

More Fun With Sparse Matrices

David Bindel

05 Nov 2015

Logistics and life lessons?

- ▶ Some progress on ampxe + MPI – watch Piazza
- ▶ Should be working on shortest paths now
- ▶ Final projects: be careful with scope!
 - ▶ A good small kernel trumps vast ambition without execution
 - ▶ Check against a slow, naive, obvious calculation.

Reminder: Conjugate Gradients

What if we only know how to multiply by A ?
About all you can do is keep multiplying!

$$\mathcal{K}_k(A, b) = \text{span} \{ b, Ab, A^2b, \dots, A^{k-1}b \}.$$

Gives surprisingly useful information!

If A is symmetric and positive definite, $x = A^{-1}b$ minimizes

$$\begin{aligned}\phi(x) &= \frac{1}{2}x^T Ax - x^T b \\ \nabla \phi(x) &= Ax - b.\end{aligned}$$

Idea: Minimize $\phi(x)$ over $\mathcal{K}_k(A, b)$.

Basis for the *method of conjugate gradients*

Convergence of CG

- ▶ KSPs are *not* stationary (no constant fixed-point iteration)
- ▶ Convergence is surprisingly subtle!
- ▶ CG convergence upper bound via *condition number*
 - ▶ Large condition number iff form $\phi(x)$ has long narrow bowl
 - ▶ Usually happens for Poisson and related problems
- ▶ *Preconditioned* problem $M^{-1}Ax = M^{-1}b$ converges faster?
- ▶ Whence M ?
 - ▶ From a stationary method?
 - ▶ From a simpler/coarser discretization?
 - ▶ From approximate factorization?

PCG

```
Compute  $r^{(0)} = b - Ax$   
for  $i = 1, 2, \dots$   
  solve  $Mz^{(i-1)} = r^{(i-1)}$   
   $\rho_{i-1} = (r^{(i-1)})^T z^{(i-1)}$   
  if  $i == 1$   
     $p^{(1)} = z^{(0)}$   
  else  
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$   
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$   
  endif  
   $q^{(i)} = Ap^{(i)}$   
   $\alpha_i = \rho_{i-1} / (p^{(i)})^T q^{(i)}$   
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$   
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$   
end
```

Parallel work:

- ▶ Solve with M
- ▶ Product with A
- ▶ Dot products
- ▶ Axpys

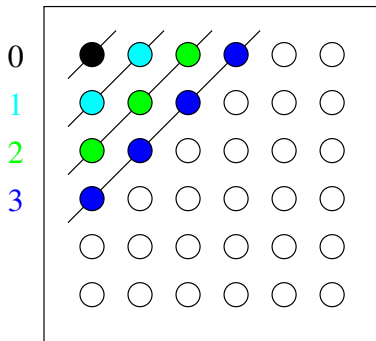
Overlap comm/comp.

PCG bottlenecks

Key: fast solve with M , product with A

- ▶ Some preconditioners parallelize better!
(Jacobi vs Gauss-Seidel)
- ▶ Balance speed with performance.
 - ▶ Speed for set up of M ?
 - ▶ Speed to apply M after setup?
- ▶ Cheaper to do two multiplies/solves at once...
 - ▶ Can't exploit in obvious way — lose stability
 - ▶ Variants allow multiple products — Hoemmen's thesis
- ▶ Lots of fiddling possible with M ; what about matvec with A ?

Thinking on (basic) CG convergence



Consider 2D Poisson with 5-point stencil on an $n \times n$ mesh.

- ▶ Information moves one grid cell per matvec.
- ▶ Cost per matvec is $O(n^2)$.
- ▶ At least $O(n^3)$ work to get information across mesh!

CG convergence: a counting approach

- ▶ Time to converge \geq time to propagate info across mesh
- ▶ For a 2D mesh: $O(n)$ matvecs, $O(n^3) = O(N^{3/2})$ cost
- ▶ For a 3D mesh: $O(n)$ matvecs, $O(n^4) = O(N^{4/3})$ cost
- ▶ “Long” meshes yield slow convergence
- ▶ 3D beats 2D because everything is closer!
 - ▶ Advice: sparse direct for 2D, CG for 3D.
 - ▶ Better advice: use a preconditioner!

