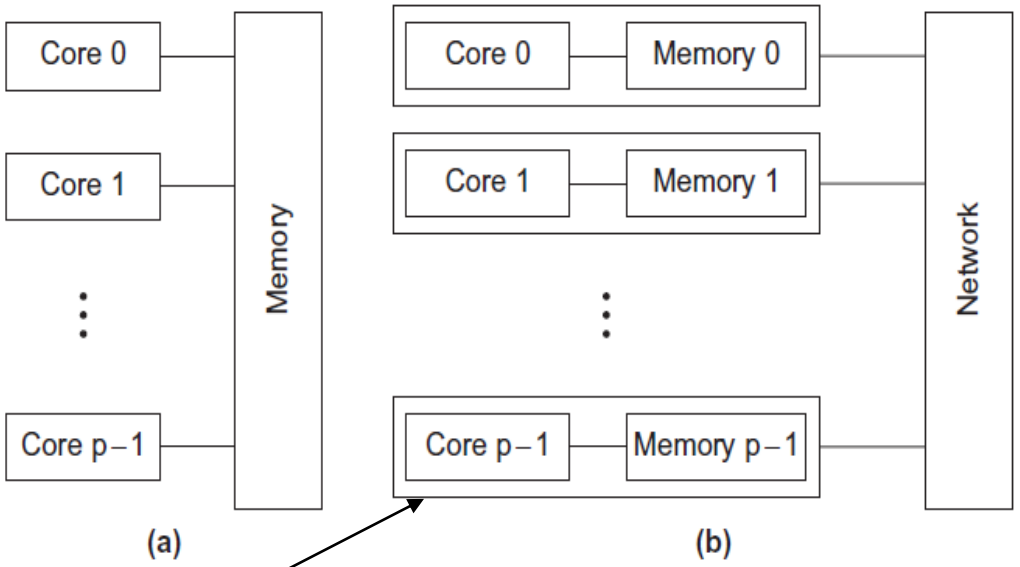
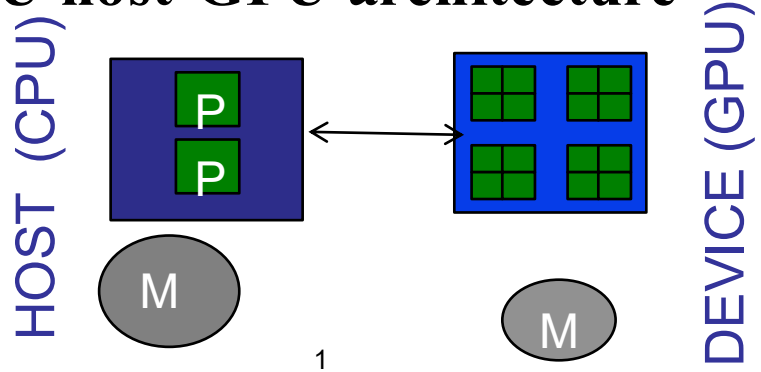
CS140 Summary: Parallel Architectures

- **Shared-memory**
  - Cache coherence
  - False sharing



- **Distributed-memory**
  - Topology, bisection bandwidth, networking cost.

CPU host-GPU architecture



Impact to paralel software: Hardware resource available (#cores). Memory location/access performance. Support coarse grain vs fine-grain paralleism?

Parallel Software

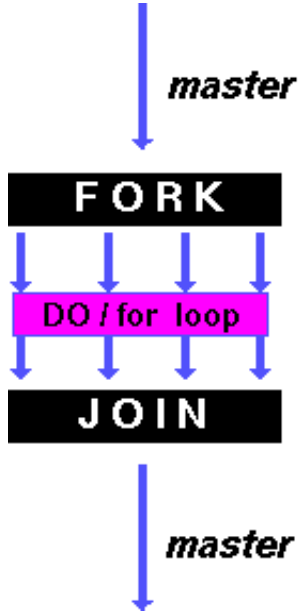
- **Task graph model. SPMD code**
  - Task granularity. Fine grain/coarse grain
  - Shared memory access or message passing
- **Performance evaluation**
  - Complexity analysis and performance assessment: Parallel time, speedup, efficiency
  - Maximum speedup
    - Length of critical path (longest path)
    - Degree of parallelism
    - Amdahl's law
- **Program analysis and transformation**
  - Dependence analysis and graph
  - Program/data partitioning with cyclic/block mapping. Task scheduling
  - Loop interchange, unrolling, blocking

