KOÇ UNIVERSITY

Case Study: Stencil Computation in MPI

Didem Unat

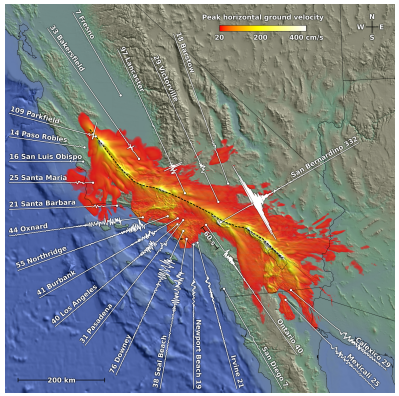
COMP 429/529 Parallel Programming

MPI

- MPI abstracts the network topology and process placement on network
  - So that user doesn't need to know about it
- As you have seen, both topology and process placement have performance implications
  - However, it is too difficult to write a portable program that manages both topology and process placement
- There are features in MPI to make more intelligent placement of ranks on the network
  - Active research area
  - Application has to be adaptive because it may run on a different machine or different subset of the network over time

Stencil Computation

- Stencils are the basis for many algorithms to numerically solve partial differential equations (PDEs)
  - Approximate derivatives numerically
  - Countless examples in scientific and engineering problems
    - Turbulence flow, seismic wave propagation, ocean currents, water flow in a pipe, blood flow, weather forecast, air flow around a wing
  - Multimedia applications (e.g. image smoothing)
  - Deep Neural Network (e.g convolution)
  - Uses finite difference or finite volume method



Earthquake Simulation
Source: Yifeng Cui (SDSC)

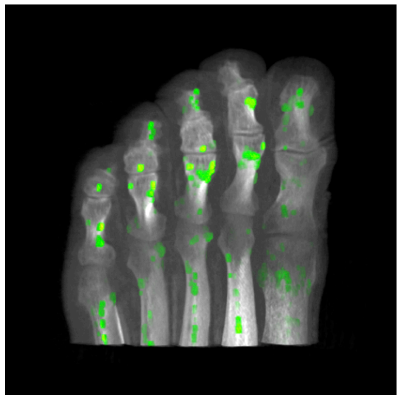
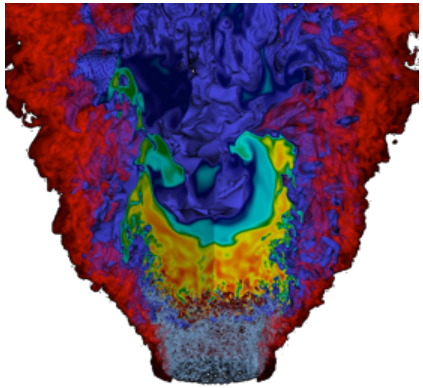


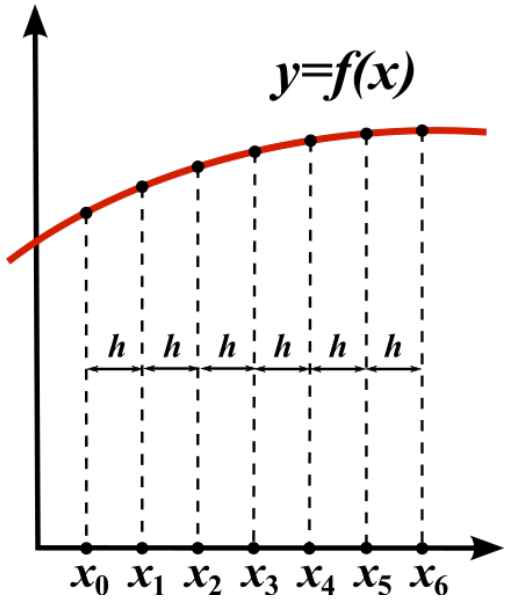
Image Corner Detection
Source: Han Suk Kim (Apple)



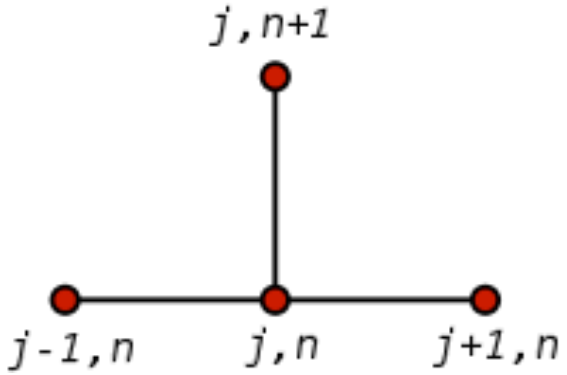
Combustion Turbulence Flow
Source: John Bell (LBNL)

Stencil Computation

- Discretizes the problem domain
- Divides the domain in space into a uniform grid (structured grid) x_0, x_1 …x_n and in time t_0, …t_N
- Produces sets of discrete numerical approximations to the derivative in an iterative manner (time-stepping)
- Performs nearest neighbors update



