# Parallel Jacobi Algorithm

Steven Dong
Applied Mathematics

# Overview

- Parallel Jacobi Algorithm
  - Different data distribution schemes
    - Row-wise distribution
    - Column-wise distribution
    - Cyclic shifting
    - Global reduction
  - Domain decomposition for solving Laplacian equations

- Related MPI functions for parallel Jacobi algorithm and your project.

# Linear Equation Solvers

- Direct solvers
  - Gauss elimination
  - LU decomposition
- Iterative solvers
  - Basic iterative solvers
    - Jacobi
    - Gauss-Seidel
    - Successive over-relaxation
  - Krylov subspace methods
    - Generalized minimum residual (GMRES)
    - Conjugate gradient

# Sequential Jacobi Algorithm

$$Ax = b$$

$$A = D + L + U$$

D is diagonal matrix

L is lower triangular matrix

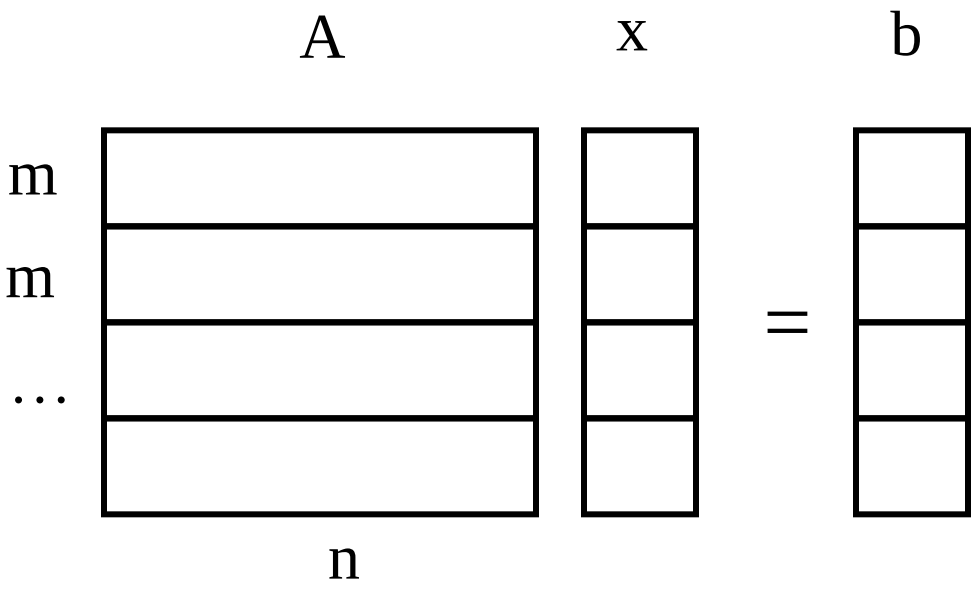
U is upper triangular matrix

$$x^{k+1} = D^{-1}(b - (L + U)x^{k})$$

# Parallel Jacobi Algorithm: Ideas

- Shared memory or distributed memory:
  - Shared-memory parallelization very straightforward
  - Consider distributed memory machine using MPI
- Questions to answer in parallelization:
  - Identify concurrency
  - Data distribution (data locality)
    - How to distribute coefficient matrix among CPUs?
    - How to distribute vector of unknowns?
    - How to distribute RHS?
  - Communication: What data needs to be communicated?
- Want to:
  - Achieve data locality
  - Minimize the number of communications
  - Overlap communications with computations
  - Load balance