Parallel Programming

- **Distributed memory programming with MPI**
  - Process-based parallelism. Communication primitives/algorithms
    - Micro/millisecond level overhead
    - Collective communications involve all the processes in a communicator with more overhead
  - Deadlock, unsafe issues in MPI
- **Shared memory programming with Pthreads**
  - Lightweight parallelism
  - Mutex locks, condition variables, barriers
    - Simple sync. cost can be in microseconds
  - False sharing, deadlock, thread safety
- **Shared memory programming with OpenMP**
- **GPU programming with Cuda**
- **Data-intensive parallel programming. Parallel I/O**
  - MapReduce: Operate on many key-value pairs on clustered machines/storage.
  - Spark operates on big lists (RDD)

Shared Memory Programming with OpenMP

- Fork-join thread parallelism with explicit/implicit synchronization
  - `pragma omp parallel`
  - `pragma omp parallel for`
  - `pragma omp parallel private ( i, x )`
  - `pragma omp atomic`
  - `pragma omp critical`
  - `pragma omp for reduction(+ : sum)`
- Loop scheduling options: static, runtime
- Make sure parallel region is thread-safe
- False sharing issue

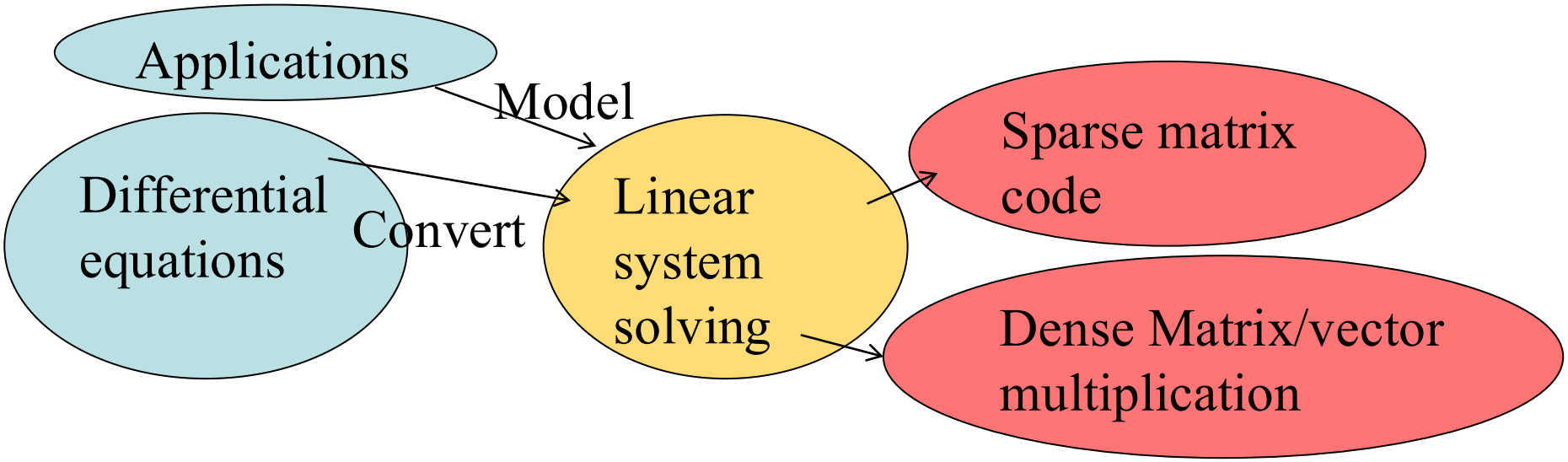


Heterogeneous data-parallel CPU+GPU execution

- **Thread scheduling in GUDA is block by block**, and warp by warp within each block.
  - Multiple blocks can run in parallel with multiple streaming multiprocessors.
- **Threads are organized as blocks of threads**, and blocks are grouped as grid.
  - 1D,2D, 3D naming for blocks/threads within a block
