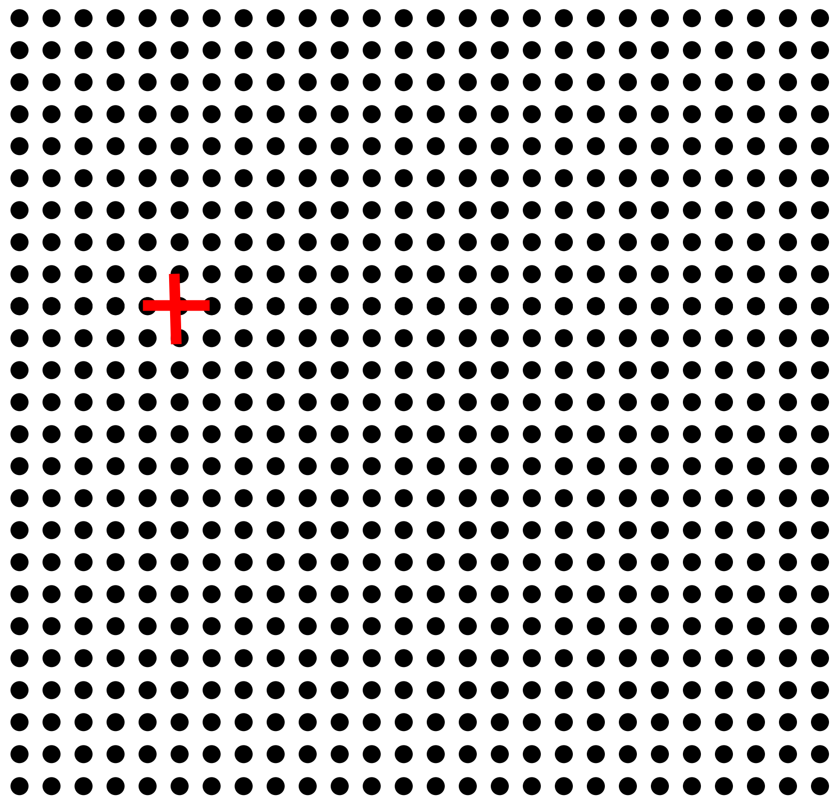
- For the non-parallel version, we:
  - Allocate a 2D matrix to hold state
    - Actually two matrices, and we will swap them between steps
  - Initialize the matrix
    - Force boundaries to be “dead”
    - Randomly generate states inside
  - At each time step:
    - Calculate each new cell state based on previous cell states (including neighbors)
    - Store new states in second matrix
    - Swap new and old matrices

Steps in Designing the Parallel Version

- Start with the “global” array as the main object
  - Natural for output – result we’re computing
- Describe decomposition in terms of global array
- Describe communication of data, still in terms of the global array
- Define the “local” arrays and the communication between them by referring to the global array

