# ECE 669

# Parallel Computer Architecture

## Lecture 5

## *Grid Computations*

# Outline

- **Motivating Problems (application case studies)**
- **Classifying problems**
- **Parallelizing applications**
- ***Examining tradeoffs***
- **Understanding communication costs**
  - **Remember: software and communication!**

# Current status

- We saw how to set up a system of equations
- How to solve them
  - Iterative
    - Jacobi, ...
    - Multigrid...
    - ⋮
  - Direct
- Poisson: Basic idea

$$0 = \frac{1}{\Delta s^2}\left[A_{i+1,j} + A_{i-1,j} + A_{i,j+1} + A^{k}_{i,j-1} - 4A_{i,j}\right] + B_{i,j}$$

**Or**

$$A_{i,j} = \frac{A_{i+1,j} + A_{i-1,j} + A_{i,j+1} + A_{i,j-1}}{4} + C_{i,j}$$

0 for Laplace

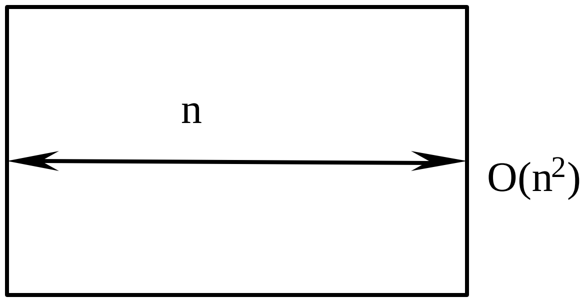
- In iterative methods

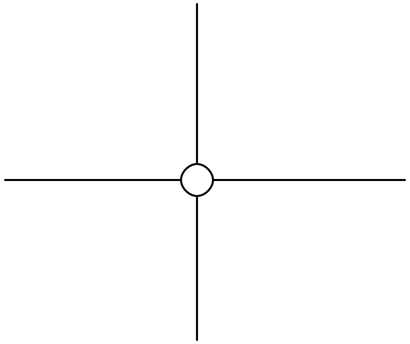
$$A^{k+1}_{i,j} = \frac{A^{k}_{i+1,j} + A^{k}_{i-1,j} + A^{k}_{i,j+1} + A^{k}_{i,j-1}}{4} = C_{i,j}$$