  - There is a limit on number of threads per block.
- **Memory layout for threads** include: global memory for all threads, shared memory for a block, and thread specific memory.
  - Understand performance bottleneck: host-GPU bandwidth (~single digits or 10 GB/s), global-memory-to-core bandwidth (~hundreds GB/s), shared memory size (~tens KB), #threads/block (~hundreds).

Basic Parallel Computing Algorithms

- **Basic operations:** Matrix/vector multiplication
- **Solving linear systems of equations**
  - Gaussian Elimination direct method for dense matrices
  - Jacobi/Gauss-Seidel iterative method
  - Convergence: diagonally dominant matrices
- **Use of iterative solver for Google PageRank**



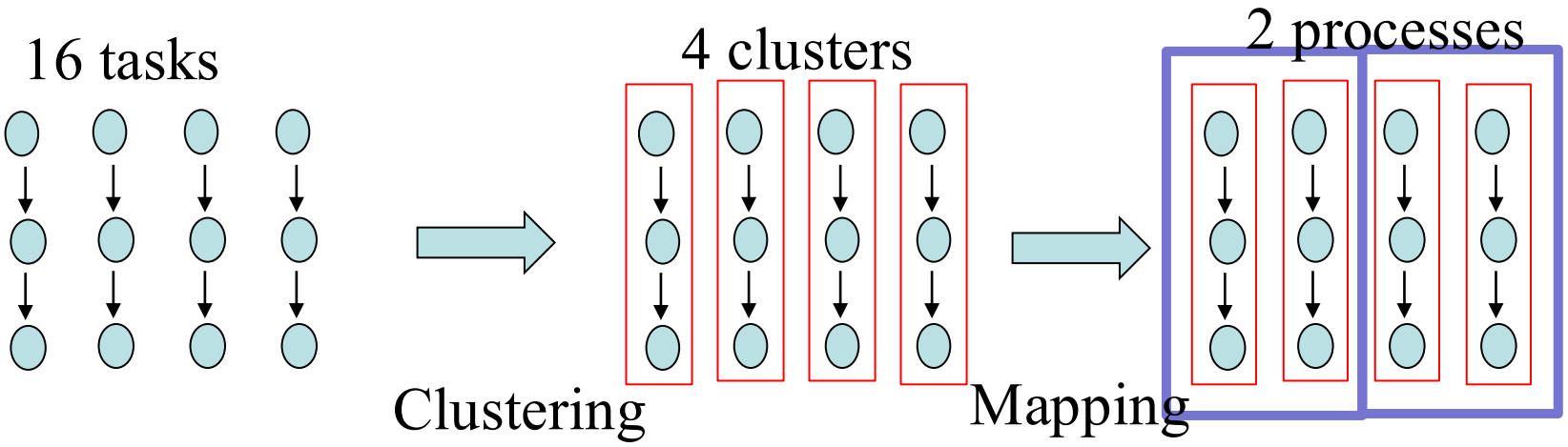
- **SGD for iterative model training in machine learning**

Key Program Parallelization Steps

1. **Form basic tasks and data units** operated by tasks with a complexity analysis to estimate task size
2. **Identify dependence among tasks**
   - Group or partition to adjust granularity → a new task graph
   - Each task receives data, runs sequentially, and then sends data. Communicate/synchronize at beginning and at end.
3. **Derive an execution schedule**
   - Cluster tasks if needed
   - Keep the targeted parallel platform/software in mind
4. **Use parallel software and synchronization primitives**
5. **Measure parallel performance**: time/speedup/efficiency
   - Is that reasonable? Maximum speedup (sequential time/critical path, degree of parallelism)

