CS240A: Conjugate Gradients and the Model Problem

The middleware of scientific computing

Continuous physical modeling

↓

Linear algebra

↓

Computers

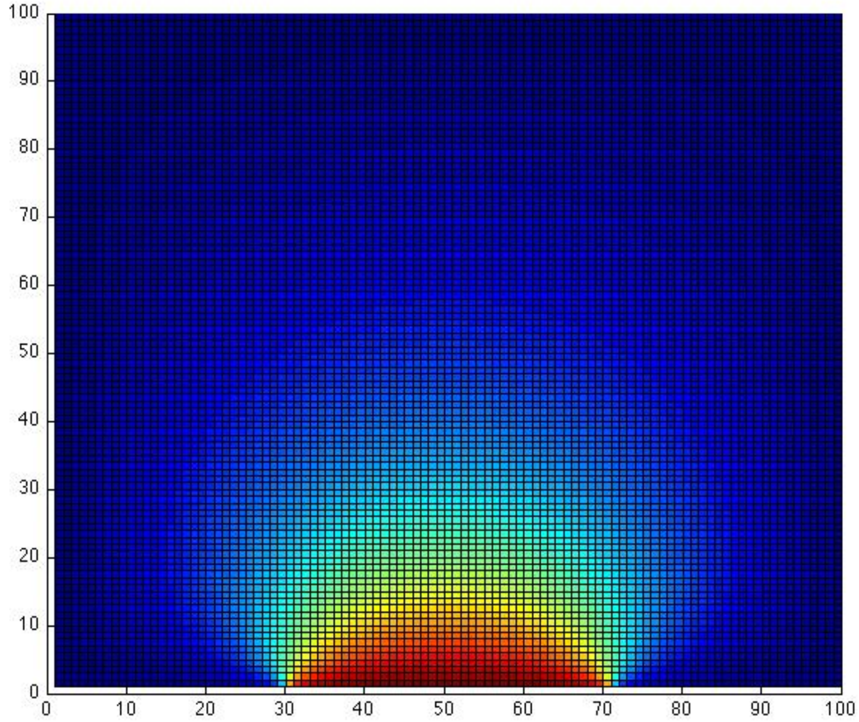
$$Ax = b$$

Example: The Temperature Problem

- A cabin in the snow
- Wall temperature is 0°, except for a radiator at 100°
- What is the temperature in the interior?

Example: The Temperature Problem

- A cabin in the snow (a square region ☺)
- Wall temperature is 0°, except for a radiator at 100°
- What is the temperature in the interior?



The physics: Poisson's equation

$$\nabla^2 u(x,y) \equiv \frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y)$$

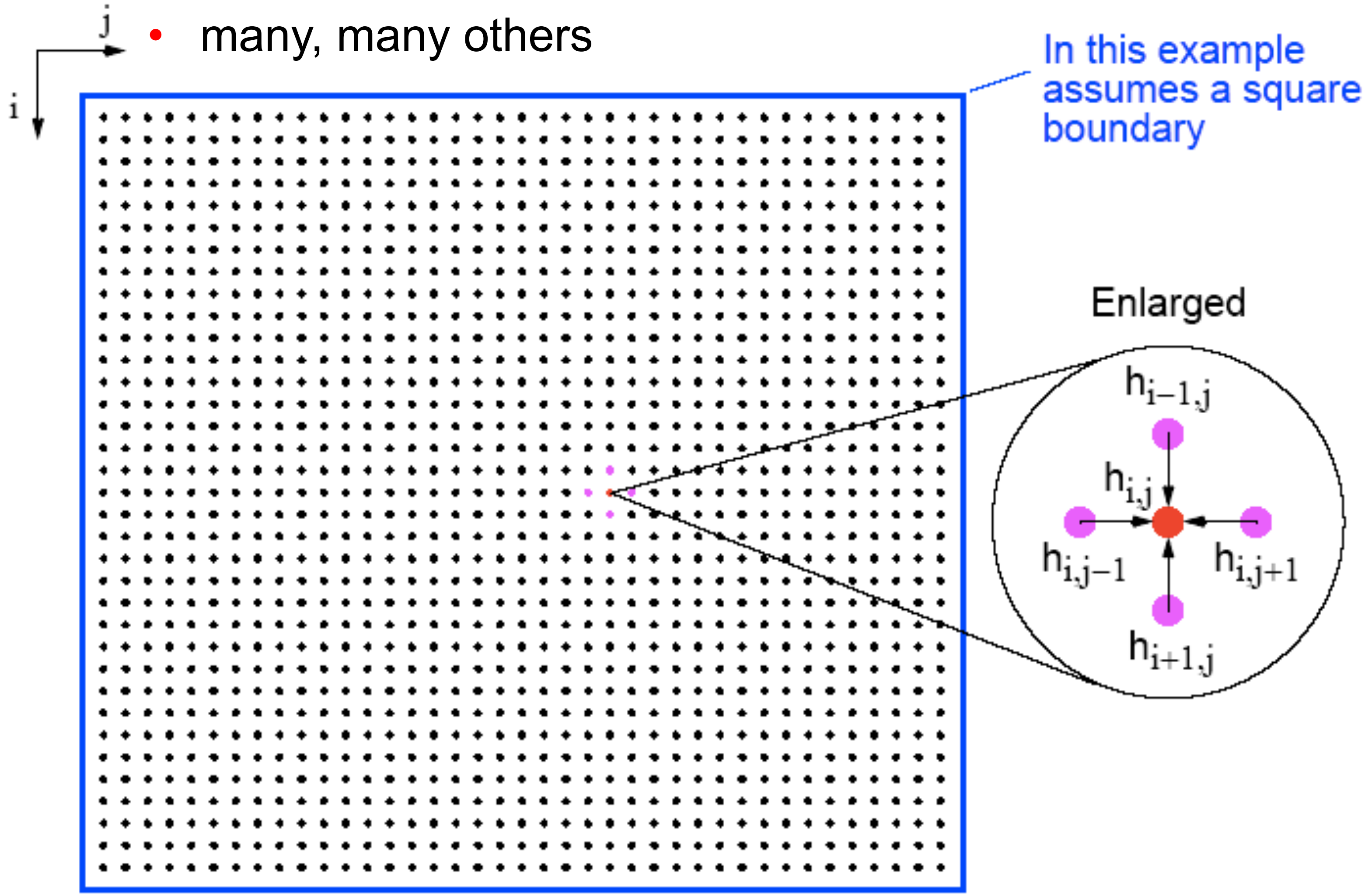
for $(x,y) \in R = \{\ (x,y)\ |\ a < x < b,\ \ c < y < d\ \}$, and

