CS 5220

Sparse linear algebra

David Bindel

2024-10-31

World of Linear Algebra

- Dense methods (last week)
- Sparse direct methods (Thurs)
- Iterative methods (today and Thurs)

Dense Methods

```
% Dense (LAPACK)
[L,U] = lu(A);
x = U\(L\b);
```

- Direct representation of matrices with simple data structures (no need for indexing data structure)
- Mostly $O(n^3)$ factorizations (on $O(n^2)$ data)
- Good for locality, get to high fraction of peak

Software Strategies: Dense Case

Assuming you want to *use* (vs develop) dense LA code:

- Learn enough to identify right algorithm
  (e.g. is it symmetric? definite? banded? etc)
- Learn high-level organizational ideas
- Make sure you have a good BLAS
- Call LAPACK/ScaLAPACK!
- For n large: wait a while

Sparse Solves?

Questions for you:

- What is a sparse matrix?
- If we only get 3% peak (vs 75%) why consider sparsity?
- What features of sparse matrices might matter for HPC?

Sparse Solves?

Consider $Ax = b$ where A sparse.

- Few nonzeros per row
  - Use a sparse format (e.g. compressed sparse row)
  - Mostly – may have dense rows/columns
  - 10% sparse is maybe best treated as dense!
  - Dense submatrix structure helps performance!
- Representation may also be implicit (just matvec)
  - Includes *data sparse* matrices

Sparse Solves?

One size does not fit all! What if A is

- From a low-order PDE solver?
- From a high-order PDE solver?
    - Spectral methods?
    - Spectral elements?
- From a social network?
- From a Gaussian process?
    - Spatio-temporal stats or ML?

Sparse Direct Methods

```
% Sparse direct (UMFPACK + COLAMD)
[L,U,P,Q] = lu(A);
x = Q*(U\(L\(P*b)));
```

- Direct representation, keep only the nonzeros
- Factorization costs depend on problem structure (1D cheap; 2D reasonable; 3D gets expensive; not easy to give a general rule, and NP hard to order for optimal sparsity)
- Robust, but hard to scale to large 3D problems

Sparse Direct Strategies

Assuming you want to use (vs develop) sparse LA code

- Identify right algorithm (mainly Cholesky vs LU)
- Get a good solver (often from list)
  - You *don't* want to roll your own!
- *Order your unknowns* for sparsity
  - Again, good to use someone else's software!
- For n large, 3D: get lots of memory and wait

Sparse Iterations

```
% Sparse iterative (PCG + incomplete Cholesky)
tol = 1e-6;
maxit = 500;
R = cholinc(A,'0');
x = pcg(A,b,tol,maxit,R',R);
```

- Only *need* $y = Ax$ (maybe $y = A^T x$)
- Produce successively better (?) approximations
- Good convergence depends on *preconditioning*
- Best preconditioners are often hard to parallelize

LA Software: the Wider World

- Dense: LAPACK, ScaLAPACK, PLAPACK, PLASMA, MAGMA
- Sparse direct: UMFPACK, TAUCS, SuperLU, MUMPS, Pardiso, SPOOLES, ...
- Sparse iterative: too many!
  - PETSc (Argonne, object-oriented C)
  - Trilinos (Sandia, C++)

Sparse Iterative Software

Assuming you want to use (vs develop) sparse LA software...

- Identify a good algorithm (GMRES? CG?)
- Pick a good preconditioner
  - Often helps to know the application
  - ... *and* to know how the solvers work!
- Play with parameters, preconditioner variants, etc...
- Swear until you get acceptable convergence?
- Repeat for the next variation on the problem

Stacking Solvers

(Typical) example from a bone modeling package:

- Outer load stepping loop
- Newton method corrector for each load step
- Preconditioned CG for linear system
- Multigrid preconditioner
- Sparse direct solver for coarse-grid solve (UMFPACK)
- LAPACK/BLAS under that