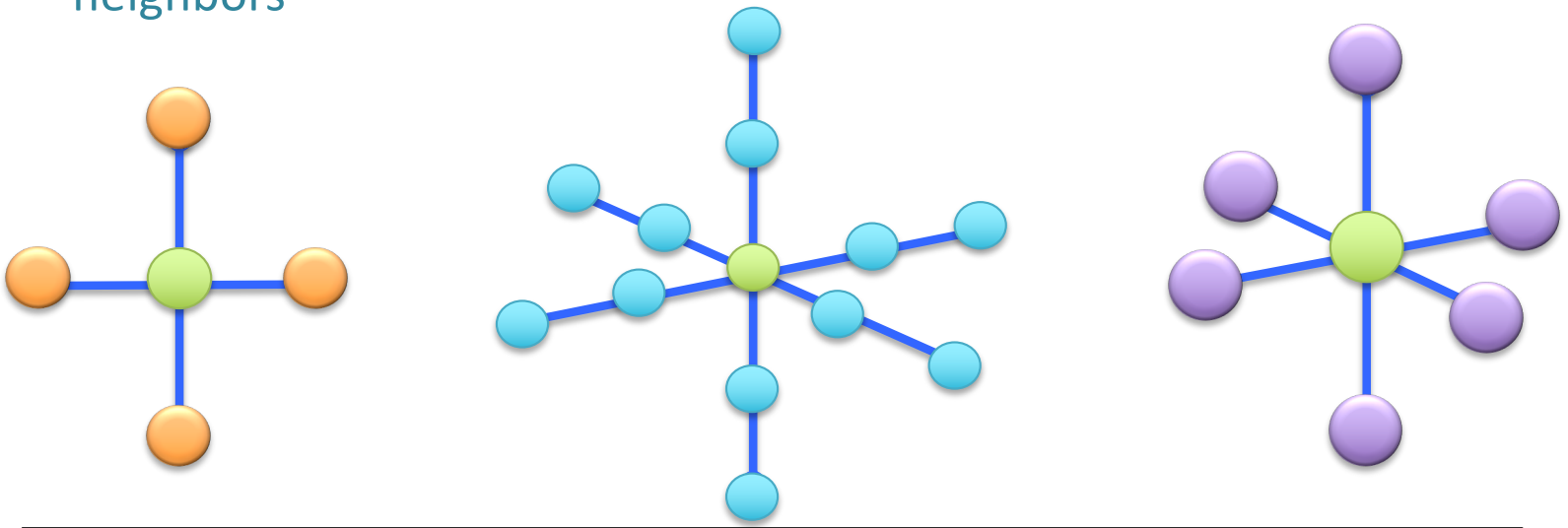
Src: Wikipedia



The stencil for the most common explicit method for the heat equation.

Stencil Shape

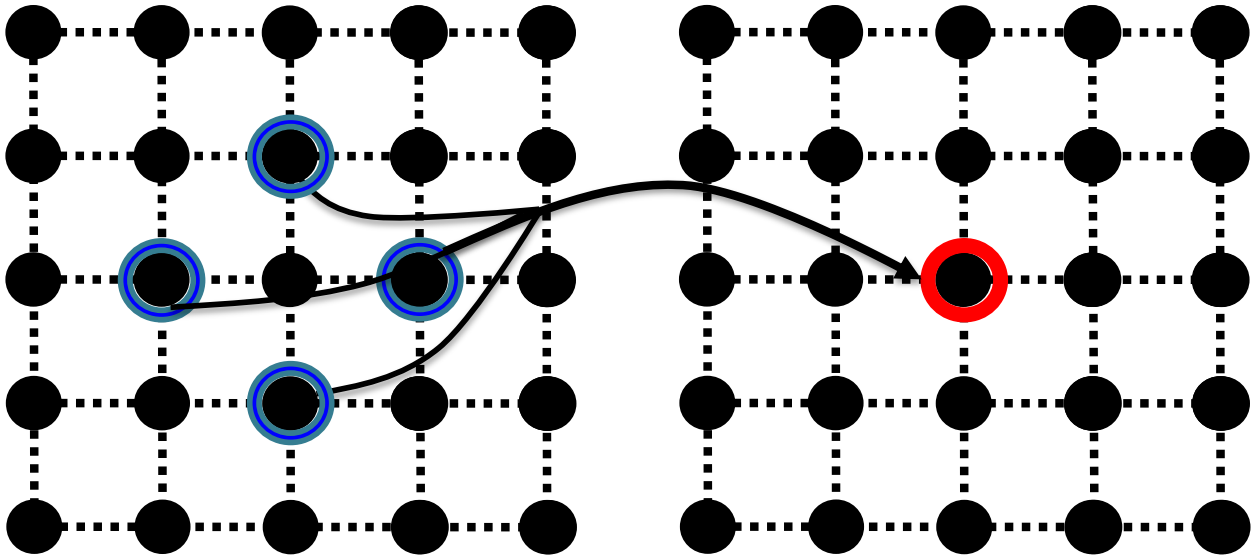
- Stencil shape can be different depending on the numerical algorithm used
- Here 5-point in 2D, 13-point and 7-point in 3D are shown
  - Next value of green point depends on the current value in the nearest neighbors



```
//image smoothing example using 5-point stencil
for iter = 1 : nSmooth
  for (i,j) in (0:N-1,  0:N-1)
       Imgnew [i,j] = (Img[i-1,j]+Img[i+1,j]+Img[i,j-1]+Img[i, j+1])/4
  Swap(Imgnew,Img)
```