$$u(x,y) = g(x,y)$$

for (x,y) on the boundary of R.

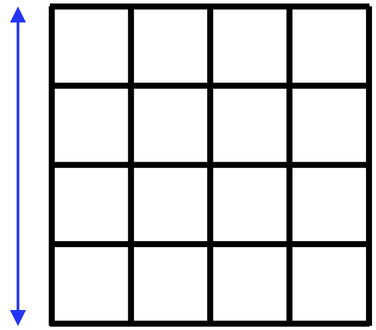
Many Physical Models Use Stencil Computations

- PDE models of heat, fluids, structures, …
- Weather, airplanes, bridges, bones, …
- Game of Life
- many, many others



Model Problem: Solving Poisson's equation for temperature

$k = n^{1/2}$



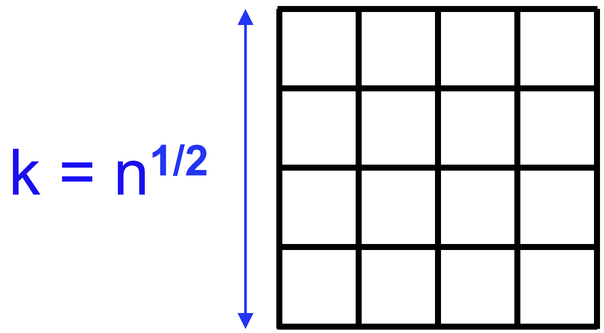
- Discrete approximation to Poisson's equation:

$$t(i) = \tfrac{1}{4} \left(t(i-k) + t(i-1) + t(i+1) + t(i+k) \right)$$

- Intuitively:

Temperature at a point is the average
of the temperatures at surrounding points

Model Problem: Solving Poisson's equation for temperature

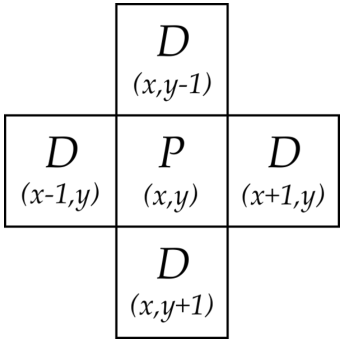


- For each i from 1 to n, except on the boundaries:

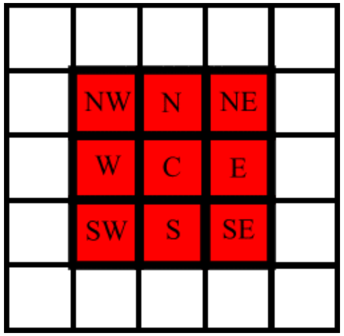
$$-\, t(i-k) - t(i-1) + 4*t(i) - t(i+1) - t(i+k) = 0$$

