Regular Mesh Algorithms

- Many scientific applications involve the solution of partial differential equations (PDEs)
- Many algorithms for approximating the solution of PDEs rely on forming a set of difference equations
  - Finite difference, finite elements, finite volume
- The exact form of the difference equations depends on the particular method
  - From the point of view of parallel programming for these algorithms, the operations are the same

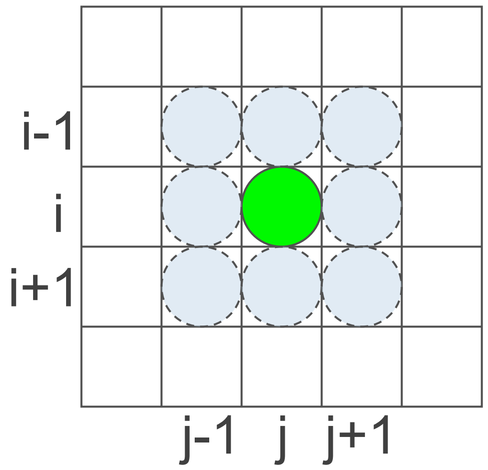
Conway's Game of Life

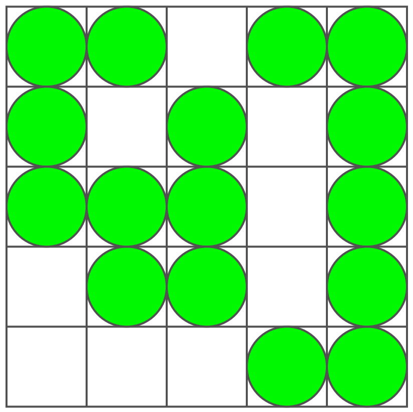
- In this tutorial, we use Conway's Game of Life as a simple example to illustrate the program issues common to many codes that use regular meshes, such as PDE solvers
  - Allows us to concentrate on the MPI issues

- 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

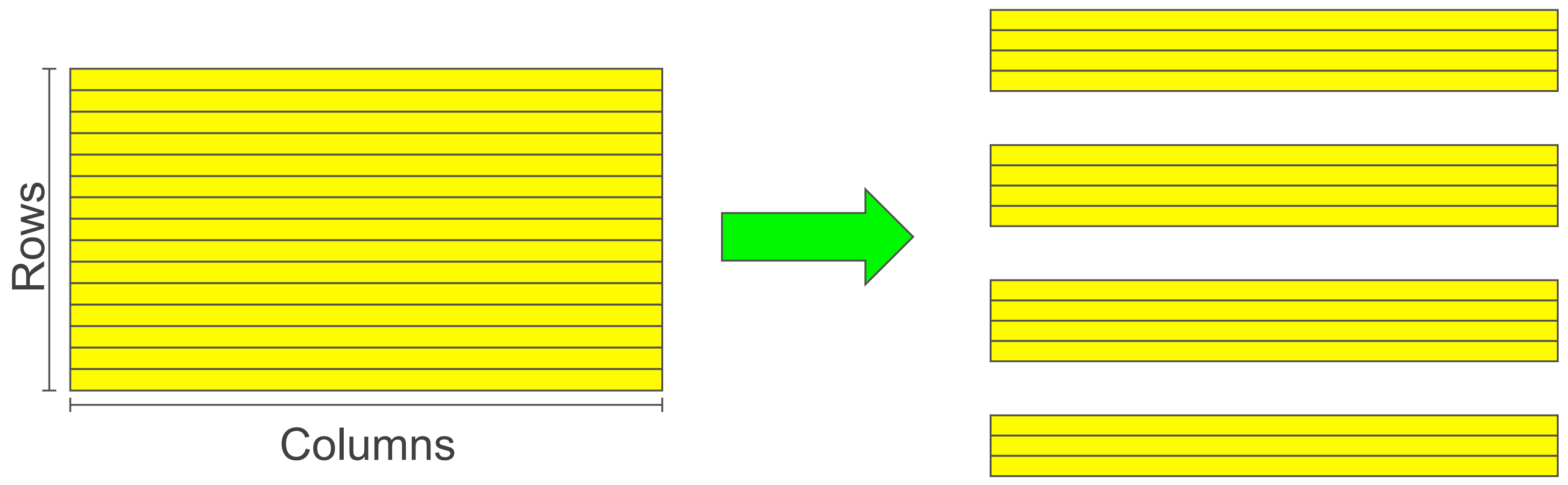
- For the non-parallel version, we:
  - Allocate a 2D matrix to hold state
    - Actually two matrices, and we will swap them between steps
  - Initialize the matrix
    - Force boundaries to be "dead"
    - Randomly generate states inside
  - At each time step:
    - Calculate each new cell state based 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

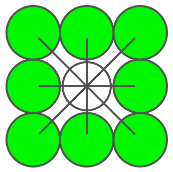
Step 1: Description of Decomposition

- By rows (1D or row-block)
  - Each process gets a group of adjacent rows
- Later we'll show a 2D decomposition