  - Iterate till no difference
  - The ultimate parallel method

# Examining Optimizations

**Slow!**

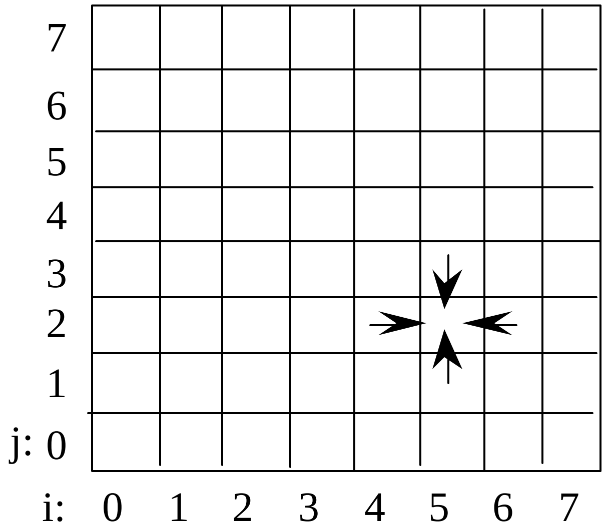


**Jacobi relaxation**

- Optimizations
  - SOR
  - Gauss-Seidel
    - Use recent values ASAP

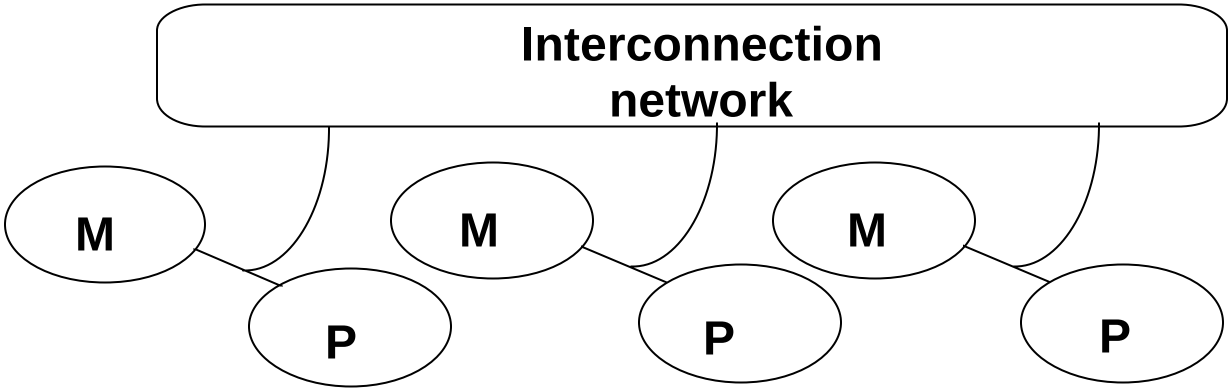
# Parallel Implementation




$$A_{i,j} = \frac{\downarrow + \rightarrow + \leftarrow + \uparrow}{4}$$

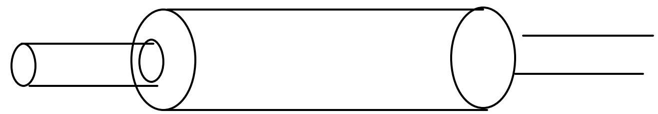
- **Q: How would you partition the problem?**
  - **Say, on 4 processors?**
- **Communication!**
- **What about synchronization?**

# Machine model




- Data is distributed among memories (ignore initial I/O costs)
- Communication over network-explicit
- Processor can compute only on data in local memory.
  - To effect communication, processor sends data to other node

# Turbo charging – Iterative methods



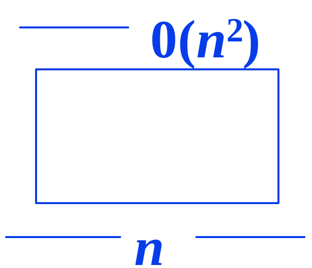
$$\nabla^2 A + B = 0$$

$$A_{i,j}^{k+1} = \frac{A_{i+1,j}^{k} + A_{i-1,j}^{k} + A_{i,j+1}^{k} + A_{i,j-1}^{k}}{4} + b_{i,j}$$

- **SOR: Successive Over Relaxation**
  - **Accelerate towards direction of change**

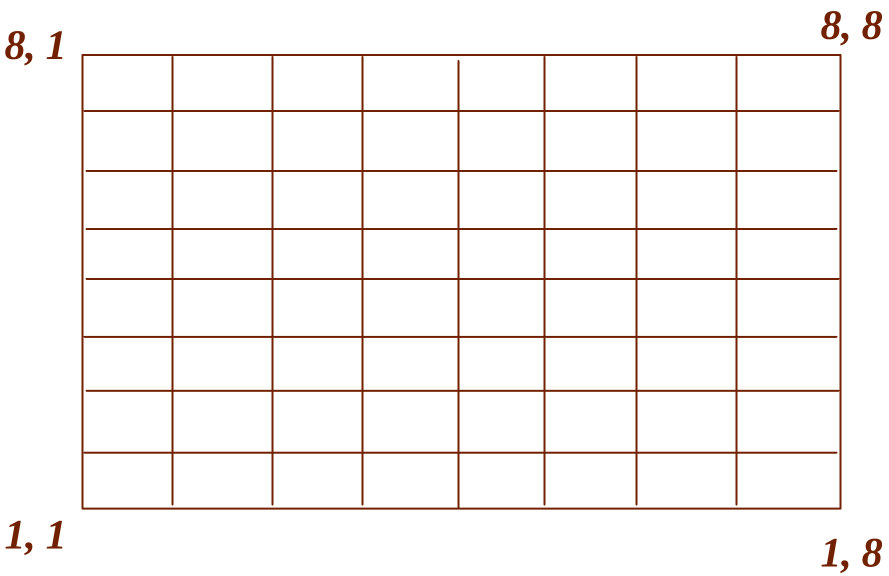
$$A_{i,j}^{k+1} = A_{i,j}^{k} + w\left[\underbrace{\frac{A_{i+1,j}^{k} + A_{i-1,j}^{k} + A_{i,j+1}^{k} + A_{i,j-1}^{k}}{4} - A_{i,j}^{k}}_{\text{Difference (new-old)}}\right]$$