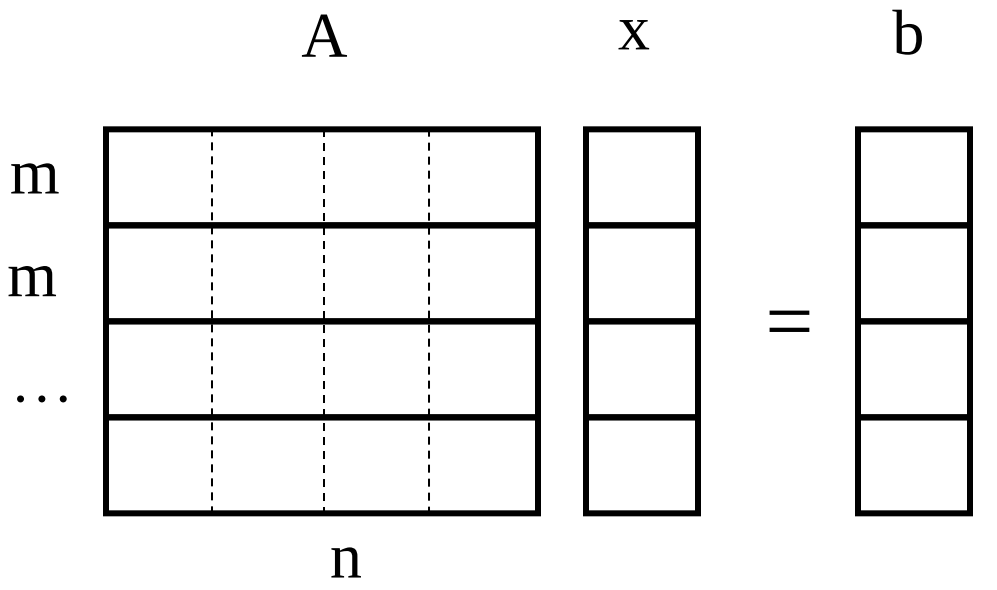
# Row-wise Distribution




A: n x n
m = n/P
P is number of CPUs

- Assume dimension of matrix can be divided by number of CPUs
- Blocks of m rows of coefficient matrix distributed to different CPUs;
- Vector of unknowns and RHS distributed similarly

# Data to be Communicated

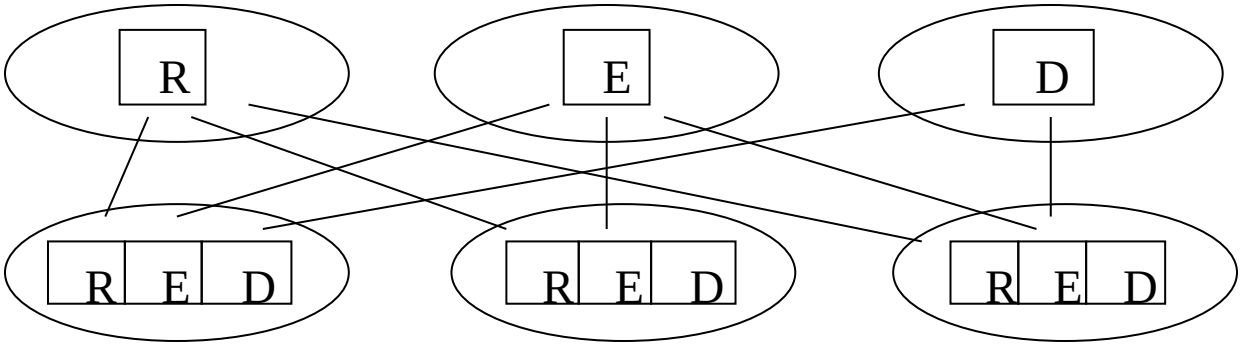



$$x^{k+1} = D^{-1}(b - (L+U)x^k)$$

- Already have all columns of matrix A on each CPU;
- Only part of vector x is available on a CPU; Cannot carry out matrix vector multiplication directly;
- Need to communicate the vector x in the computations.