- n equations in n unknowns: A*t = b
- Each row of A has at most 5 nonzeros
- In three dimensions, $k = n^{1/3}$ and each row has at most 7 nzs

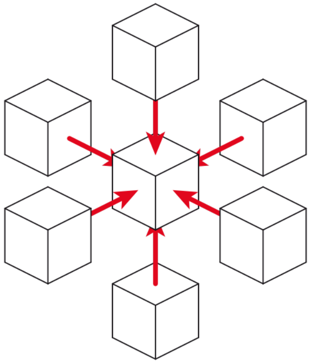
Examples of stencils



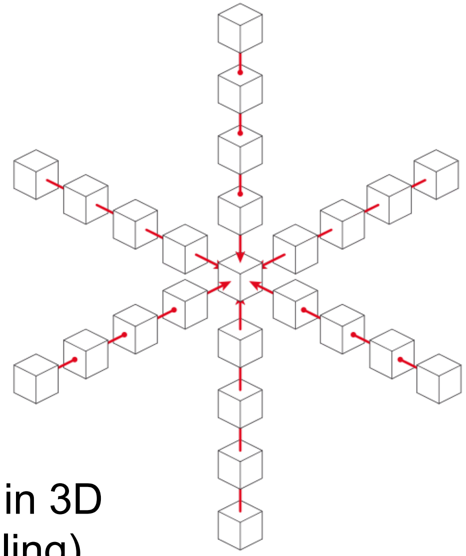
5-point stencil in 2D
(temperature problem)



9-point stencil in 2D
(game of Life)



7-point stencil in 3D
(3D temperature problem)