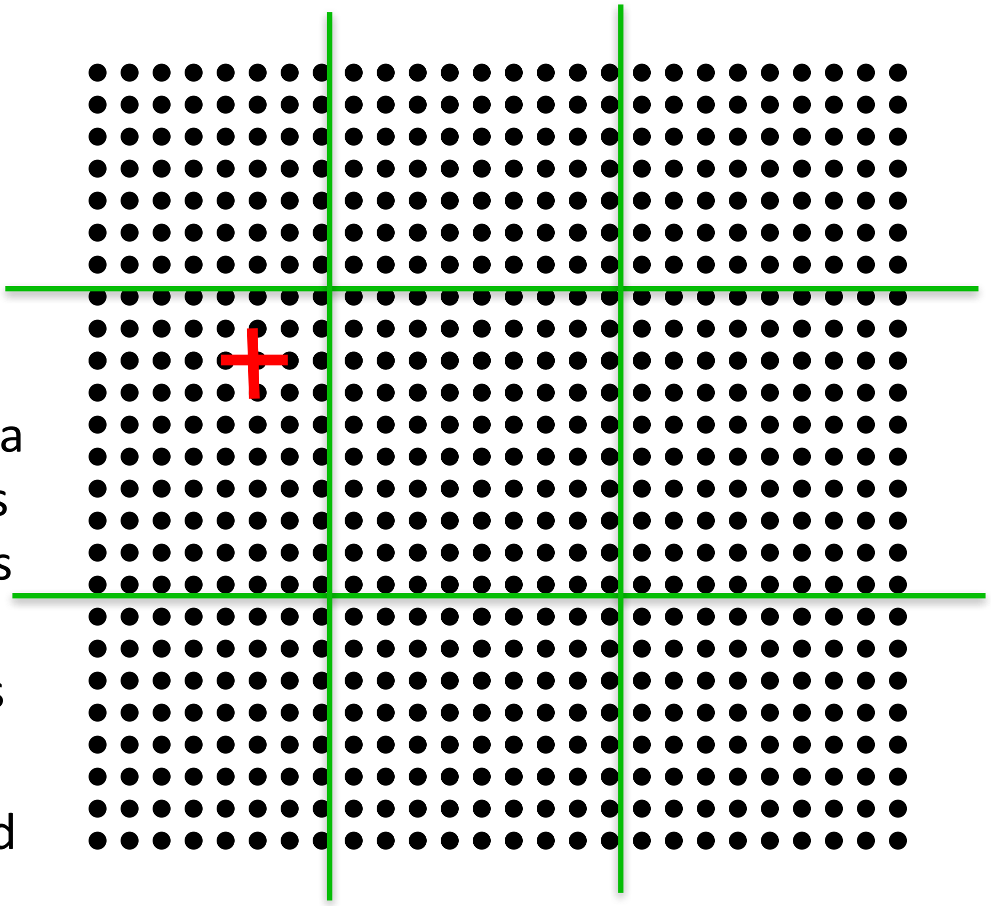
The Global Data Structure

- Each circle is a mesh point
- Difference equation evaluated at each point involves the four neighbors
- The red “plus” is called the method’s stencil
- Good numerical algorithms form a matrix equation Au=f; solving this requires computing Bv, where B is a matrix derived from A. These evaluations involve computations with the neighbors on the mesh.