Plan

- Up next: Stationary iterations
- Thurs: Krylov and sparse direct

Reading: Templates book

Warm-Up

Page Rank:

$$\pi^{(k+1)} = (1-\alpha)P\pi^{(k)} + \alpha\mathbf{1}$$

where

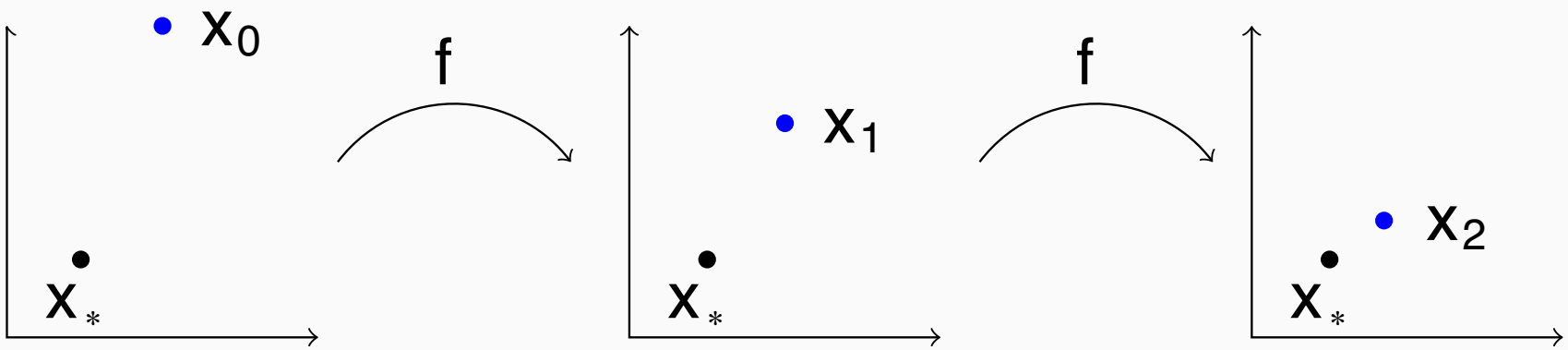
- π_j represents probability of being at node j
- $P_{ij} = 1/d_j$ if there is an edge j to i (0 o.w.)
- α is a damping factor (often around 0.15)

Max difference from limit π^* decreases by $1-\alpha$ per step.

Brain Storm!

- How would you represent P?
- How might you tune *serial* Page Rank?
- How might you *parallelize* Page Rank?
- What barriers would you anticipate for high performance?

Fixed Point Iteration



$$x_{k+1} = f(x_k) \rightarrow x_* = f(x_*)$$

Iterative Idea

- f is a *contraction* if $\|f(x) - f(y)\| < \|x - y\|$, $x \neq y$.
- f has a unique *fixed point* $x_* = f(x_*)$.
- For $x_{k+1} = f(x_k)$, $x_k \to x_*$.
- If $\|f(x) - f(y)\| < \alpha\|x - y\|$, $\alpha < 1$, for all x, y, then

$$\|x_k - x_*\| < \alpha^k \|x - x_*\|$$

- Looks good *if* α not too near 1...

Correction Form

Contraction mapping $f(x) = x + g(x)$ where

- $g(x_*) = 0$
- $\|g(x) - x_*\| \leq \alpha \|x - x_*\|$

Approximate (e.g. with lowered precision):

$$\|\hat{g}(x) - g(x)\| \leq \beta \|x - x_*\|.$$

- Convergence still guaranteed if $\alpha + \beta < 1$!
- Can also analyze absolute errors, etc (another class)

Stationary Iterations

Write $Ax = b$ as $A = M - K$; get fixed point of

$$Mx_{k+1} = Kx_k + b$$

or

$$x_{k+1} = M^{-1}(Kx_k + b) = (M^{-1}K)x_k + M^{-1}b.$$

- Convergence if $\rho(M^{-1}K) < 1$
- Best case for convergence: $M = A$
- Cheapest case: $M = I$

Stationary Iterations

Write $Ax = b$ as $A = M - K$; get fixed point of

$$Mx_{k+1} = Kx_k + b$$

or

$$x_{k+1} = x_k + M^{-1}(b - Ax_k).$$

- Correction form is good for mixed precision!
- Also useful for building to Krylov methods