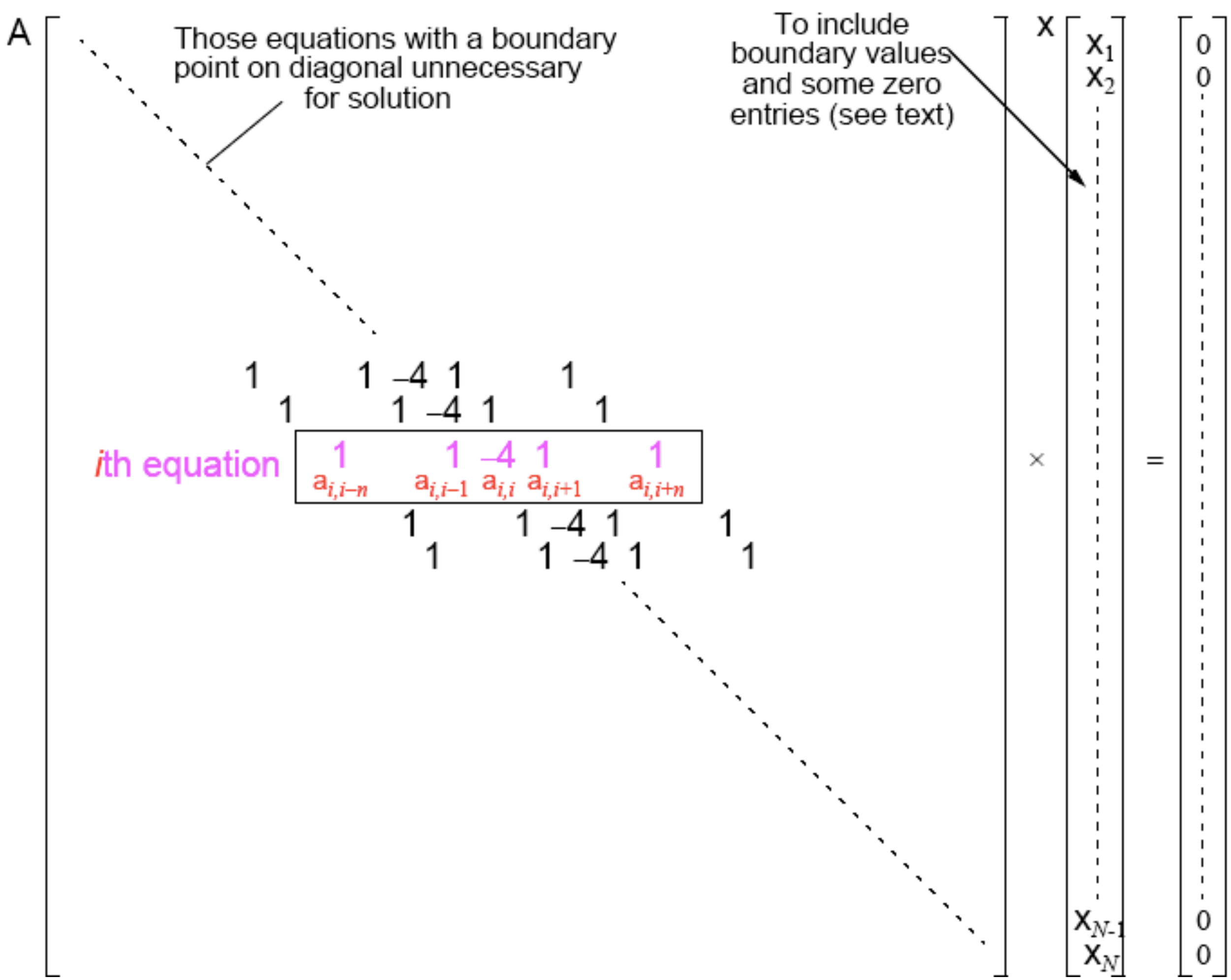


25-point stencil in 3D
(seismic modeling)

… and many more

A Stencil Computation Solves a System of Linear Equations

- Solve $Ax = b$ for x
- Matrix A, right-hand side vector b, unknown vector x
- A is *sparse*: most of the entries are 0



The Landscape of Ax=b Solvers

	Direct $A = LU$	Iterative $y' = Ay$
Non-symmetric	**Pivoting LU**	**GMRES, BiCGSTAB, …**
Symmetric positive definite	**Cholesky**	**Conjugate gradient**

More General ↕ **More Robust**

More Robust ⟷ **Less Storage (if sparse)**

CS 240A: Solving Ax = b in parallel

- Dense A: Gaussian elimination with partial pivoting (LU)
  - See Jim Demmel's slides
  - Same flavor as matrix * matrix, but more complicated
- Sparse A: Iterative methods – Conjugate gradient, etc.
  - Sparse matrix times dense vector
- Sparse A: Gaussian elimination – Cholesky, LU, etc.
  - Graph algorithms
- Sparse A: Preconditioned iterative methods and multigrid
  - Mixture of lots of things

CS 240A: Solving Ax = b in parallel

- Dense A: Gaussian elimination with partial pivoting
  - See Jim Demmel' s slides
  - Same flavor as matrix * matrix, but more complicated

- Sparse A: Iterative methods – Conjugate gradient etc.
  - Sparse matrix times dense vector

- Sparse A: Gaussian elimination – Cholesky, LU, etc.
  - Graph algorithms

- Sparse A: Preconditioned iterative methods and multigrid
  - Mixture of lots of things

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

<u>**for**</u> $k = 1, 2, 3, \ldots$

$x_k = x_{k-1} + \ldots$ new approx solution

$r_k = \ldots$ new residual

$d_k = \ldots$ new search direction

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

for $k = 1, 2, 3, \ldots$

$\alpha_k = \ldots$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ new approx solution

$r_k = \ldots$ new residual

$d_k = \ldots$ new search direction

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

for $k = 1, 2, 3, \ldots$

$\alpha_k = (r^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ new approx solution

$r_k = \ldots$ new residual

$d_k = \ldots$ new search direction

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

for $k = 1, 2, 3, \ldots$

$\alpha_k = (r^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ new approx solution

$r_k = \ldots$ new residual

$\beta_k = (r^T_k r_k) / (r^T_{k-1} r_{k-1})$

$d_k = r_k + \beta_k d_{k-1}$ new search direction

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

<u>for</u> $k = 1, 2, 3, \ldots$

$\alpha_k = (r^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ new approx solution

$r_k = r_{k-1} - \alpha_k A d_{k-1}$ new residual

$\beta_k = (r^T_k r_k) / (r^T_{k-1} r_{k-1})$

$d_k = r_k + \beta_k d_{k-1}$ new search direction

Conjugate gradient iteration to solve A*x=b

$x_0 = 0, \quad r_0 = b, \quad d_0 = r_0$ (these are all vectors)

for $k = 1, 2, 3, \ldots$

$\alpha_k = (r^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ approximate solution

$r_k = r_{k-1} - \alpha_k A d_{k-1}$ residual $= b - Ax_k$

$\beta_k = (r^T_k r_k) / (r^T_{k-1} r_{k-1})$ improvement

$d_k = r_k + \beta_k d_{k-1}$ search direction

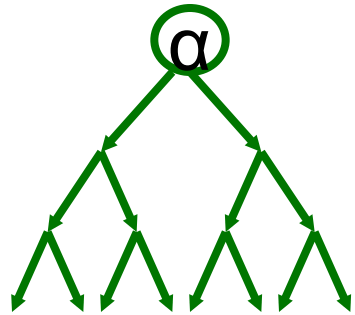
- One matrix-vector multiplication per iteration
- Two vector dot products per iteration
- Four n-vectors of working storage

Vector and matrix primitives for CG

- DAXPY: $v = \alpha * v + \beta * w$ (vectors v, w; scalars α, β)
  - Broadcast the scalars α and β, then independent * and +
  - comm volume = 2p, span = log n
- DDOT: $\alpha = v^T * w = \Sigma_j v[j] * w[j]$ (vectors v, w; scalar α)
  - Independent *, then + reduction
  - comm volume = p, span = log n
- Matvec: $v = A * w$ (matrix A, vectors v, w)
  - The hard part
  - But all you need is a subroutine to compute v from w
  - Sometimes you don't need to store A (e.g. temperature problem)
  - Usually you do need to store A, but it's *sparse* ...