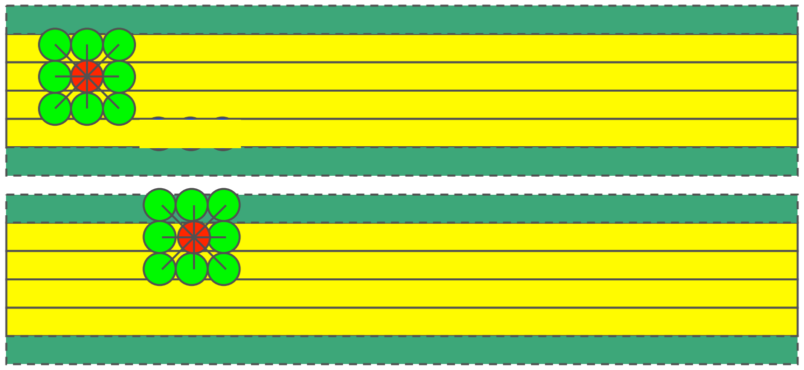


Step 2: Communication

- "Stencil" requires read access to data from neighbor cells



- We allocate extra space on each process to store neighbor cells
- Use send/recv or RMA to update prior to computation

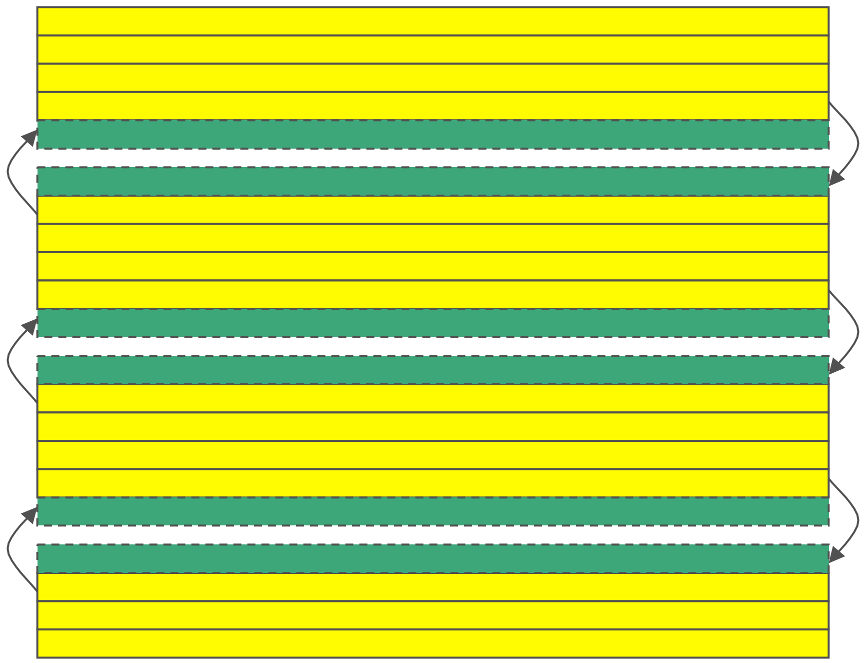


Step 3: Define the Local Arrays

- Correspondence between the local and global array
- "Global" array is an abstraction; there is no one global array allocated anywhere
- Instead, we compute parts of it (the local arrays) on each process
- Provide ways to output the global array by combining the values on each process (parallel I/O!)

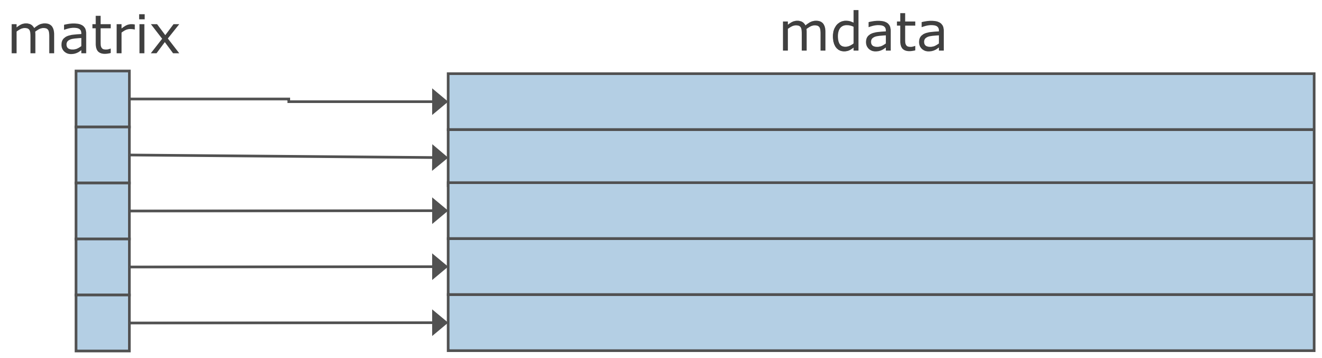
Boundary Regions

- In order to calculate next state of cells in edge rows, need data from adjacent rows
- Need to communicate these regions at each step
  - First cut: use Isend and Irecv
  - Revisit with RMA later



Life Point-to-Point Code Walkthrough

- Points to observe in the code:
  - Handling of command-line arguments
  - Allocation of local arrays
  - Use of a routine to implement halo exchange
    - Hides details of exchange



Allows us to use matrix[row][col] to address elements

See mlife.c pp. 1-8 for code example.

Note: Parsing Arguments

- MPI standard does not guarantee that command line arguments will be passed to all processes.
  - Process arguments on rank 0
  - Broadcast options to others
    - Derived types allow one bcast to handle most args
  - Two ways to deal with strings
    - Big, fixed-size buffers
    - Two-step approach: size first, data second (what we do in the code)

See mlife.c pp. 9-10 for code example.

Point-to-Point Exchange

- Duplicate communicator to ensure communications do not conflict
  - This is good practice when developing MPI codes, but is not required in this code
  - If this code were made into a component for use in other codes, the duplicate communicator would be required
- Non-blocking sends and receives allow implementation greater flexibility in passing messages

See mlife-pt2pt.c pp. 1-3 for code example.