# How to Communicate Vector X?

- Gather partial vector x on each CPU to form the whole vector; Then matrix-vector multiplication on different CPUs proceed independently. (textbook)

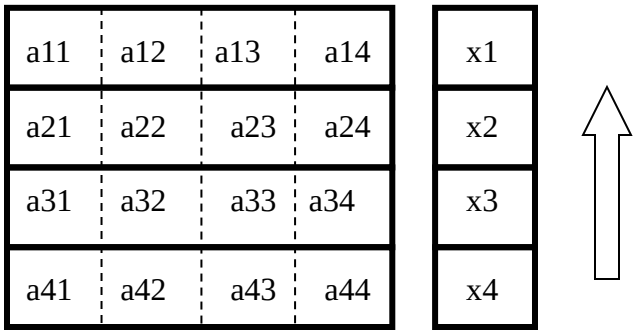



- Need MPI_Allgather() function call;
- Simple to implement, but
  - A lot of communications
  - Does not scale well for a large number of processors.

# How to Communicate X?

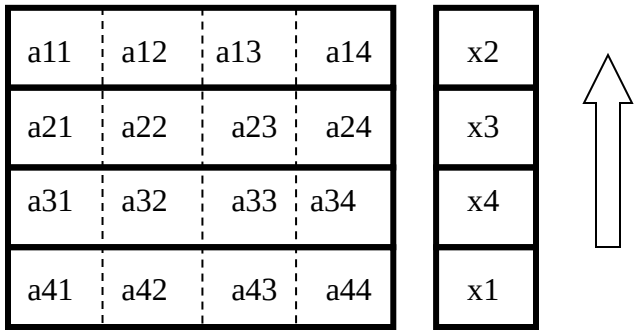
- Another method: Cyclic shift
  - Shift partial vector x upward at each step;
  - Do partial matrix-vector multiplication on each CPU at each step;
  - After P steps (P is the number of CPUs), the overall matrix-vector multiplication is complete.
- Each CPU needs only to communicate with neighboring CPUs
  - Provides opportunities to overlap communication with computations
- Detailed illustration …

(1)



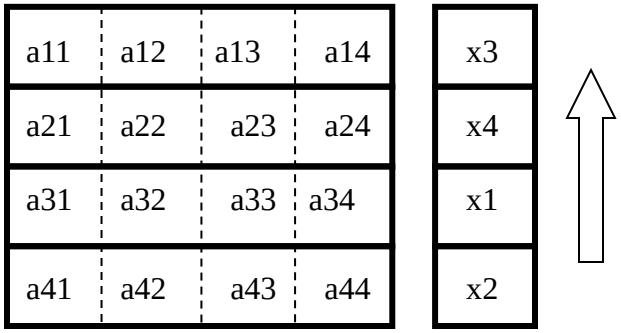

a11*x1 + a12*x2 + a13*x3 + a14*x4 Cpu 0
a21*x1 + a22*x2 + a23*x3 + a24*x4 Cpu 1
a31*x1 + a32*x2 + a33*x3 + a34*x4 Cpu 2
a41*x1 + a42*x2 + a43*x3 + a44*x4 Cpu 3

(2)



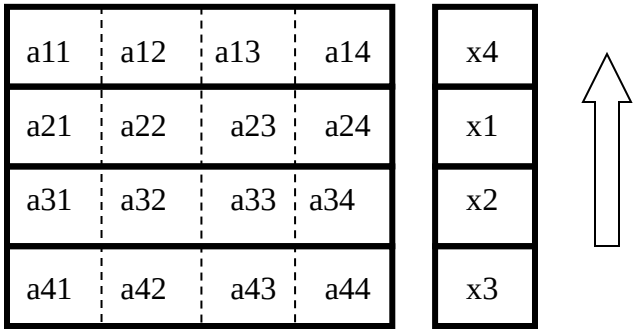

a11*x1 + a12*x2 + a13*x3 + a14*x4
a21*x1 + a22*x2 + a23*x3 + a24*x4
a31*x1 + a32*x2 + a33*x3 + a34*x4
a41*x1 + a42*x2 + a43*x3 + a44*x4