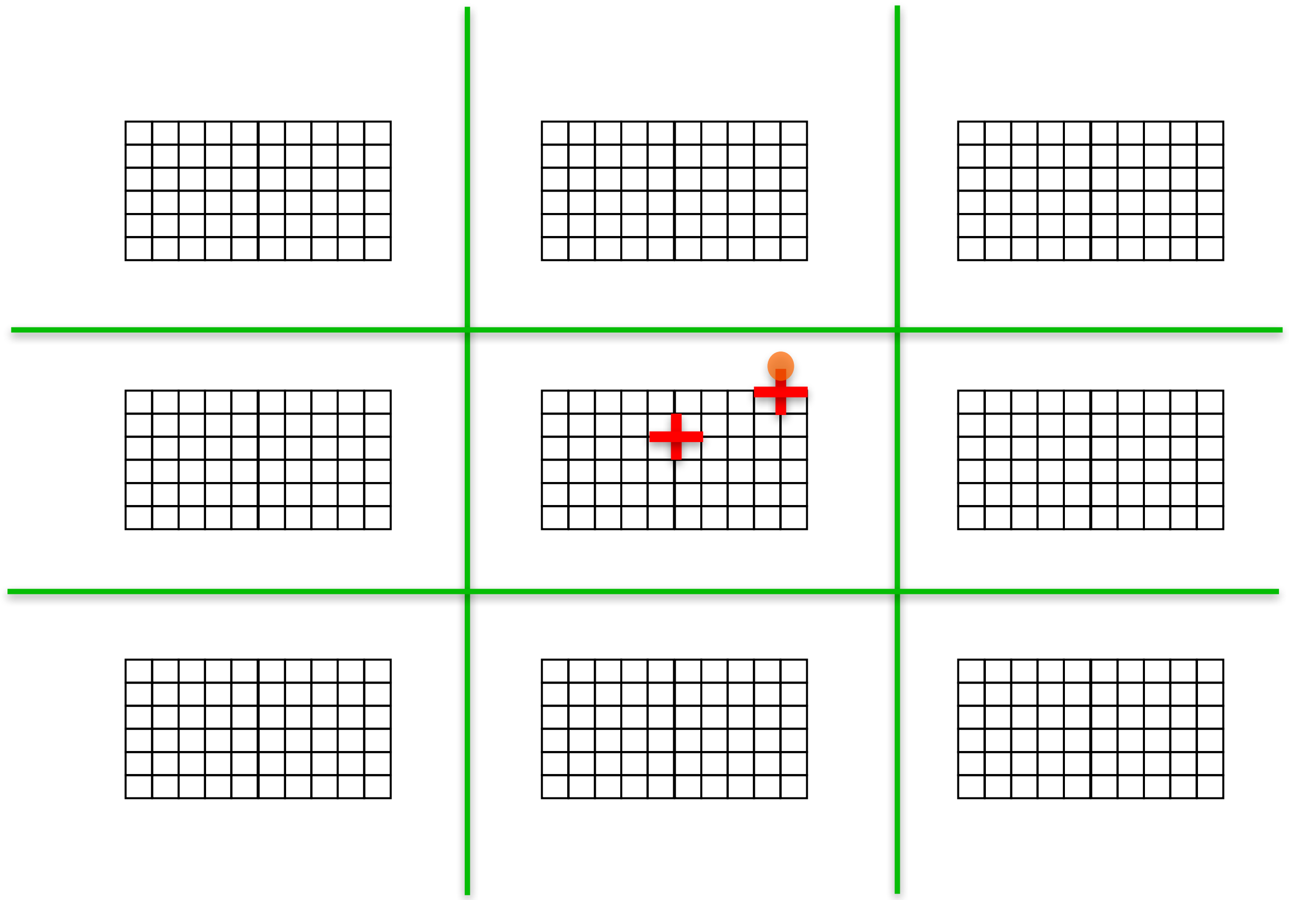


The Global Data Structure

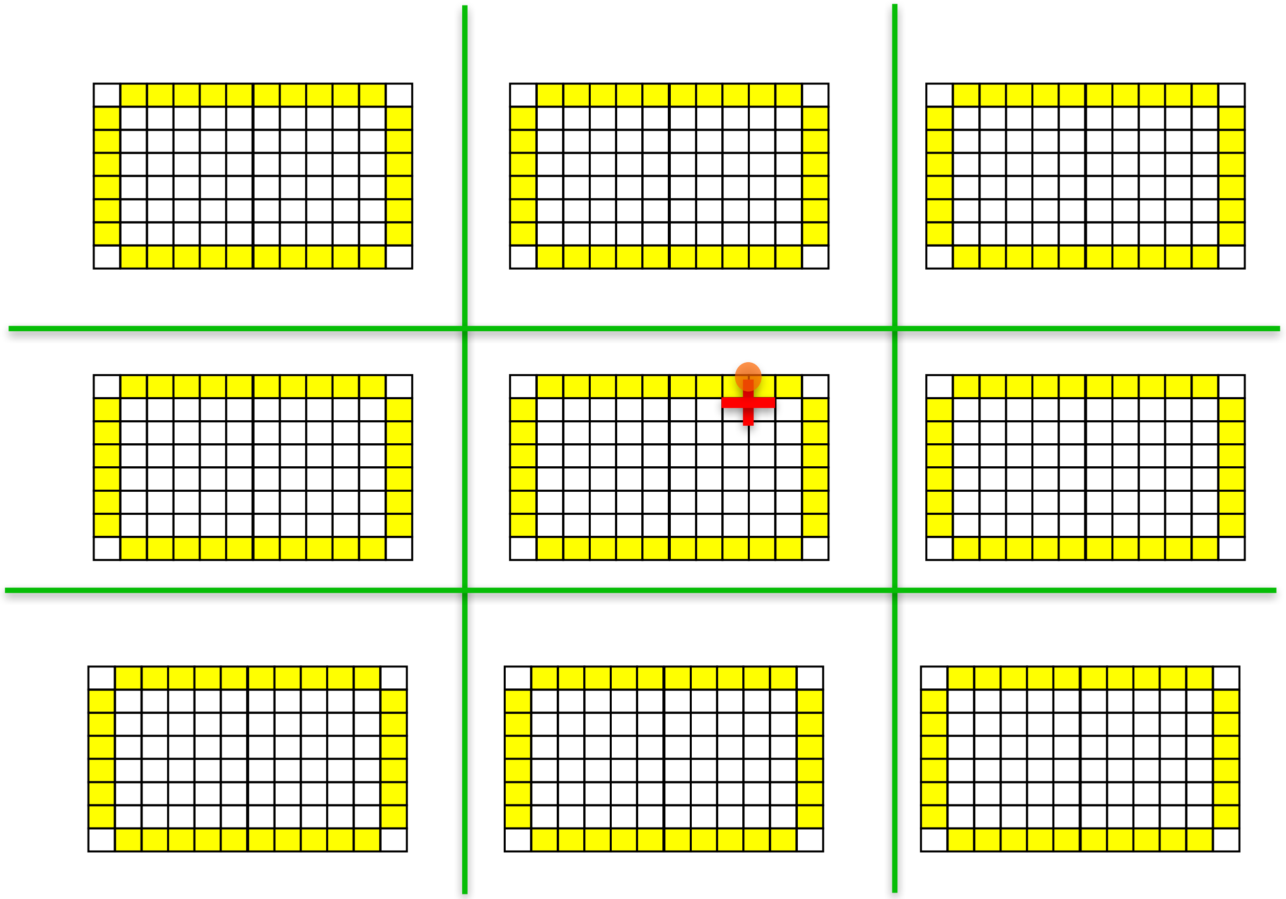
- Each circle is a mesh point
- Difference equation evaluated at each point involves the four neighbors
- The red “plus” is called the method’s stencil
- Good numerical algorithms form a matrix equation Au=f; solving this requires computing Bv, where B is a matrix derived from A. These evaluations involve computations with the neighbors on the mesh.
- Decompose mesh into equal sized (work) pieces