Cell Update

- Since we are updating the next value of a cell from its neighbors, typically we need to keep two grids around
  - Current and Next Grid
- There are techniques to implement with one grid, they are called update-in-place methods

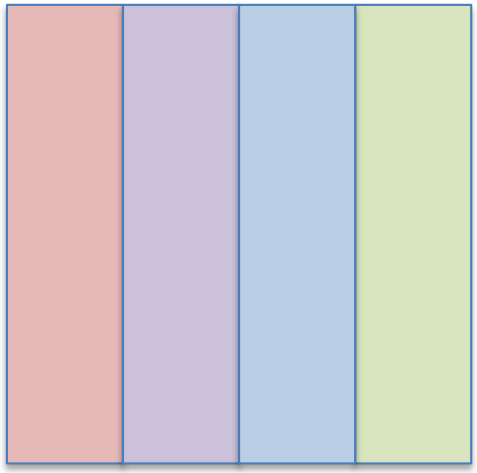


Grid(*t*) Grid(*t+1*)

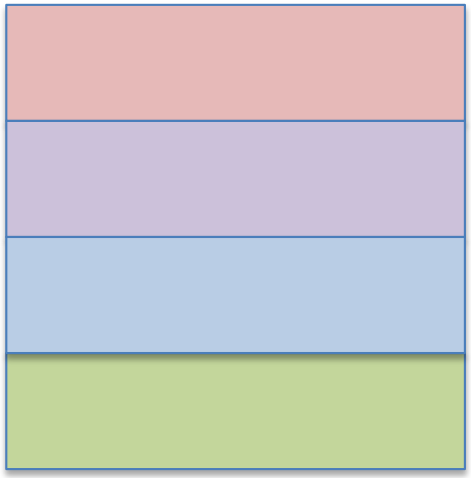
Parallel Implementation

- Partition the structured grid, assigning each partition to a unique process
- Different partitioning methods according to the ***processor geometry***