(3)




a11*x1 + a12*x2 + a13*x3 + a14*x4
a21*x1 + a22*x2 + a23*x3 + a24*x4
a31*x1 + a32*x2 + a33*x3 + a34*x4
a41*x1 + a42*x2 + a43*x3 + a44*x4

(4)




a11*x1 + a12*x2 + a13*x3 + a14*x4
a21*x1 + a22*x2 + a23*x3 + a24*x4
a31*x1 + a32*x2 + a33*x3 + a34*x4
a41*x1 + a42*x2 + a43*x3 + a44*x4

# Overlap Communications with Computations

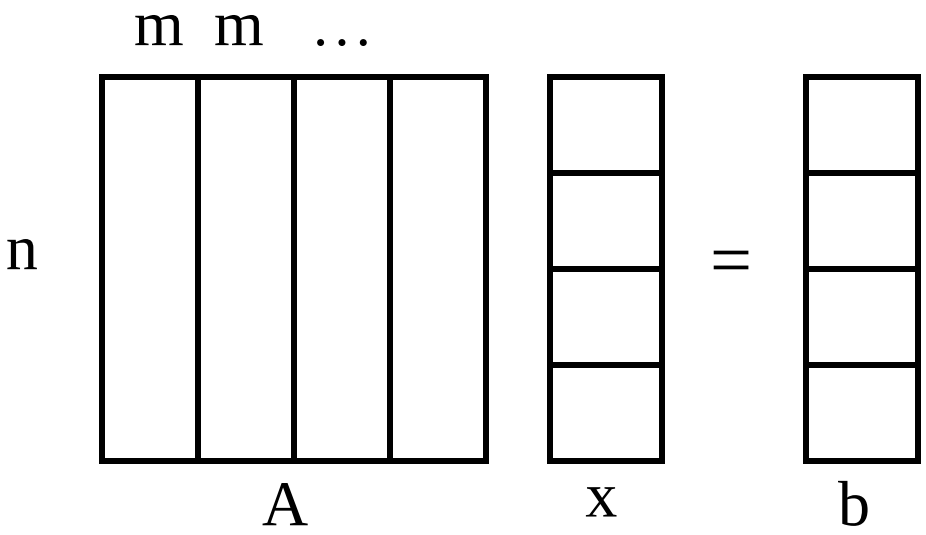
- Communications:
  - Each CPU needs to send its own partial vector x to upper neighboring CPU;
  - Each CPU needs to receive data from lower neighboring CPU
- Overlap communications with computations: Each CPU does the following:
  - Post non-blocking requests to send data to upper neighbor to to receive data from lower neighbor; This returns immediately
  - Do partial computation with data currently available;
  - Check non-blocking communication status; wait if necessary;
  - Repeat above steps

# Stopping Criterion

$$\left\| x^{k+1} - x^{k} \right\| < \varepsilon \left\| b \right\| \qquad \left\| \vec{A} - \vec{B} \right\| = \sqrt{\sum_{i} (A_i - B_i)^2}$$

- Computing norm requires information of the whole vector;
- Need a global reduction (SUM) to compute the norm using MPI_Allreduce or MPI_Reduce.