Necessary Data Transfers

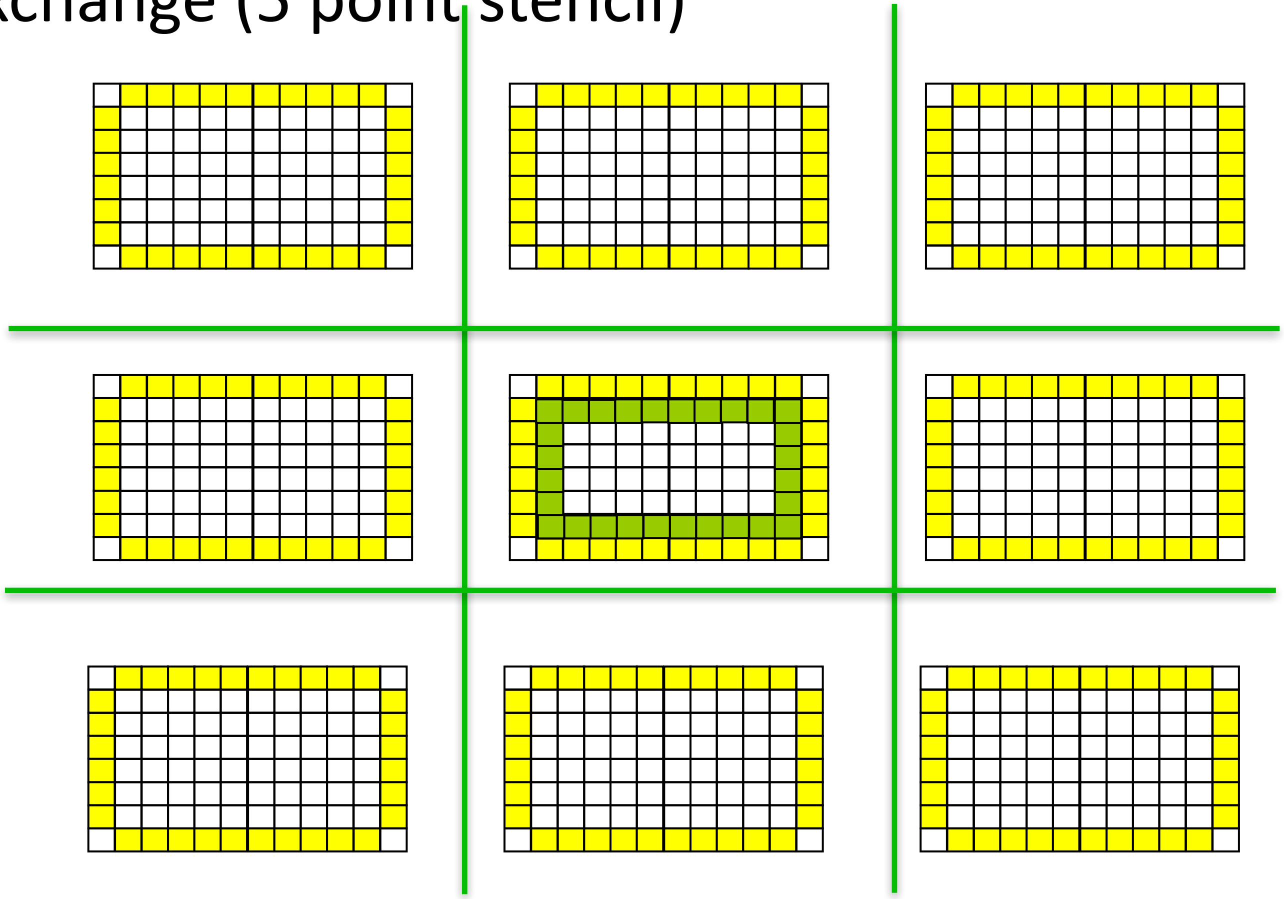


Necessary Data Transfers



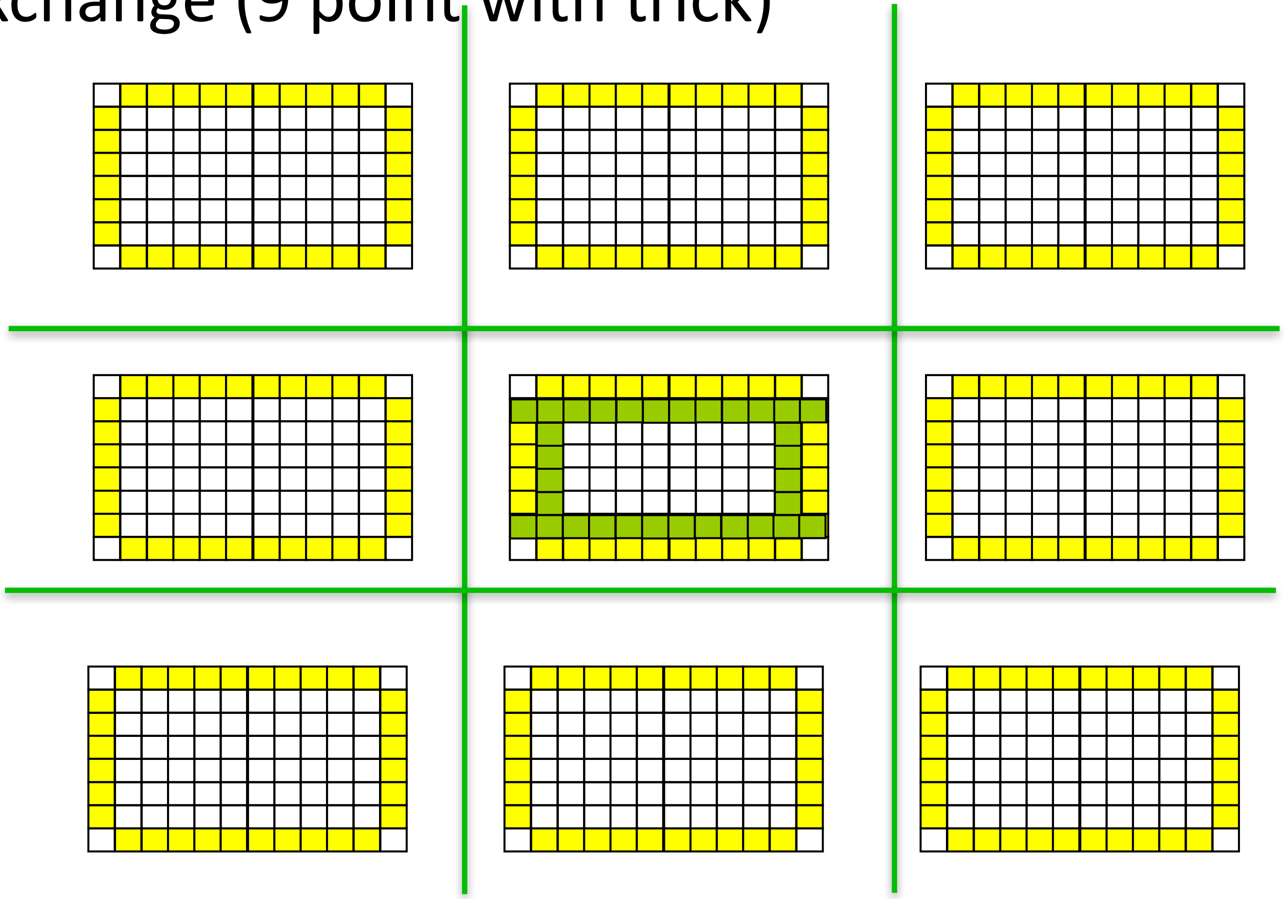
Necessary Data Transfers

- Provide access to remote data