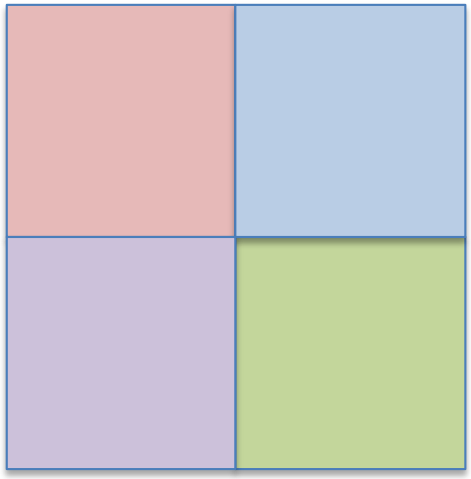
4 X 1



1 X 4

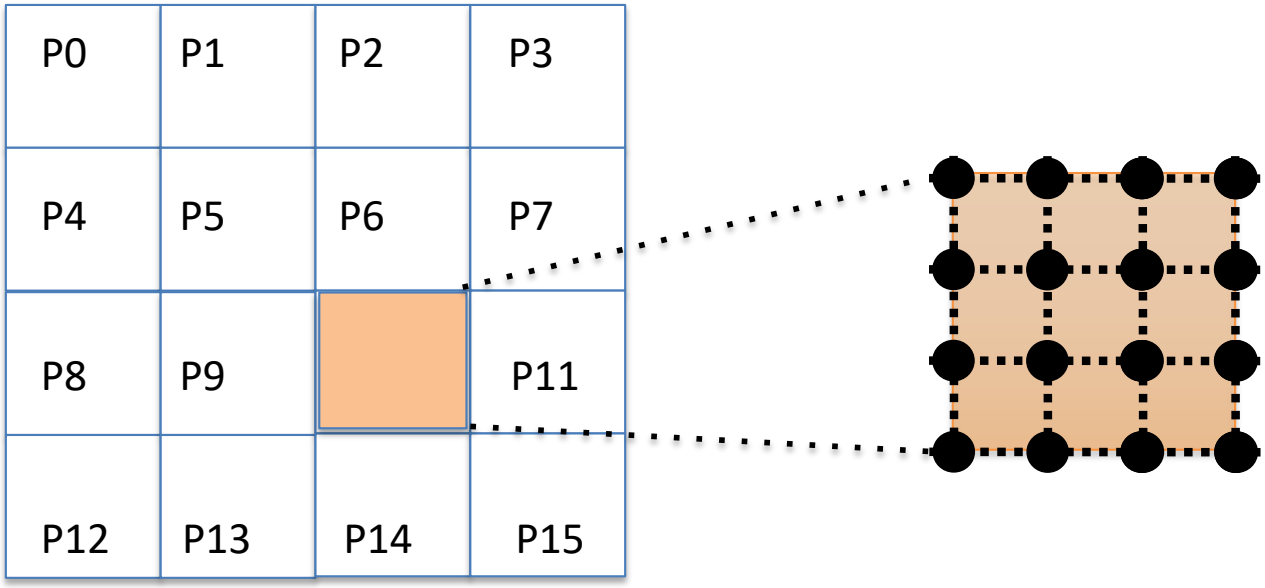


2 X 2



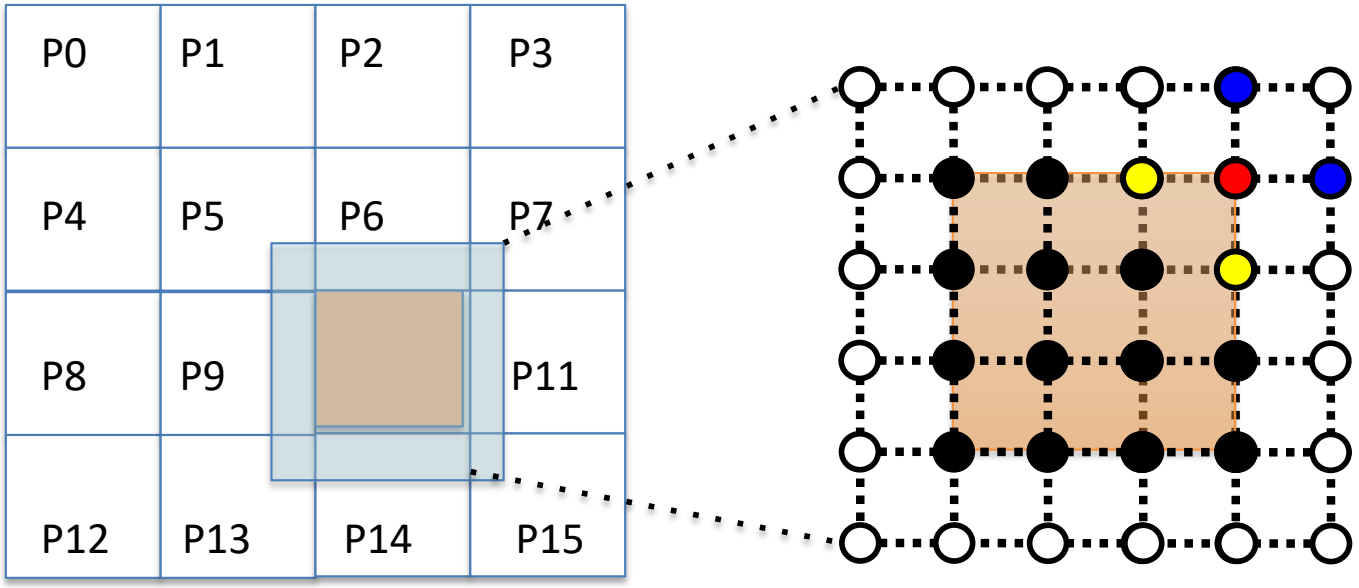
Data Dependences

- Each process has its own data partition
- Do we need any data points from neighboring processes?