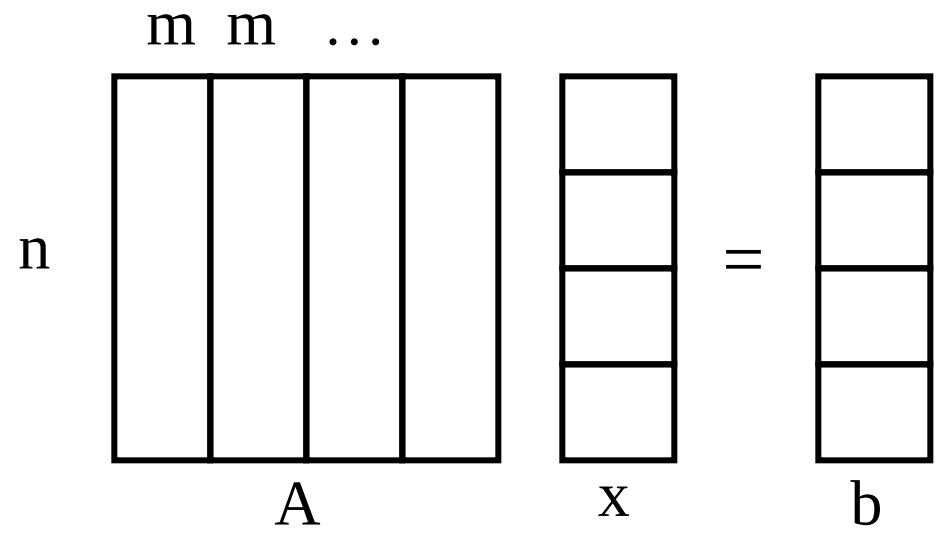
# Column-wise Distribution




$$x^{k+1} = D^{-1}(b - (L+U)x^k)$$

- Blocks of m columns of coefficient matrix A are distributed to different CPUs;
- Blocks of m rows of vector x and b are distributed to different CPUs;

# Data to be Communicated



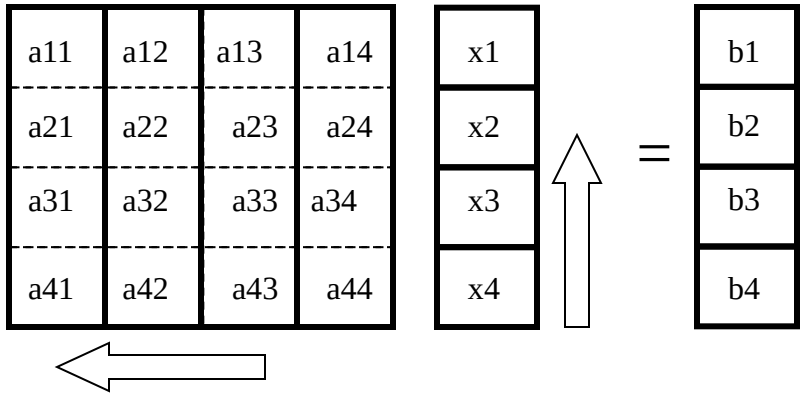

- Already have coefficient matrix data of m columns, and a block of m rows of vector x;
- So a partial A*x can be computed on each CPU independently.
- Need communication to get whole A*x;

# How to Communicate

- After getting partial A*x, can do global reduction (SUM) using MPI_Allreduce to get the whole A*x. So a new vector x can be calculated.

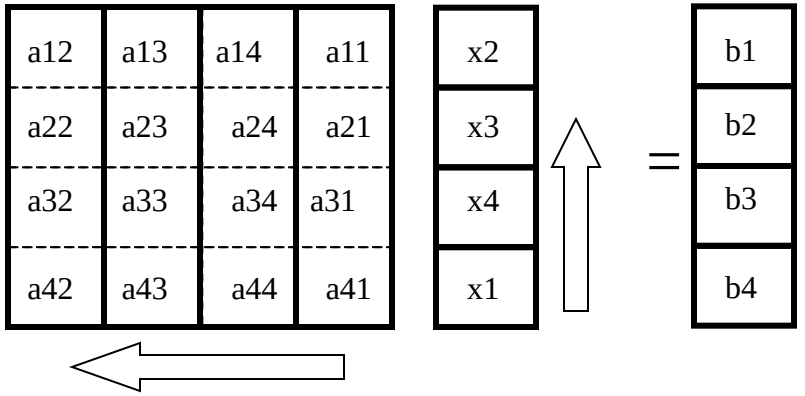
- Another method: Cyclic shift
  - Shift coefficient matrix left-ward and vector of unknowns upward at each step;
  - Do a partial matrix-vector multiplication, and subtract it from the RHS;
  - After P steps (P is number of CPUs), matrix-vector multiplication is completed and subtracted from RHS; Can compute new vector x.
- Detailed illustration …