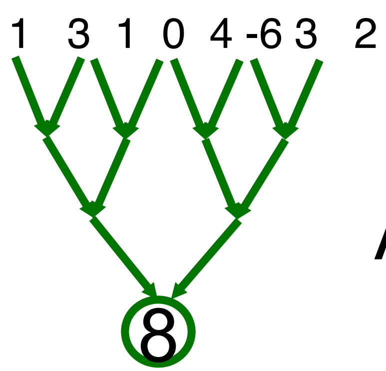
Broadcast and reduction

- Broadcast of 1 value to p processors in log p time



Broadcast

- Reduction of p values to 1 in log p time
- Takes advantage of associativity in +, *, min, max, etc.



Add-reduction

Where's the data (temperature problem)?

- The matrix A: Nowhere!!

- The vectors x, b, r, d:
  - Each vector is one value per stencil point
  - Divide stencil points among processors, n/p points each

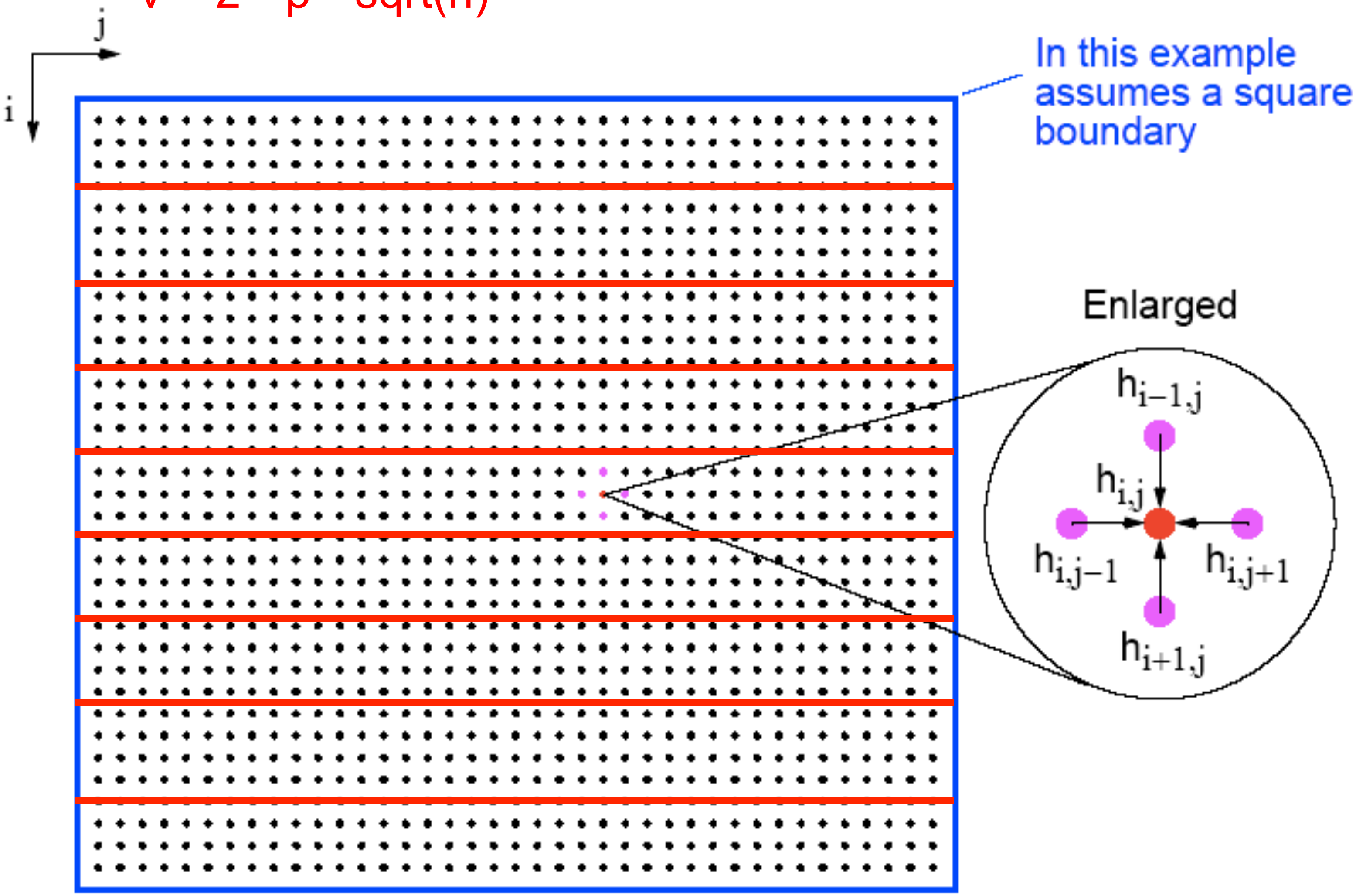
- How do you divide up the sqrt(n) by sqrt(n) region of points?