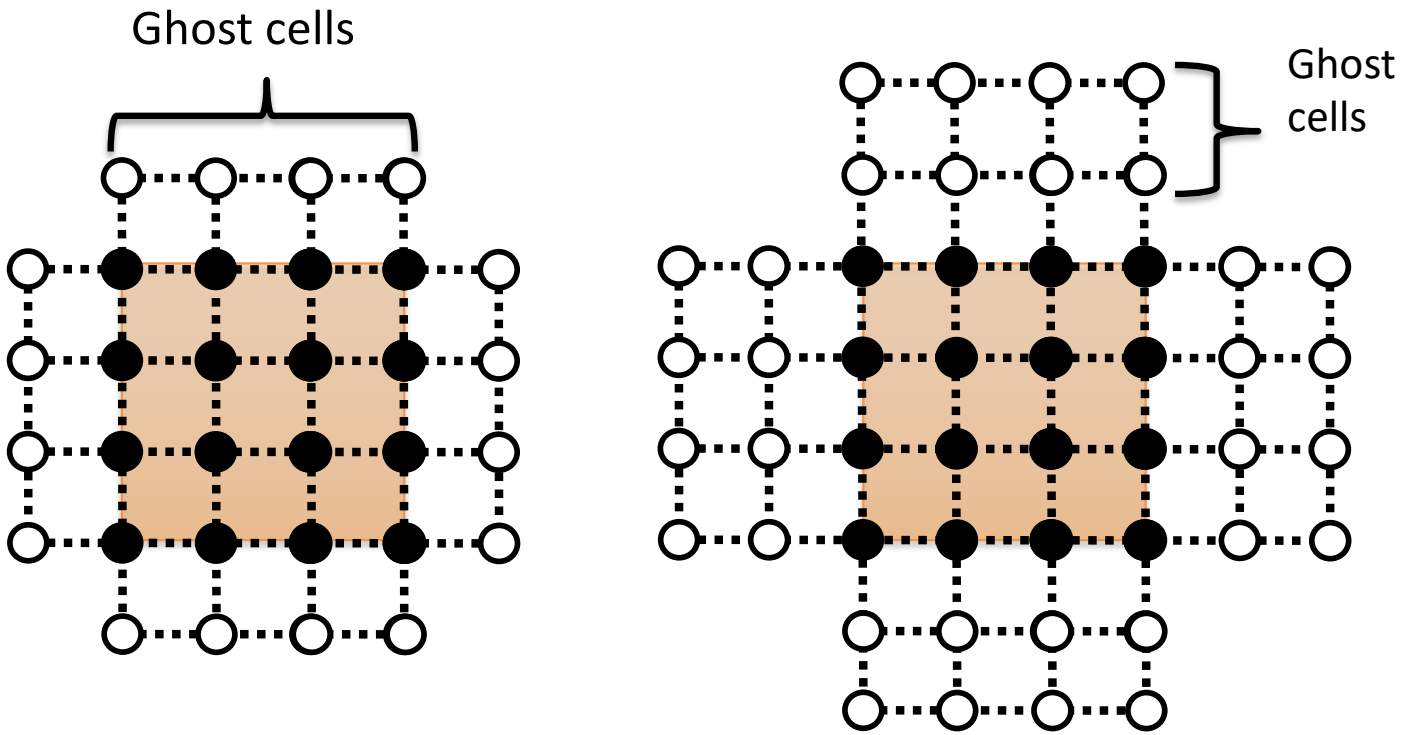
Data Dependences

- In order to update red, we need yellow points (local) and blue points from other processes (remote)
  - Too expensive to communication individual array values one at a time
  - “Ghost” cells hold a copies off-process values



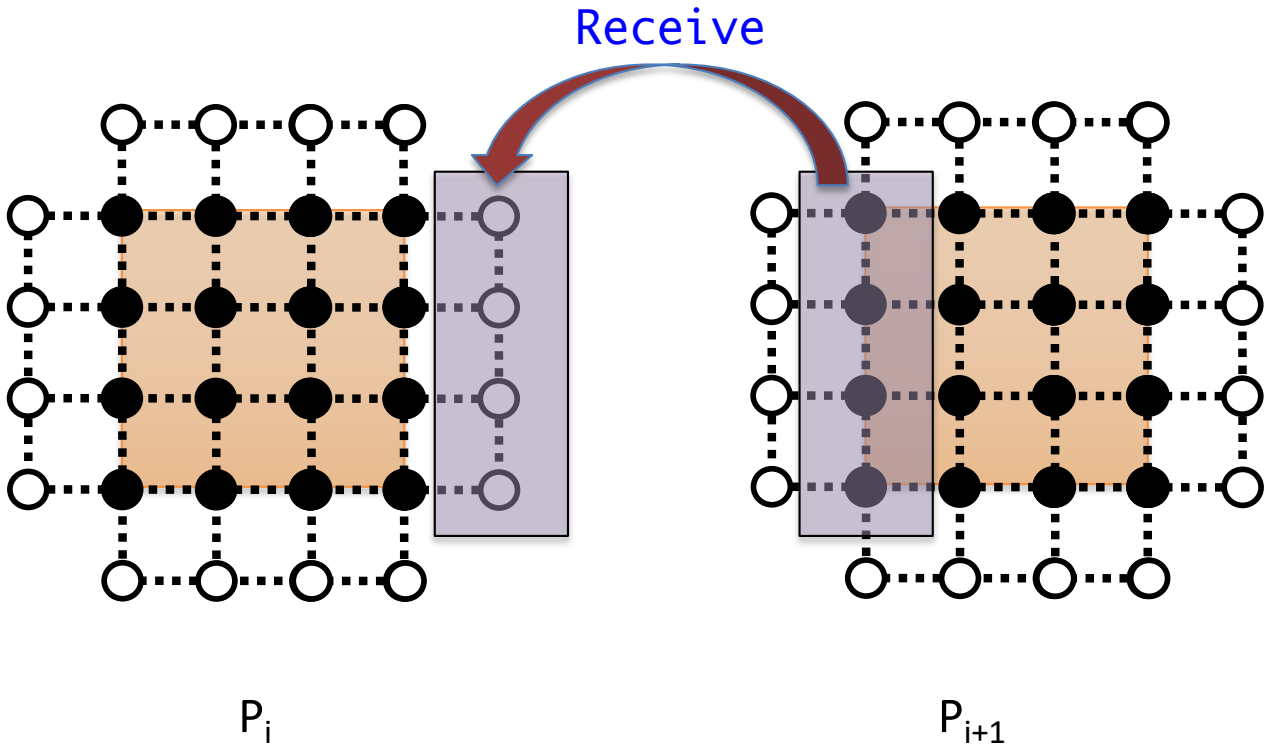
Ghost Cells (Halos)