through a *halo* exchange (5 point stencil)



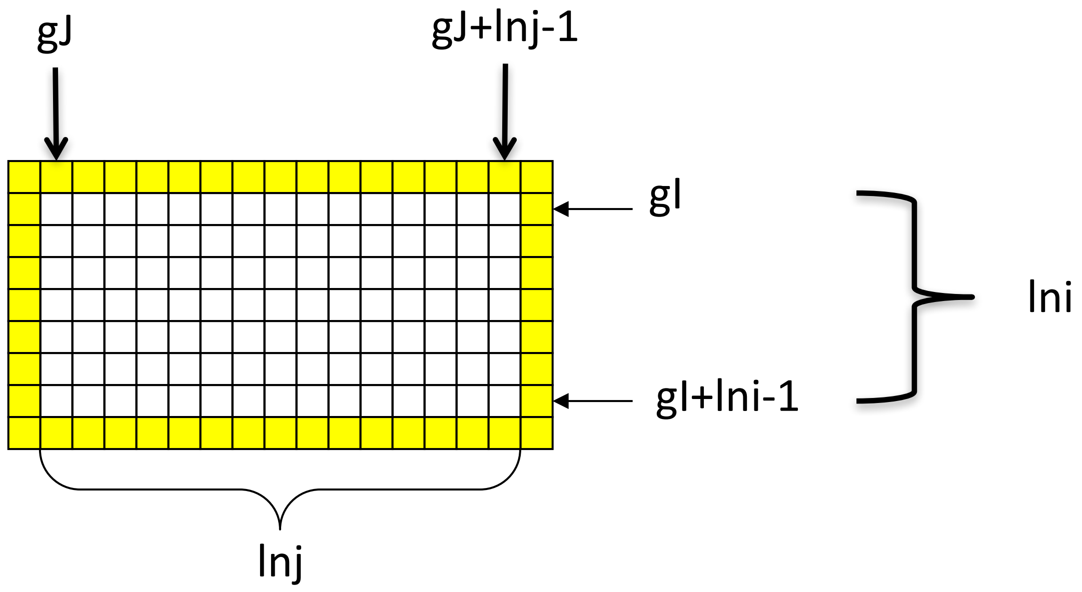
Necessary Data Transfers

- Provide access to remote data through a *halo* exchange (9 point with trick)