- Block row (or block col) layout: v = 2 * p * sqrt(n)

- 2-dimensional block layout: v = 4 * sqrt(p) * sqrt(n)

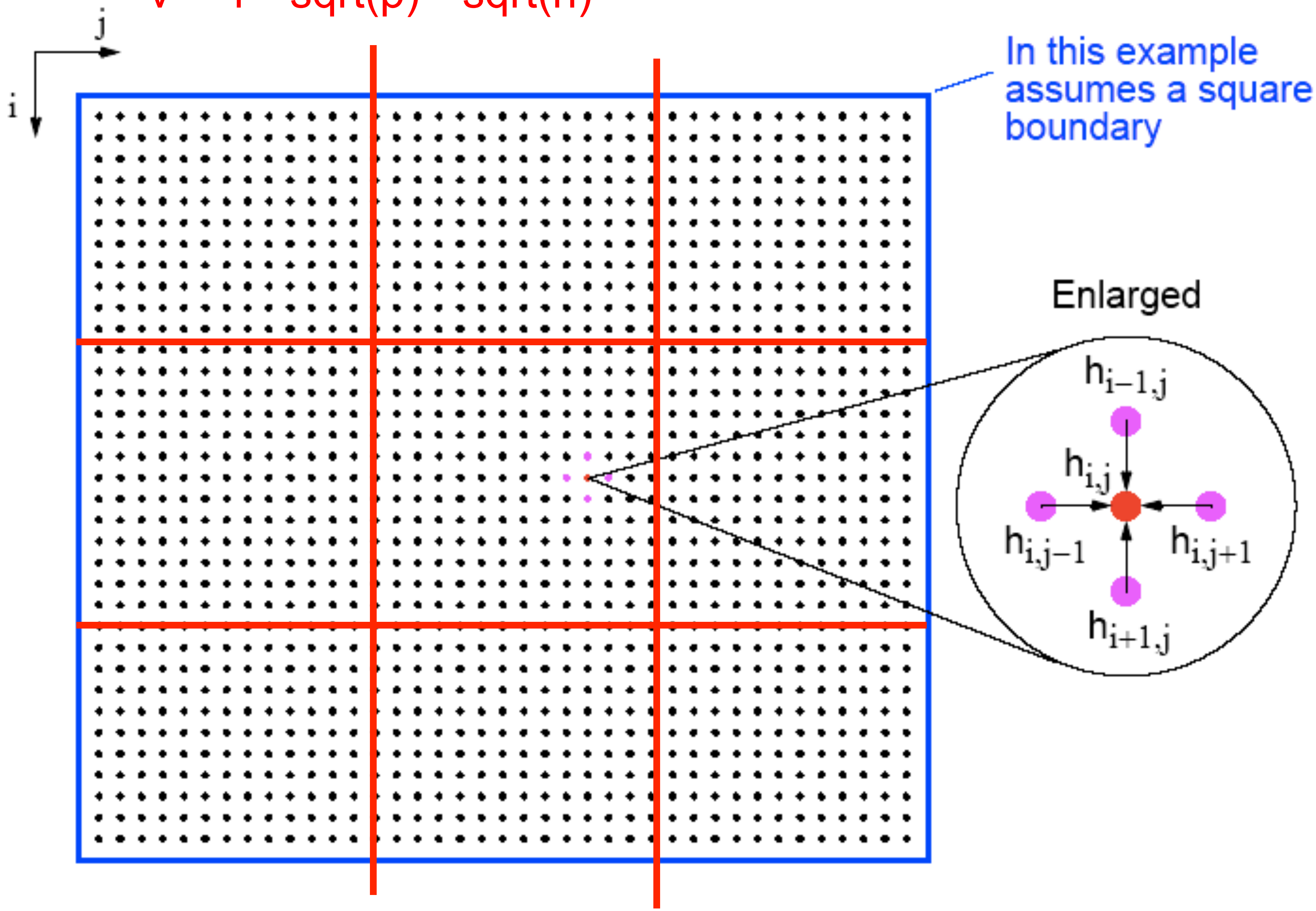
How do you partition the sqrt(n) by sqrt(n) stencil points?