(1)



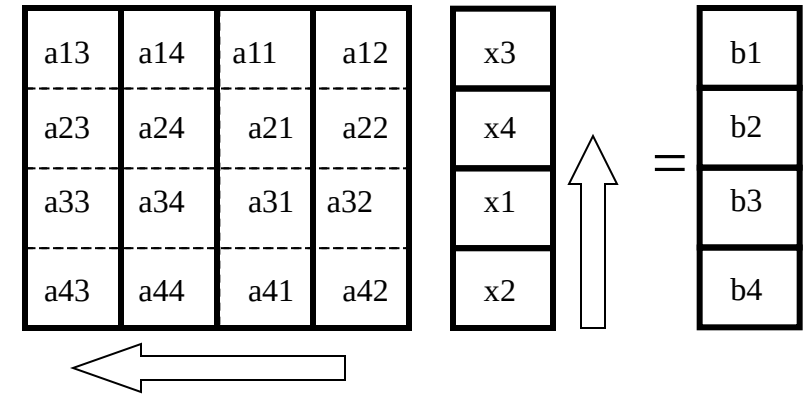

b1 - a11*x1 - a12*x2 - a13*x3 - a14*x4
b2 - a21*x1 - a22*x2 - a23*x3 - a24*x4
b3 - a31*x1 - a32*x2 - a33*x3 - a34*x4
b4 - a41*x1 - a42*x2 - a43*x3 - a44*x4

(2)



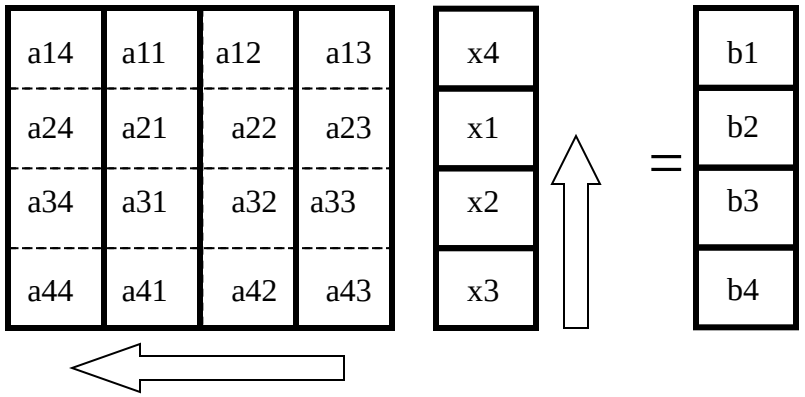

b1 - a11*x1 - a12*x2 - a13*x3 - a14*x4
b2 - a21*x1 - a22*x2 - a23*x3 - a24*x4
b3 - a31*x1 - a32*x2 - a33*x3 - a34*x4
b4 - a41*x1 - a42*x2 - a43*x3 - a44*x4

(3)




b1 - a11*x1 - a12*x2 - a13*x3 - a14*x4
b2 - a21*x1 - a22*x2 - a23*x3 - a24*x4
b3 - a31*x1 - a32*x2 - a33*x3 - a34*x4
b4 - a41*x1 - a42*x2 - a43*x3 - a44*x4

(4)




b1 - a11*x1 - a12*x2 - a13*x3 - a14*x4
b2 - a21*x1 - a22*x2 - a23*x3 - a24*x4
b3 - a31*x1 - a32*x2 - a33*x3 - a34*x4
b4 - a41*x1 - a42*x2 - a43*x3 - a44*x4