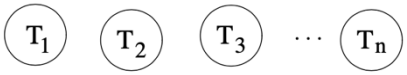
Strategies for Mapping and Scheduling

- **Option 1:** Directly map tasks to p processes (threads)
- **Option 2:**
  - Step 1. Map tasks to clusters.
    - Cluster tasks to reduce unnecessary communication/synchronization
    - If needed, assign data ownership (e.g. owner-compute rule)
  - Step 2. Map clusters to p processes (threads)

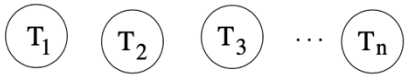


Examples

- **Matrix vector multiplication**



- **Matrix-matrix multiplication**
  - Two partitioning choices

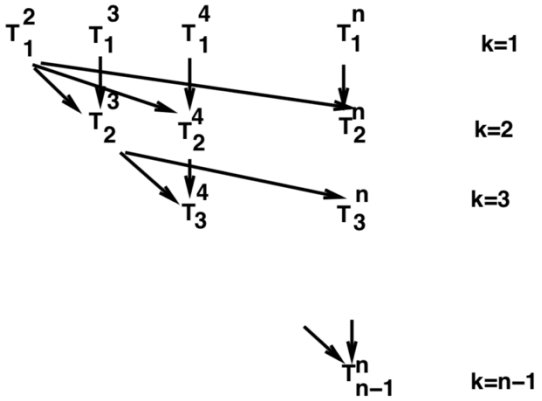


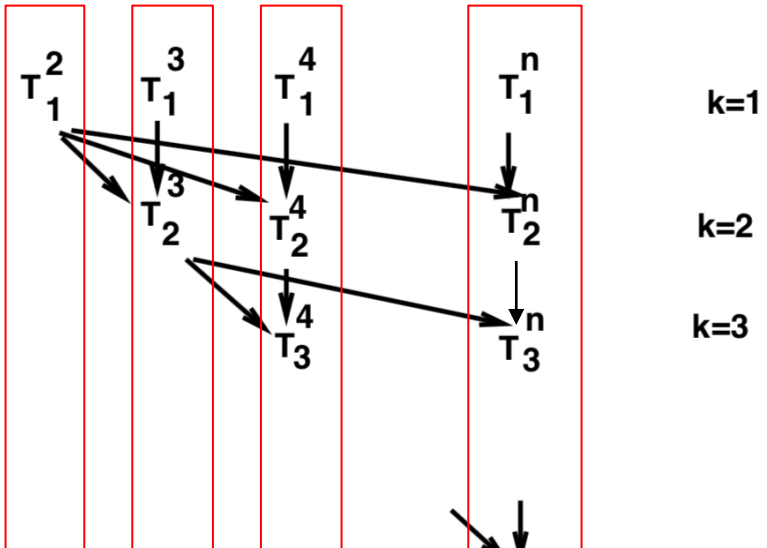
$$\begin{matrix} T_{1,1} & T_{1,2} & \cdots T_{1,n} \\ T_{2,1} & T_{2,2} & \cdots T_{2,n} \\ \cdots & & \\ T_{n,1} & T_{n,2} & \cdots T_{n,n} \end{matrix}$$

Clustering for scheduling

$$\boxed{\begin{matrix} T_{1,1} & T_{1,2} & \cdots T_{1,n} \end{matrix}}$$
$$\boxed{\begin{matrix} T_{2,1} & T_{2,2} & \cdots T_{2,n} \end{matrix}}$$
$$\boxed{\begin{matrix} T_{n,1} & T_{n,2} & \cdots T_{n,n} \end{matrix}}$$