- First version: number the grid by rows
- Leads to a block row decomposition of the region
- v = 2 * p * sqrt(n)



How do you partition the sqrt(n) by sqrt(n) stencil points?

- Second version: 2D block decomposition
- Numbering is a little more complicated
- v = 4 * sqrt(p) * sqrt(n)



Where's the data (temperature problem)?

- The matrix A: Nowhere!!
- The vectors x, b, r, d:
  - Each vector is one value per stencil point
  - Divide stencil points among processors, n/p points each
- How do you divide up the sqrt(n) by sqrt(n) region of points?
- Block row (or block col) layout: v = 2 * p * sqrt(n)
- 2-dimensional block layout: v = 4 * sqrt(p) * sqrt(n)

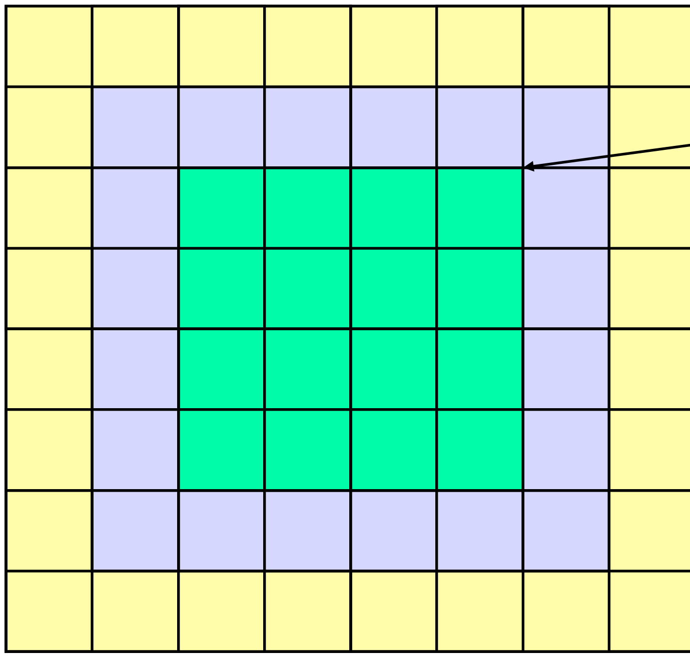
Detailed complexity measures for data movement I: Latency/Bandwidth Model

<u>Moving data between processors by message-passing</u>

- Machine parameters:
  - α latency (message startup time in seconds)
  - β inverse bandwidth (in seconds per word)
  - between nodes of Triton, $\alpha \sim 2.2 \times 10^{-6}$ and $\beta \sim 6.4 \times 10^{-9}$
- Time to send & recv or bcast a message of w words: $\alpha + w*\beta$
- t_{comm} total commmunication time
- t_{comp} total computation time
- Total parallel time: $t_p = t_{comp} + t_{comm}$

Ghost Nodes in Stencil Computations

Comm cost = α * (#messages) + β * (total size of messages)



Green = my interior nodes

Blue = my boundary nodes

Yellow
= neighbors' boundary nodes
= my "ghost nodes"

- Keep a ghost copy of neighbors' boundary nodes
- Communicate every second iteration, not every iteration
- Reduces #messages, not total size of messages
- Costs extra memory and computation
- Can also use more than one layer of ghost nodes

The Landscape of Ax = b Algorithms

	Gaussian elimination	Iterative
Any matrix	**Pivoting LU**	**GMRES, BiCGSTAB, …**
Symmetric positive definite matrix	**Cholesky**	**Conjugate gradient**

More General

More Robust

More Robust **Less Storage**

Conjugate gradient in general

- CG can be used to solve *any* system $Ax = b$, if …

Conjugate gradient in general

- CG can be used to solve *any* system $Ax = b$, if …
- The matrix A is *symmetric* ($a_{ij} = a_{ji}$) …
- … and *positive definite* (all eigenvalues > 0).

Conjugate gradient in general

- CG can be used to solve *any* system $Ax = b$, if …
- The matrix A is *symmetric* ($a_{ij} = a_{ji}$) …
- … and *positive definite* (all eigenvalues > 0).

- Symmetric positive definite matrices occur a lot in scientific computing & data analysis!

Conjugate gradient in general

- CG can be used to solve *any* system $Ax = b$, if …
- The matrix A is *symmetric* ($a_{ij} = a_{ji}$) …
- … and *positive definite* (all eigenvalues > 0).