- Ghost cells are allocated for the information needed from other processes
  - Thus each processor allocates (N+ 2*ghosts)^{d} space to accommodate ghost region
  - Depth of ghost region depends on the numerical method used



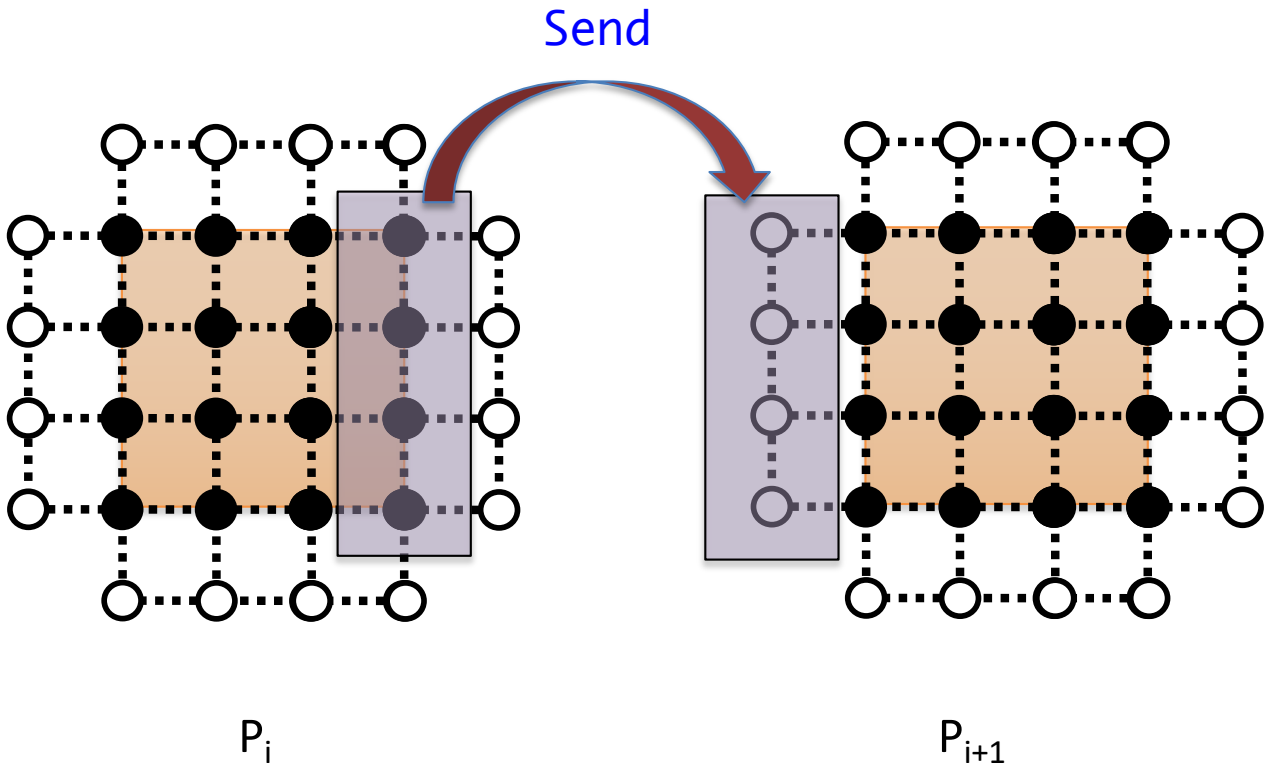
Ghost Cell Exchange- Receive

- Need to update ghosts every iteration so that a process has the latest values
- This is done with message exchange between processes