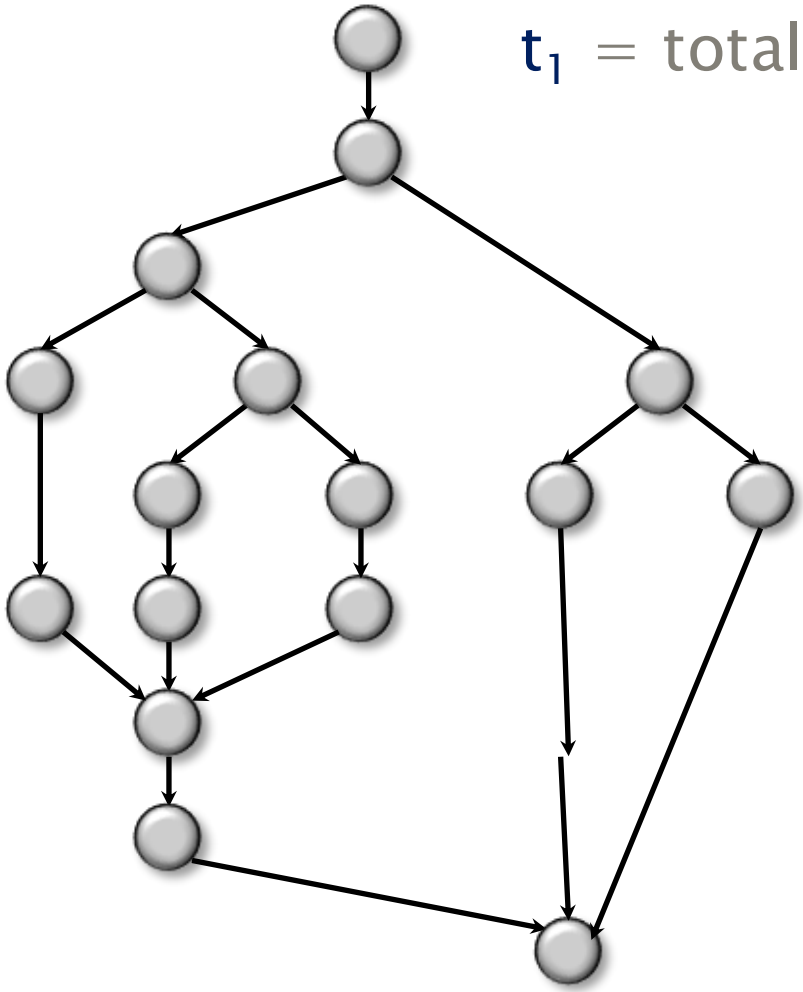
- **Solving linear systems of equations with GE**





Estimation for best parallel performance

t_p = execution time on p processors



t_1 = total *work*

t_∞ = *critical (longest) path length*

Minimum workload

- $t_p \geq t_1/p$

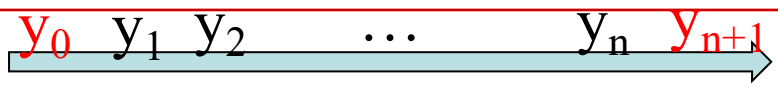
Limit by critical path length

- $t_p \geq t_\infty$

Speedup limit

$\leq \min(p,$ degree of parallelism, $t_1 / t_\infty)$

Jacobi and GS for iterative solving with 1D variables



- Solve *n* linear equations with 1D variables: $y_{i-1} - 2\,y_i + y_{i+1} = h^2$
- Iterative Jacobi method:

Repeat

  For $i = 1$ to n

    $y_i^{new} = 0.5(y_{i-1}^{old} + y_{i+1}^{old} - h^2)$

  Endfor

Until $\| \vec{y}^{new} - \vec{y}^{old} \| < \varepsilon$

$$y_0 = y_{n+1} = 0.$$

$$y_0 - 2y_1 + y_2 = h^2$$

$$y_1 - 2y_2 + y_3 = h^2$$

$$\vdots$$

$$y_{n-1} - 2y_n + y_{n+1} = h^2$$

Diagonally dominant: $2 \geq 1+1$

- What is dependence graph for each Jacobi iteration?
- How about Gauss-Seidel?