- Symmetric positive definite matrices occur a lot in scientific computing & data analysis!

- But usually the matrix isn't just a stencil.
- Now we do need to store the matrix A. Where's the data?

Conjugate gradient in general

- CG can be used to solve *any* system $Ax = b$, if …
- The matrix A is *symmetric* ($a_{ij} = a_{ji}$) …
- … and *positive definite* (all eigenvalues > 0).

- Symmetric positive definite matrices occur a lot in scientific computing & data analysis!

- But usually the matrix isn't just a stencil.
- Now we do need to store the matrix A. Where's the data?

- The key is to use graph data structures and algorithms.

Vector and matrix primitives for CG

- DAXPY: $v = \alpha * v + \beta * w$ (vectors v, w; scalars α, β)
  - Broadcast the scalars α and β, then independent * and +
  - comm volume = 2p, span = log n
- DDOT: $\alpha = v^T * w = \sum_j v[j]*w[j]$ (vectors v, w; scalar α)
  - Independent *, then + reduction
  - comm volume = p, span = log n
- Matvec: $v = A*w$ (matrix A, vectors v, w)
  - The hard part
  - But all you need is a subroutine to compute v from w
  - Sometimes you don't need to store A (e.g. temperature problem)
  - Usually you do need to store A, but it's *sparse* ...

Conjugate gradient: Krylov subspaces

- Eigenvalues: $Av = \lambda v \qquad \{ \lambda_1, \lambda_2, \ldots, \lambda_n \}$
- Cayley-Hamilton theorem:

$$(A - \lambda_1 I) \cdot (A - \lambda_2 I) \cdots (A - \lambda_n I) = 0$$

Therefore $\sum_{0 \le i \le n} c_i A^i = 0$ for some c_i

so $A^{-1} = \sum_{1 \le i \le n} (-c_i/c_0) A^{i-1}$

- Krylov subspace:

Therefore if $Ax = b$, then $x = A^{-1} b$ and

$$x \in \text{span}(b, Ab, A^2 b, \ldots, A^{n-1} b) = K_n(A, b)$$

Conjugate gradient: Orthogonal sequences

- Krylov subspace: $K_i(A, b) = \text{span}(b, Ab, A^2b, \ldots, A^{i-1}b)$
- Conjugate gradient algorithm:

  <u>for</u> $i = 1, 2, 3, \ldots$

  find $x_i \in K_i(A, b)$

  such that $r_i = (b - Ax_i) \perp K_i(A, b)$

- Notice $r_i \in K_{i+1}(A, b)$, so $r_i \perp r_j$ for all $j < i$
- Similarly, the "directions" are A-orthogonal:

  $$(x_i - x_{i-1})^T \cdot A \cdot (x_j - x_{j-1}) = 0$$

- The magic: Short recurrences. . .

  A is symmetric => can get next residual and direction from the previous one, without saving them all.

Conjugate gradient: Convergence

- In exact arithmetic, CG converges in n steps
  (completely unrealistic!!)
- Accuracy after k steps of CG is related to:
  - consider polynomials of degree k that are equal to 1 at 0.
  - how small can such a polynomial be at all the eigenvalues of A?
- Thus, eigenvalues close together are good.
- Condition number: $\kappa(A) = ||A||_2 \, ||A^{-1}||_2 = \lambda_{max}(A) / \lambda_{min}(A)$
- Residual is reduced by a constant factor by
  $O(\kappa^{1/2}(A))$ iterations of CG.

Other Krylov subspace methods

- Nonsymmetric linear systems:
  - GMRES:
    for $i = 1, 2, 3, \ldots$
    find $x_i \in K_i(A, b)$ such that $r_i = (Ax_i - b) \perp K_i(A, b)$
    But, no short recurrence => save old vectors => lots more space
    (Usually "restarted" every k iterations to use less space.)
  - BiCGStab, QMR, etc.:
    Two spaces $K_i(A, b)$ and $K_i(A^T, b)$ w/ mutually orthogonal bases
    Short recurrences => O(n) space, but less robust
  - Convergence and preconditioning more delicate than CG
  - Active area of current research
- Eigenvalues: Lanczos (symmetric), Arnoldi (nonsymmetric)

Conjugate gradient iteration

$x_0 = 0, \quad r_0 = b, \quad d_0 = r_0$

for $k = 1, 2, 3, \ldots$

$\alpha_k = (r^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ approx solution

$r_k = r_{k-1} - \alpha_k A d_{k-1}$ residual

$\beta_k = (r^T_k r_k) / (r^T_{k-1} r_{k-1})$ improvement

$d_k = r_k + \beta_k d_{k-1}$ search direction

- One matrix-vector multiplication per iteration
- Two vector dot products per iteration
- Four n-vectors of working storage