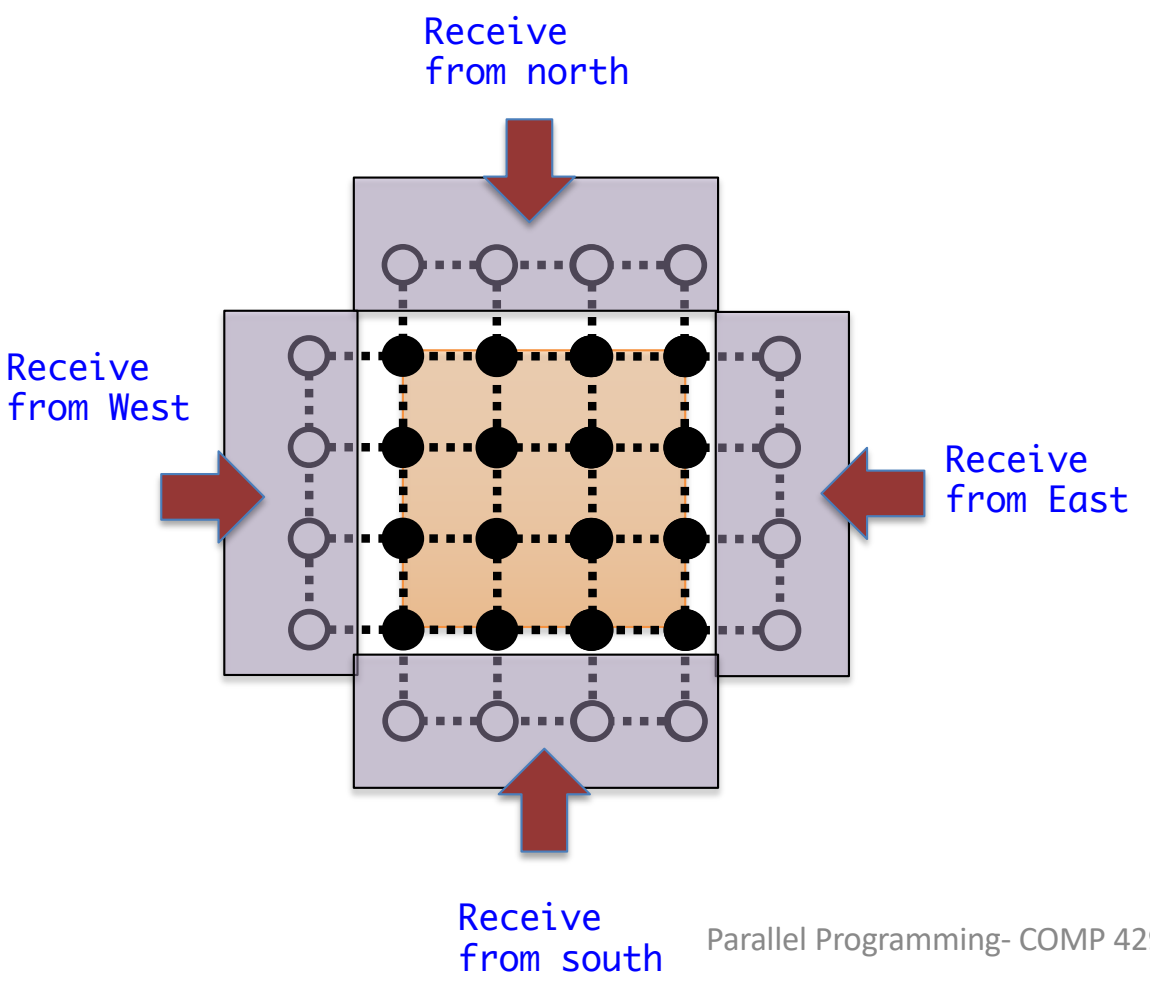
Ghost Cell Exchange- Send

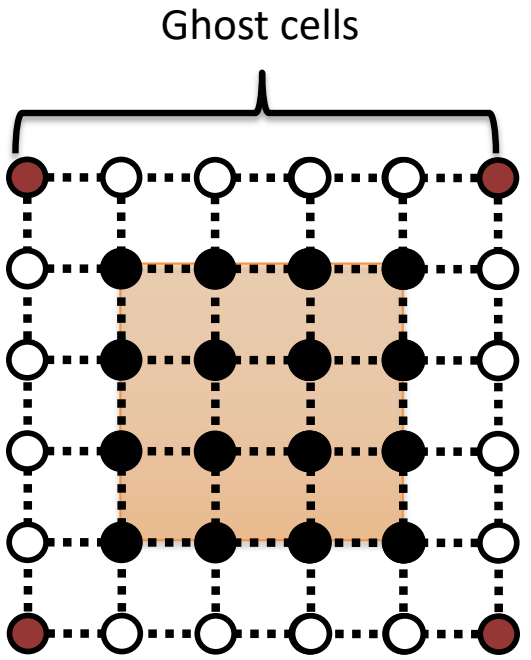
- Need to update ghosts every iteration so that a process has the latest values
- This is done with message exchange between processes
  - Note that the elements may not be contiguous in memory – need to pack a message at the source and unpack at the destination