**Difference (new-old)**



# Multigrid

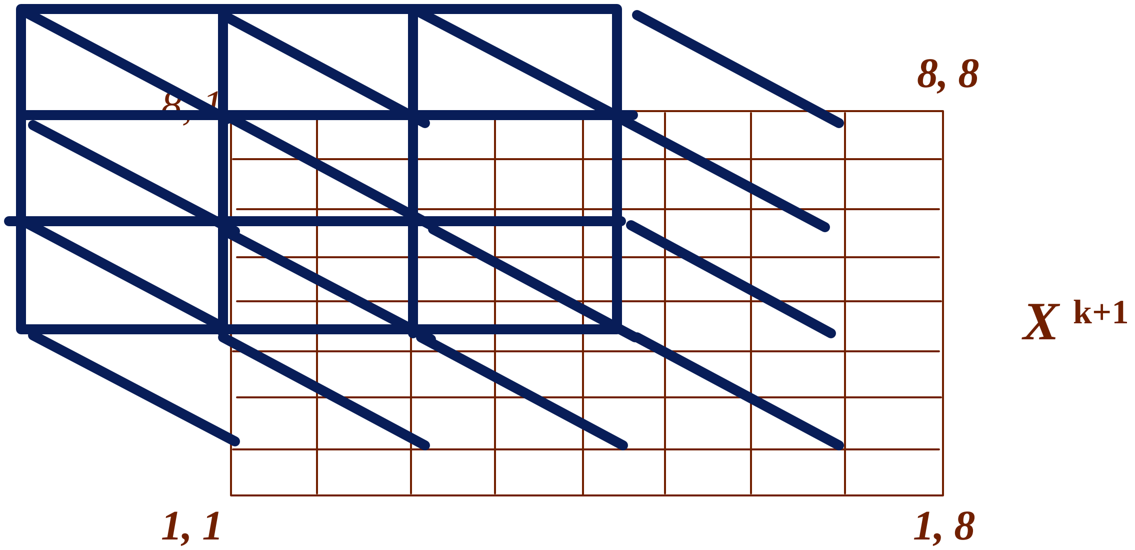
- Basic idea ---> Solve on coarse grid ---> then on fine grid
  - In practice -- solve for errors in next finer grid.  But communication and computation patterns stay the same.