Performance Model

Exercise: Model time to completion as a function of

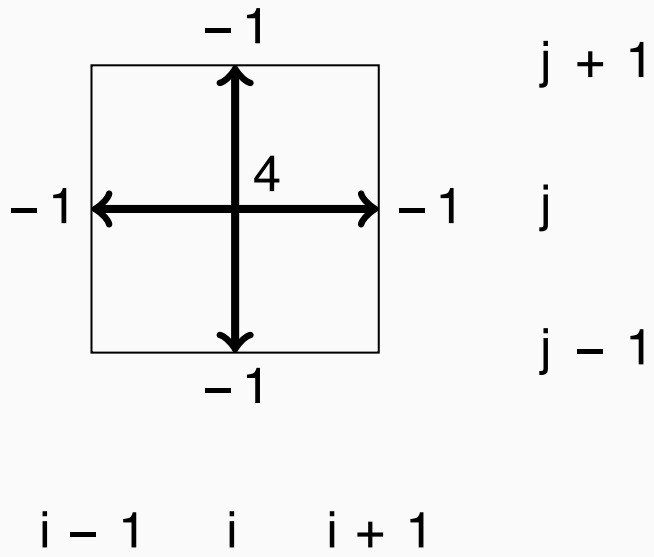
- Setup cost T_{setup}
- Residual cost T_{resid}
- Cost to solve with M, T_{pc}
- Cost to apply update T_{update}
- Initial error norm $\|e\|$
- Contraction rate α
- Desired tolerance τ

Stationary Iterations

- Starting point: choose something between

Jacobi	$M = \mathrm{diag}(A)$
Gauss-Seidel	$M = \mathrm{tril}(A)$

Reminder: Discretized 2D Poisson



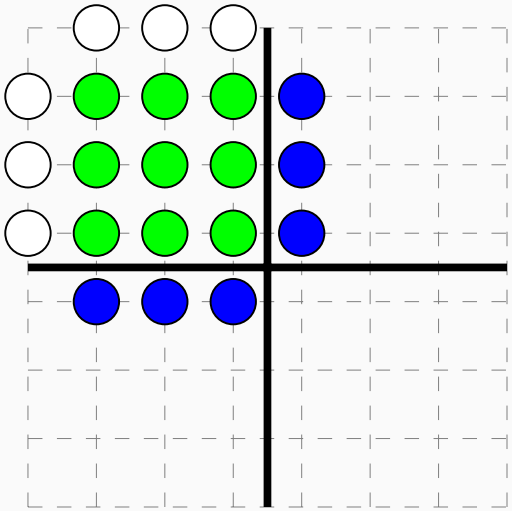
$$(Lu)_{i,j} = h^{-2}\left(4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}\right)$$

Jacobi on 2D Poisson

Assuming homogeneous Dirichlet boundary conditions

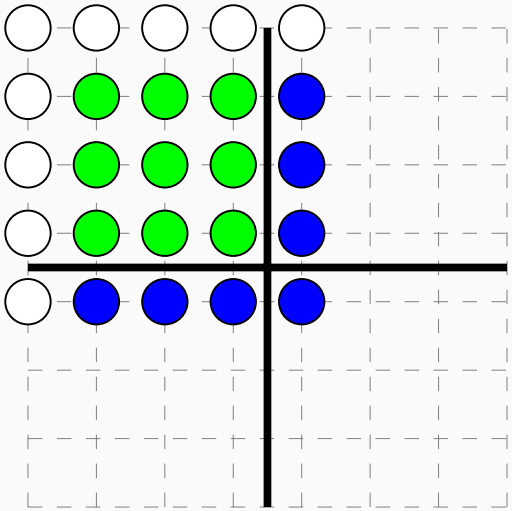
```
for step = 1:nsteps

  for i = 2:n-1
    for j = 2:n-1
      u_next(i,j) = ...
        ( u(i,j+1) + u(i,j-1) + ...
          u(i-1,j) + u(i+1,j) )/4 - ...
        h^2*f(i,j)/4;
    end
  end
  u = u_next;

end
```

Parallel (5 point stencil)



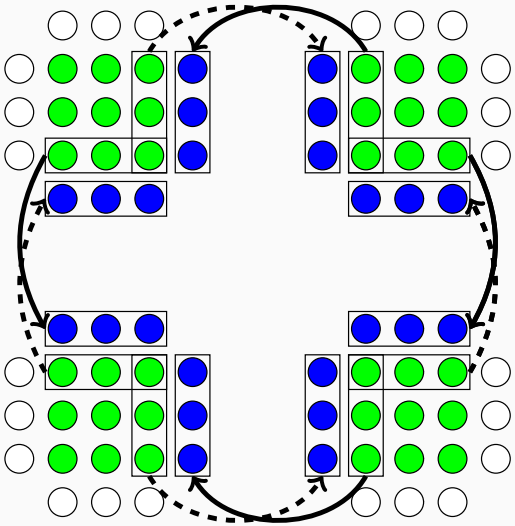
Boundary values:	white
Data on P0:	green
Ghost cell data:	blue