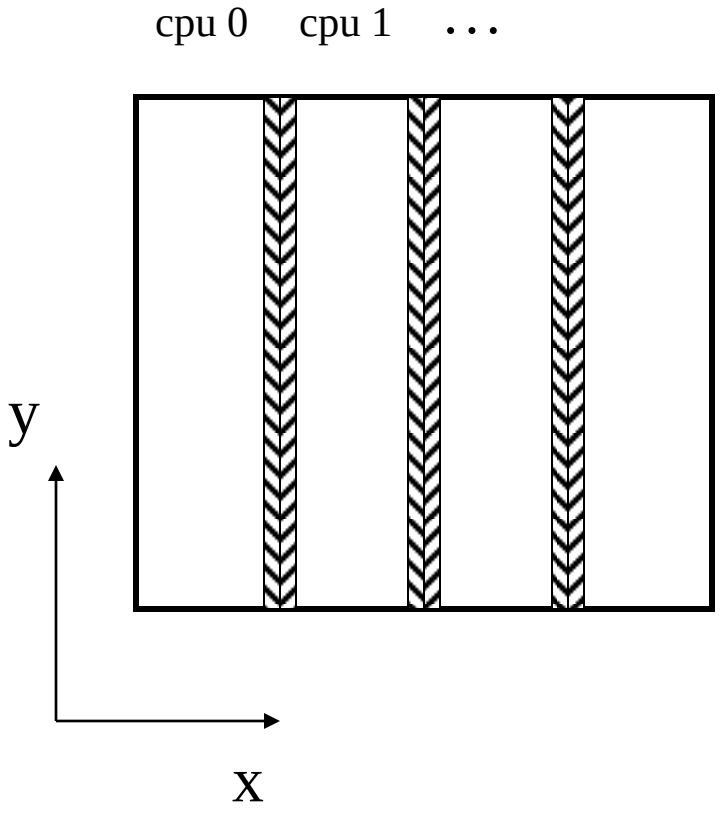
# Solving Diffusion Equation

$$\nabla^2 f + q = 0$$

$$f_{ij} = \frac{1}{4}\left( f_{i-1\,j} + f_{i+1\,j} + f_{ij-1} + f_{ij+1} + \Delta x^2 q_{ij} \right)$$

- How do we solve it in parallel in practice?
- Need to do domain decomposition.

# Domain Decomposition



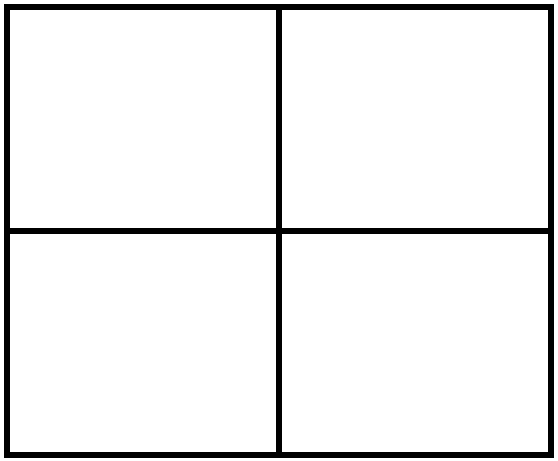

- Column-wise decomposition
- Boundary points depend on data from neighboring CPU
  - During each iteration, need send own boundary data to neighbors, and receive boundary data from neighboring CPUs.
- Interior points depend only on data residing on the same CPU (local data).

# Overlap Communication with Computations

- Compute boundary points and interior points at different stages;
- Specifically:
  - At the beginning of an iteration, post non-blocking send to and receive from requests for communicating boundary data with neighboring CPUs;
  - Update values on interior points;
  - Check communication status (should complete by this point), wait if necessary;
  - Boundary data received, update boundary points;
  - Begin next iteration, repeat above steps.

# Other Domain Decompositions

1D decomposition



2D decomposition

# Related MPI Functions for Parallel Jacobi Algorithm

- MPI_Allgather()
- MPI_Isend()
- MPI_Irecv()
- MPI_Reduce()
- MPI_Allreduce()

# MPI Programming Related to Your Project

- Parallel Jacobi Algorithm
- Compiling MPI programs
- Running MPI programs
- Machines: www.cascv.brown.edu