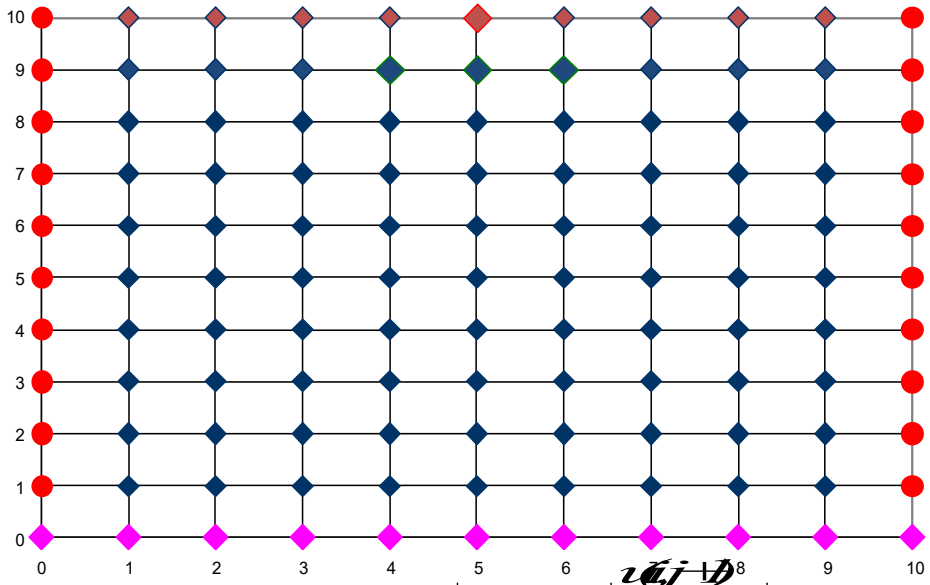
11

Jacobi and GS for iterative solving with 2D variables

- 91 variables in a 11x11 grid and the boundary values are known

$0 < i,j < 10$

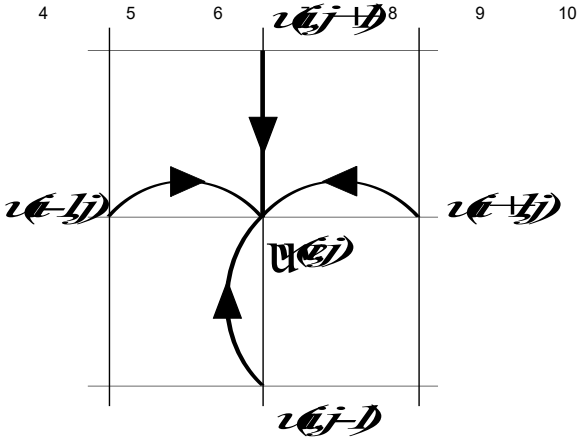
$$4u_{i,j} - u_{i,j-1} - u_{i,j+1} - u_{i-1,j} - u_{i+1,j} = 0$$