Message Exchange with All Neighbors

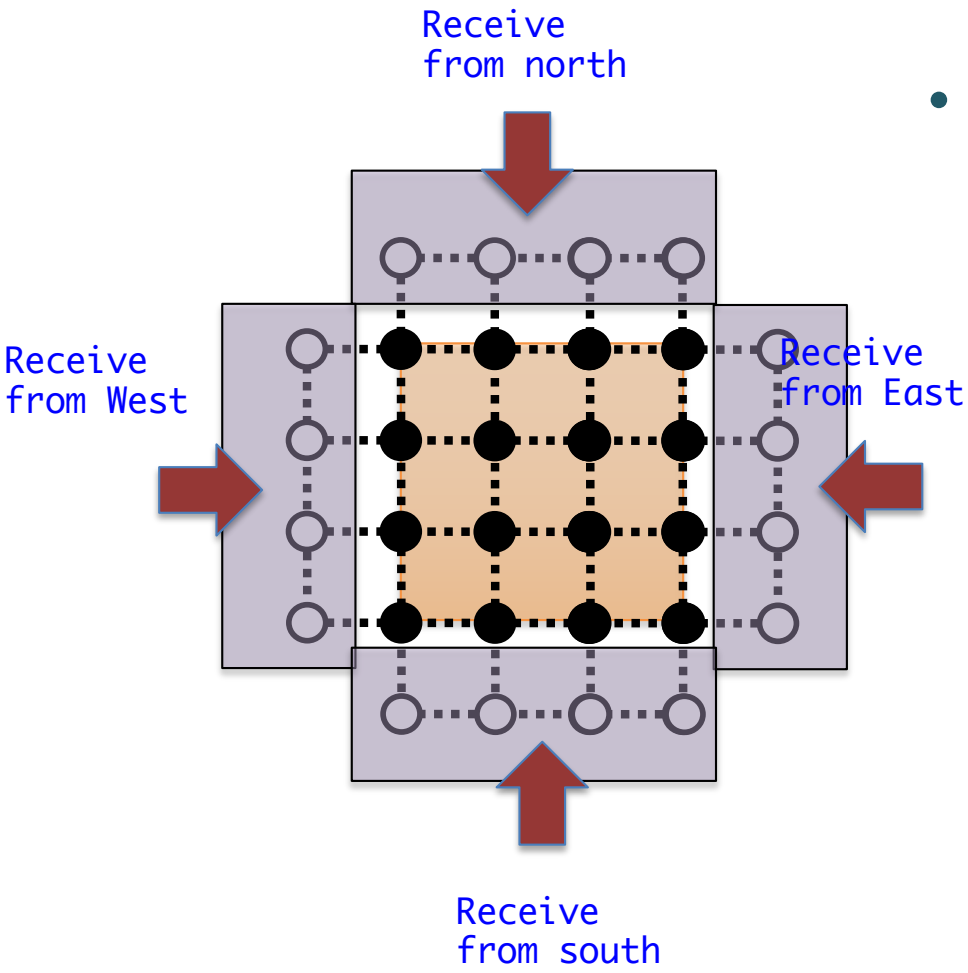
- How many processors involved in communication depends on the numerical algorithm and domain decomposition
  - In 2D, typically sends/receives involve 4 neighbors (N, S, W, E)
- How many neighbors are there for 3D problems?





If the computation uses the corner cells in the update, then need to exchange ghosts with 4 more processes: NE, NW, SE and SW

Corner Cells



- Instead of exchanging messages with corner processes, experimental studies showed that it is better to receive messages from N, S, W, E in two steps
  - Because corner points are also sent to our neighbors
  - For example, NW corner point has to be sent to our N and W neighbors. We can receive that data from them instead of our NW neighbor

Ghost Cell Exchange

- What happens to processors that are at the edge of the physical domain of the problem?
  - For example, no north and west neighbors for P0

P0	P1	P2	P3
P4	P5	P6	P7
P8	P9	P10	P11
P12	P13	P14	P15

- Depends again on the numerical algorithms' boundary condition
  - If it is periodic, then P0's west neighbor is P3
  - Or it may mirror itself
  - Or it may be set to a constant value (zero).
  - Or it may use a function to calculate its value at the boundary

ODE and PDE Solvers

- ODE solver:
  - No data dependency, trivially parallelizable
  - Requires a lot of registers to hold temporary variables
  - CPU intensive
- PDE solver:
  - Jacobi update for the 5-point Laplacian operator.
  - Sweeps over a uniformly spaced mesh
  - Updates the voltage according to weighted contributions from the four nearest neighboring positions in space

PDE part:

$$E^{t}[i,j] = E^{t-1}[i,j] + \alpha(E^{t-1}[i+1,j] + E^{t-1}[i-1,j] + E^{t-1}[i,j+1] + E^{t-1}[i,j-1] - 4E^{t-1}[i,j])$$

Pseudo Code for Parallelization

```
Broadcast the input arguments
Create buffers for send and receive messages

While (t < T)
    Post Ireceive for all neighbors
    Send data to neighbors
    Wait for completion
    Unpack the message
    Perform the PDE and ODE solvers

    If (plot at t is true and rank is 0)
      Gather data from all processes
      Plot the mesh
```

Non-blocking P2P comm

Collective Comm

Acknowledgments

- These slides are inspired and partly adapted from
  - Mary Hall (Univ. of Utah)
  - Scott Baden (UCSD)
  - The course book (Pacheco)
  - https://computing.llnl.gov/tutorials/mpi/