Parallel (9 point stencil)



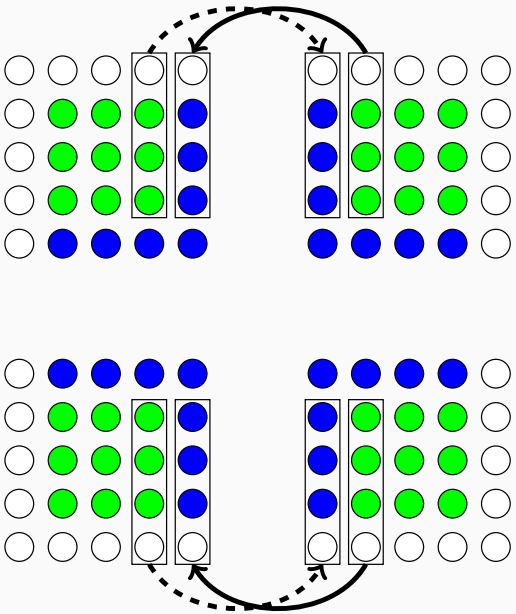
Boundary values:	white
Data on P0:	green
Ghost cell data:	blue

Parallel (5 point stencil)



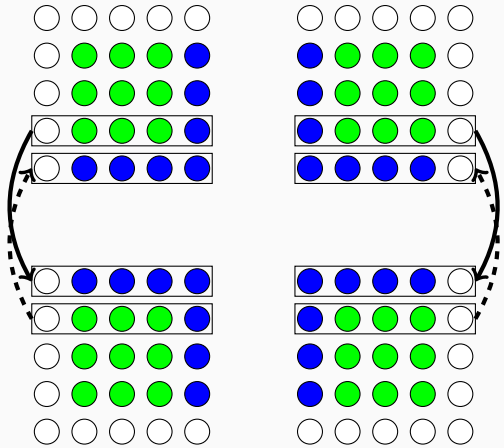
Communicate ghost cells before each step.

Parallel (9 point stencil)



Communicate in two phases (EW, NS) to get corners.

Parallel (9 point stencil)



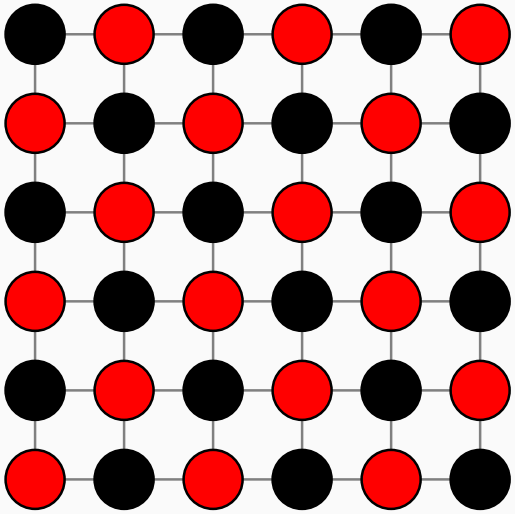
Communicate in two phases (EW, NS) to get corners.

Gauss-Seidel on 2D Poisson

```
for step = 1:nsteps

  for i = 2:n-1
    for j = 2:n-1
      u(i,j) = ...
        ( u(i,j+1) + u(i,j-1) + ...
          u(i-1,j) + u(i+1,j) )/4 - ...
        h^2*f(i,j)/4;
    end
  end

end
```

Bottom values depend on top; how to parallelize?

Red-Black Gauss-Seidel



Red depends only on black, and vice-versa.

Generalization: multi-color orderings

Red-Black Gauss-Seidel Step

```
for i = 2:n-1
  for j = 2:n-1
    if mod(i+j,2) == 0
      u(i,j) = ...
    end
  end
end

for i = 2:n-1
  for j = 2:n-1
    if mod(i+j,2) == 1
      u(i,j) = ...
    end
  end
end
```

Parallel Red-Black Gauss-Seidel

At each step

- Send black ghost cells
- Update red cells
- Send red ghost cells
- Update black ghost cells

More Sophistication

- Successive over-relaxation (SOR): extrapolate G-S
- Block Jacobi: M a block diagonal matrix from A
    - Other block variants similar
- Generalize to overlapping (Schwarz) methods
- Alternating Direction Implicit (ADI): alternately solve on vertical lines and horizontal lines
- Multigrid

Mostly the opening act for *Krylov methods*.