Solve using Jacobi or Gauss-Seidel

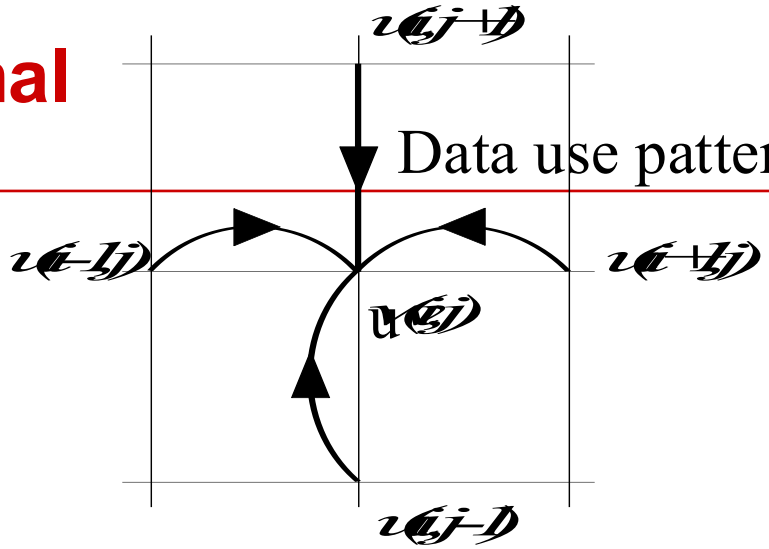
For i =1 to 9
  For j= 1 to 9

$$u_{i,j} = (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) / 4$$



Data use pattern

Jacobi on a two dimensional variable space



- **Jacobi method allows full parallelism, but slower convergence**

Repeat

  For i=1 to n

    For j=1 to n

$$u_{i,j}^{new} = 0.25(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}).$$