$X^{k+1}$

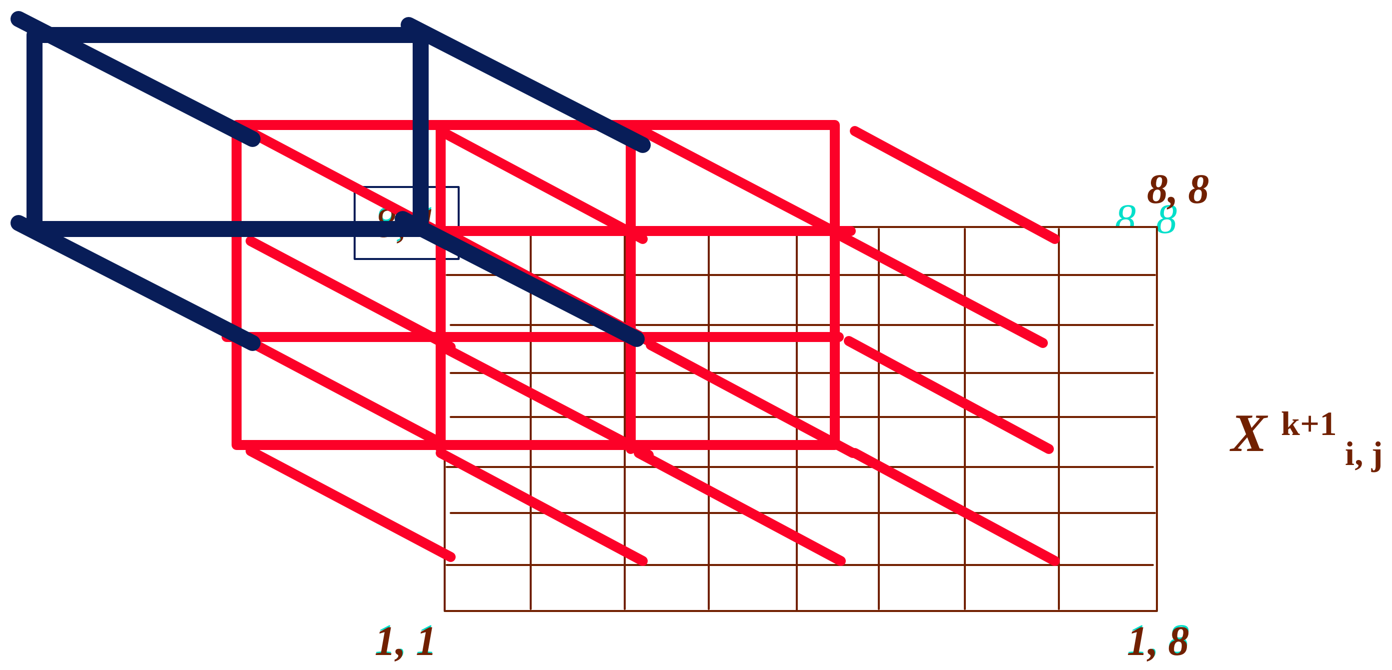
# Multigrid

- Basic idea ---> Solve on coarse grid
  ---> then on fine grid



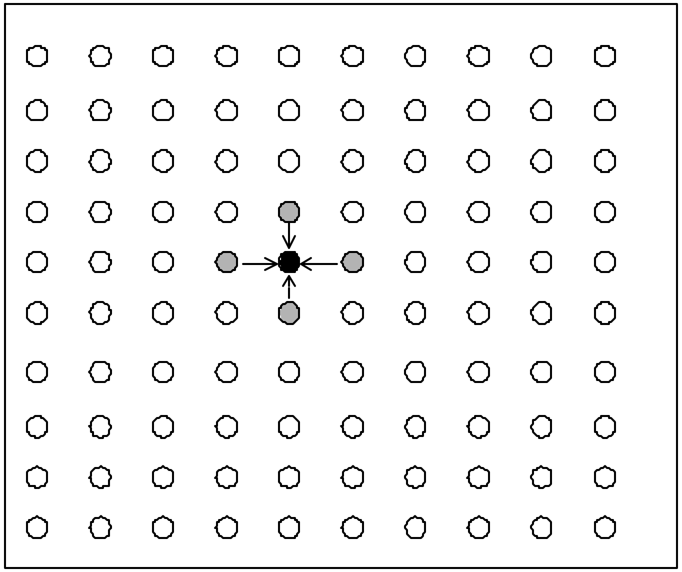
# Multigrid

- Basic idea ---> Solve on coarse grid
  ---> then on fine grid



# Example: iterative equation solver

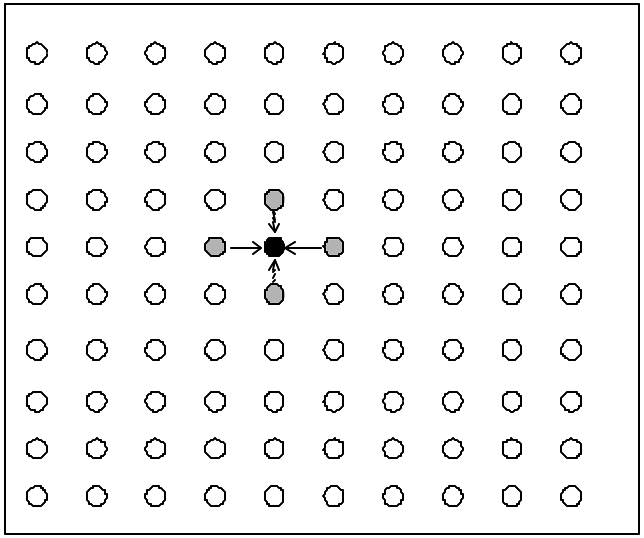
- **Simplified version of a piece of Ocean simulation**
- **Illustrate program in low-level parallel language**
  - **C-like pseudocode with simple extensions for parallelism**
  - **Expose basic comm. and synch. primitives**
  - **State of most real parallel programming today**



Expression for updating each interior point:

$$A[i,j] = 0.2 \times (A[i,j] + A[i,j-1] + A[i-1,j] + A[i,j+1] + A[i+1,j])$$

# Grid Solver



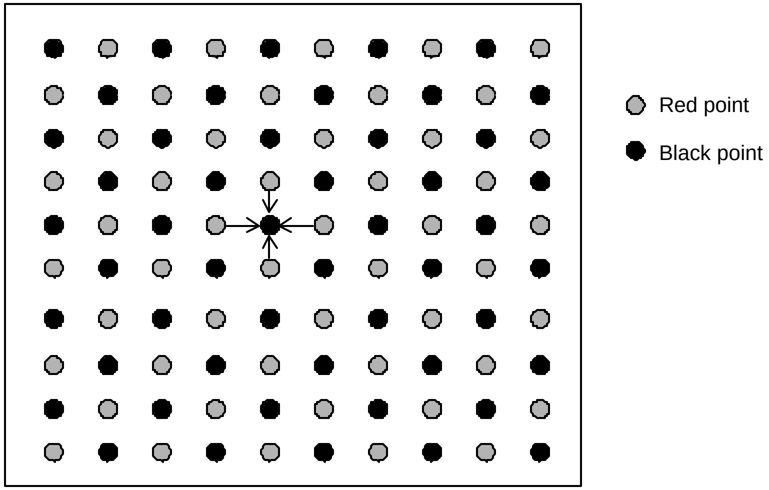
Expression for updating each interior point:

$$A[i,j] = 0.2 \times (A[i,j] + A[i,j-1] + A[i-1,j] + A[i,j+1] + A[i+1,j])$$

- **Gauss-Seidel (near-neighbor) sweeps to convergence**
  - **Interior n-by-n points of (n+2)-by-(n+2) updated in each sweep**
  - **Updates done in-place in grid**
  - **Difference from previous value computed**
  - **Check  if has converged**
    - **to within a tolerance parameter**

# Exploit Application Knowledge

- Reorder grid traversal: red-black ordering




- **Different ordering of updates: may converge quicker or slower**
- **Red sweep and black sweep are each fully parallel:**
- **Global synchronization between them**

# Point to Point Event Synchronization

- **One process notifies another of an event so it can proceed**
  - **Common example: producer-consumer (bounded buffer)**
  - **Concurrent programming on uniprocessor: semaphores**
  - **Shared address space parallel programs: semaphores, or use ordinary variables as flags**

| $P_1$ | $P_2$ |
|---|---|
| | `A = 1;` |
| `a: while (flag is 0) do nothing;` | `b: flag = 1;` |
| `print A;` | |

•*Busy-waiting* or *spinning*

# Group Event Synchronization

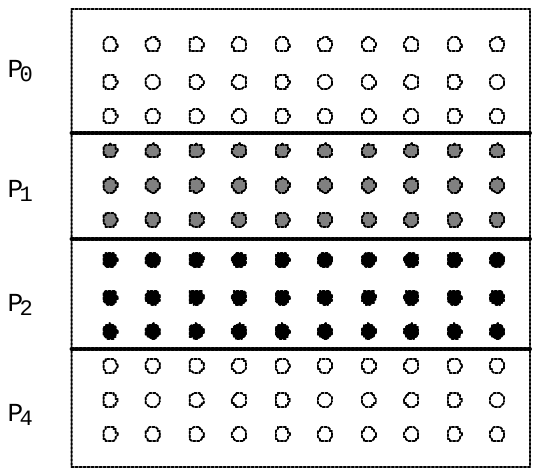
- **Subset of processes involved**
  - **Can use flags or barriers (involving only the subset)**
  - **Concept of producers and consumers**

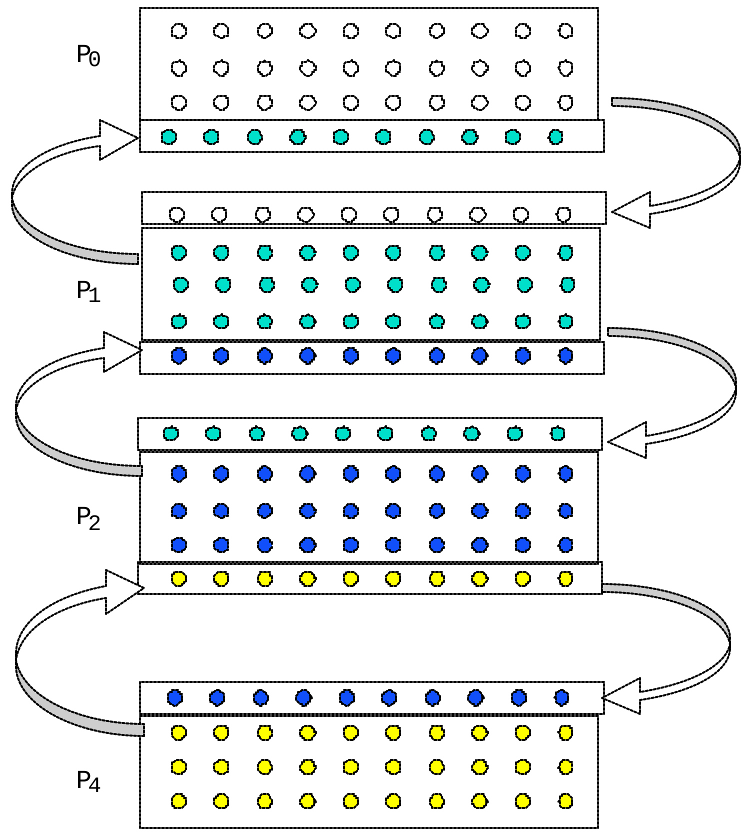
- **Major types:**
  - **Single-producer, multiple-consumer**
  - **Multiple-producer, single-consumer**
  - **Multiple-producer, single-consumer**