The Local Data Structure

- Each process has its local "patch" of the global array
  - "g" gives indices and sizes in the global array
  - "l" gives sizes in the local array
  - Always allocate a halo around the patch



The Program

- mlife2d.c
  - Main program
  - Runs “Life” with 5 different choices of halo exchange implementations and with 2 different communicators
  - Runs each of those twice
    - For some systems, the first run will involve library and network initialization, and the timing will reflect that (it does for me on my Cray)
  - Prints the time taken for each of the exchange implementation each time

The Program

- moptions.c
  - Contains routine to parse command line options and to abort with a message
  - Note use of an MPI Datatype to distribute options from process 0 to all other processes
  - Returns values in MLIFEoptions struct.
  - Modify this file to change defaults or add new options (e.g., you could select a single exchange algorithm to run)

The Program

- patch.c
  - Contains routines that
    - Determine a 2-d array of processes
    - Determine the coordinates and sizes of the local patch on each process in the 2-d array
  - Returns the information in the struct MLIFEPatchDesc .

The Program

- mtiming.c
  - Contains the routines to execute Conway's game of Life and to time the communication costs and the time spent in the "life" iteration.
  - Timing returned in struct MLIFETiming
  - Loop over i,j and that calls "MLIFE_nextstate" implements the stencil computation for "life"
    - This is *not* a high performance implementation
  - Fortran users: The (*exchange)(patch …) is simply a call to the function that was passed as the routine parameter "exchange". In Fortran, this parameter would have been declared external.
  - Fortran users: A C trick is used to "swap" the arrays so that the next state is always in array "m2", computed from "m1". In Fortran, you can do the same thing with Fortran pointers.

The Program

- These files implement the halo exchange:
  - mlife2d-pt2pt.c : Use MPI vector datatype and MPI_Isend and MPI_Irecv, with the “diagonal trick”
  - mlife2d-pt2ptuv.c : Use user packing/unpacking instead of MPI vector datatypes (the UV in the name)
  - mlife2d-pt2ptsnd.c : Use MPI_Send and MPI_Irecv, with MPI vector datatype
  - mlife2d-pt2pt9.c : Use MPI_Irecv and MPI_Isend, send directly to all 8 neighbors (do not use diagonal trick)
  - mlife2d-fence.c : Use MPI RMA with Put

The Program

- Each Exchange file implements three functions
  - An "Init". Creates any persistent data needed for the exchange. Returns a pointer to the data (with *(void **)privateData = (void*)ptr). Return a null pointer if there is no persistent data
  - An "End". Frees any allocated objects
  - An "Exchange". Performs the halo exchange

The Program

- mlife2d-io-stdout1.c
  - Implements a simple, character oriented output to stdout
  - Only for 256 columns
  - Entire display must be visible (uses xterm/ascii terminal command sequences)
  - Contains an init, “checkpoint”, finalize, and some other functions
    - General interface to all creating and restoring from checkpoints; other implementations of this routines in this file can use MPI IO to provide parallel I/O for large arrays

Building and Running the Program

- First run configure, passing the name of the C compiler for MPI as the MPICC variable. E.g.,
  - ./configure MPICC=mpcc
- Make
- Run with mpiexec or as appropriate for your system:
  - mpiexec –n 16 ./mlife2d –x 1000 –y 1000 –i 100
- Run with visual output:
  - mpiexec –n 4 ./mlife2d –c –i 10