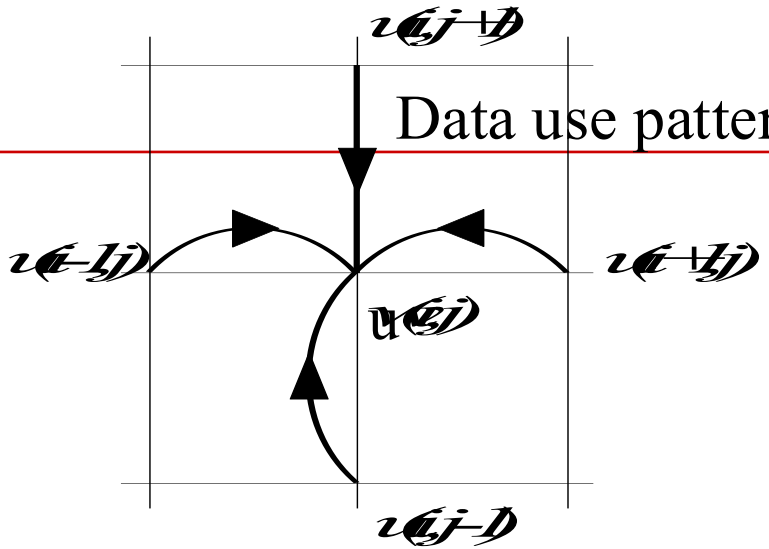
    EndFor

  EndFor

Until $\| u_{ij}^{new} - u_{ij} \| < \epsilon$

Gauss-Seidel on a 2D variable space

- **Faster convergence**



Repeat

$u^{old} = u.$

For i=1 to n

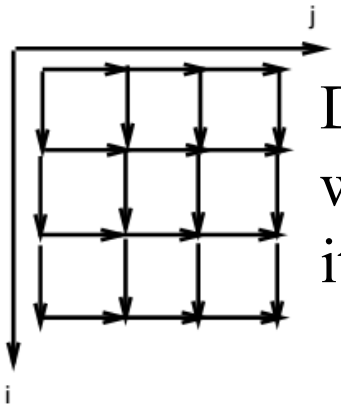
For j=1 to n

$u_{i,j} = 0.25(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}).$

EndFor

EndFor

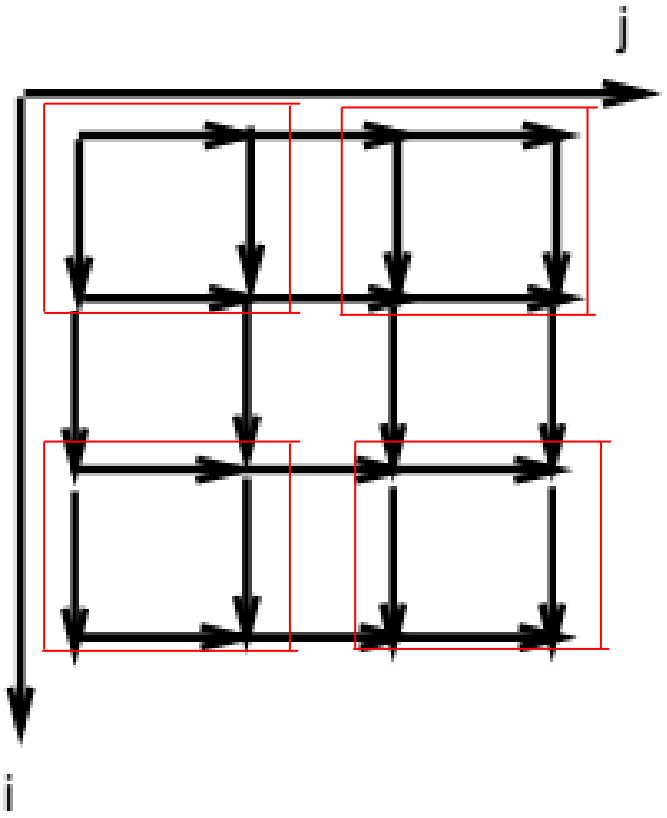
Until $\| u_{ij} - u_{ij}^{old} \| < \epsilon$

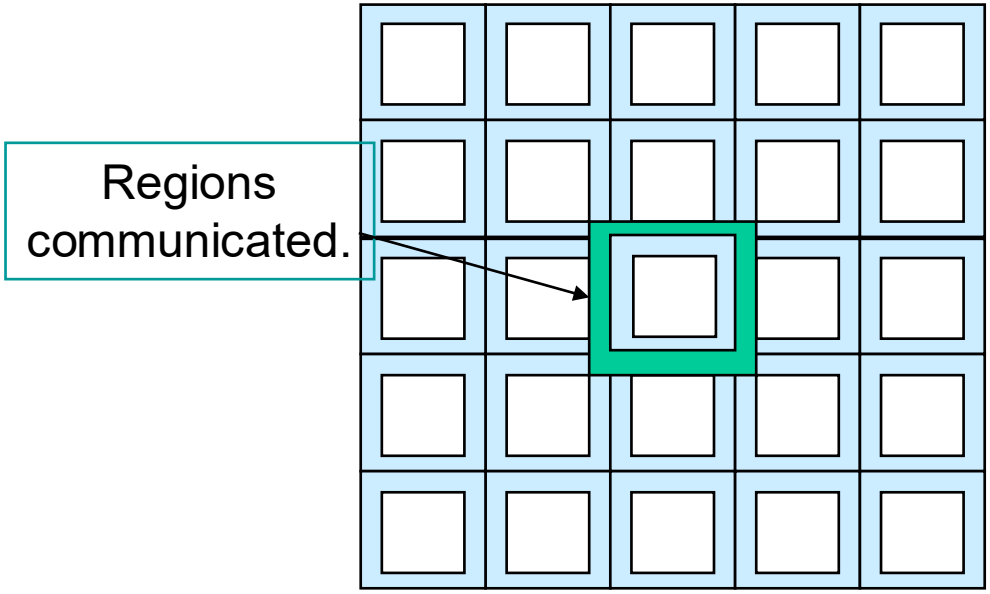


Dependence within each GS iteration

Coarse grain partitioning for parallelization with MPI: distributing data among processes

- **Partition a large grid into a mesh of large blocks**
  - Communicate boundary points boundary to neighboring processes.





Open-book Final Exam Format

- 3 hours. 26-30 questions.
- Multi-choice questions and few short-answer questions
  - MPI, Pthread, OpenMP code
  - GPU CUDA programming
  - Parallel program design, partitioning, dependence, scheduling, best parallel performance, cost estimation (complexity), parallelization with MPI or threads
  - Solving linear equations with GE, Jacobi and Gauss Seidel
  - Iterative computing applications:
    - Google PageRank
    - Model training with SGD
  - MapReduce Spark
  - ML Applications: Transformers

Course Evaluation

- https://my-ucsb.bluera.com/

  Open until midnight of this Friday, March 13

- If you like this course, please give the thumbs up to support.
  - Your feedback is appreciated, including positive feedback so I will continue in the future