# Message Passing Grid Solver

- Cannot declare A to be global shared array
  - compose it logically from per-process private arrays
  - usually allocated in accordance with the assignment of work
    - process assigned a set of rows allocates them locally
- Transfers of entire rows between traversals
- Structurally similar to shared memory
- Orchestration different
  - data structures and data access/naming
  - communication
  - synchronization
- Ghost rows

# Data Layout and Orchestration




Data partition allocated per processor

Add ghost rows to hold boundary data

Send edges to neighbors

Receive into ghost rows

Compute as in sequential program

# Notes on Message Passing Program

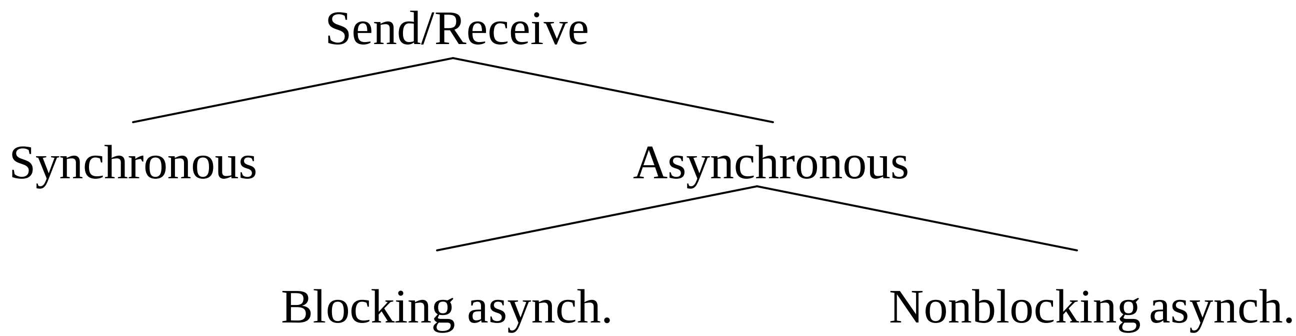
- Use of ghost rows
- Communication done at beginning of iteration, so no asynchrony
- Communication in whole rows, not element at a time
- Core  similar, but indices/bounds in local rather than global space
- Synchronization through sends and receives
  - Could implement locks and barriers with messages

# Send and Receive Alternatives

Can extend functionality: stride, scatter-gather, groups

Semantic flavors: based on when control is returned

Affect when data structures or buffers can be reused at either end




- **Affect event synch (mutual excl. by fiat: only one process touches data)**
- **Affect ease of programming and performance**

° **Synchronous messages provide built-in synch. through match**

# Orchestration: Summary

- **Shared address space**
  - **Shared and private data explicitly separate**
  - **Communication implicit in access patterns**
  - **No *correctness* need for data distribution**
  - **Synchronization via atomic operations on shared data**
  - **Synchronization explicit and distinct from data communication**

- **Message passing**
  - **Data distribution among local address spaces needed**
  - **No explicit shared structures (implicit in comm. patterns)**
  - **Communication is explicit**
  - **Synchronization implicit in communication (at least in synch. case)**

# Correctness in Grid Solver Program

| | SAS | Msg-Passing |
|---|---|---|
| Explicit global data structure? | **Yes** | **No** |
| Assignment indept of data layout? | **Yes** | **No** |
| Communication | **Implicit** | **Explicit** |
| Synchronization | **Explicit** | **Implicit** |
| Explicit replication of border rows? | **No** | **Yes** |

- **Decomposition and Assignment similar in SAS and message-passing**
- **Orchestration is different**
  - **Data structures, data access/naming, communication, synchronization**
  - **Performance?**

# Summary

- Several techniques to parallelizing grid problems
- Specify as a series of difference relations
- Use of currently computer values can speed convergence
- Multigrid methods require specialized communication
- Understanding shared memory and message passing constraints for grid computation
  - Remember: software and communication!