CG convergence: an eigenvalue approach

Define the *condition number* for $\kappa(L)$ s.p.d:

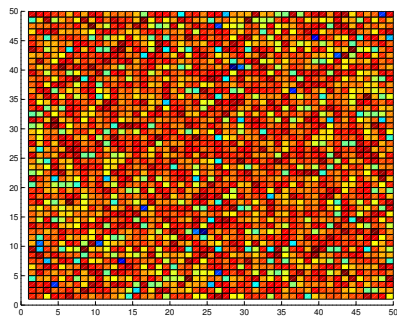
$$\kappa(L) = \frac{\lambda_{\max}(L)}{\lambda_{\min}(L)}$$

Describes how elongated the level surfaces of ϕ are.

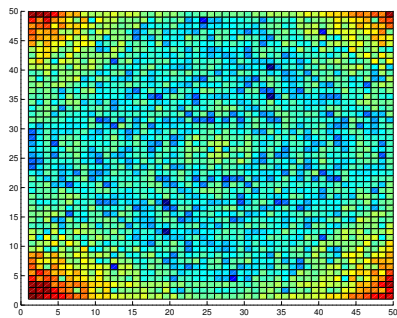
- ▶ For Poisson, $\kappa(L) = O(h^{-2})$
- ▶ CG steps to reduce error by $1/2 = O(\sqrt{\kappa}) = O(h^{-1})$.

Similar back-of-the-envelope estimates for some other PDEs. But these are not always that useful... can be pessimistic if there are only a few extreme eigenvalues.

CG convergence: a frequency-domain approach



FFT of e_0



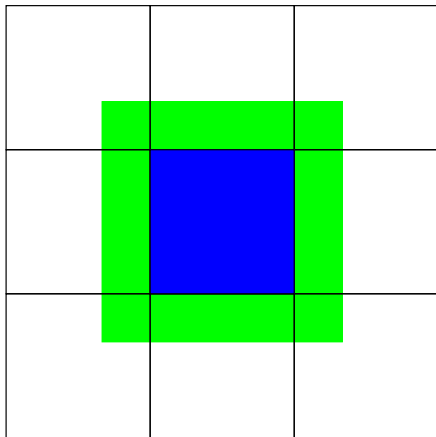
FFT of e_{10}

Error e_k after k steps of CG gets smoother!

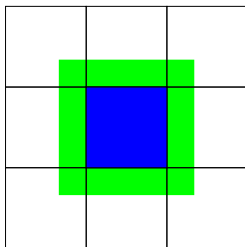
Choosing preconditioners for 2D Poisson

- ▶ CG already handles high-frequency error
- ▶ Want something to deal with lower frequency!
- ▶ Jacobi useless
 - ▶ Doesn't even change Krylov subspace!
- ▶ Better idea: block Jacobi?
 - ▶ Q: How should things split up?
 - ▶ A: Minimize blocks across domain.
 - ▶ Compatible with minimizing communication!

Restrictive Additive Schwartz (RAS)



Restrictive Additive Schwartz (RAS)



- ▶ Get **ghost cell data**
- ▶ Solve *everything* local (including neighbor data)
- ▶ Update **local values** for next step
- ▶ Default strategy in PETSc

Multilevel Ideas

- ▶ RAS propagates information by one processor per step
- ▶ For scalability, still need to get around this!
- ▶ Basic idea: use multiple grids
 - ▶ Fine grid gives lots of work, kills high-freq error
 - ▶ Coarse grid cheaply gets info across mesh, kills low freq

More on this another time.

CG performance

Two ways to get better performance from CG:

1. Better preconditioner
 - ▶ Improves asymptotic complexity?
 - ▶ ... but application dependent
2. Tuned implementation
 - ▶ Improves constant in big-O
 - ▶ ... but application independent?

Benchmark idea (?): no preconditioner, just tune.

Tuning PCG

Compute $r^{(0)} = b - Ax$

for $i = 1, 2, \dots$

 solve $Mz^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = (r^{(i-1)})^T z^{(i-1)}$

 if $i == 1$

$p^{(1)} = z^{(0)}$

 else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

 endif

$q^{(i)} = Ap^{(i)}$

$\alpha_i = \rho_{i-1} / (p^{(i)})^T q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

end

- ▶ Most work in A, M
- ▶ Vector ops synchronize
- ▶ Overlap comm, comp?

Tuning PCG

Compute $r^{(0)} = b - Ax$

$p_{-1} = 0; \beta_{-1} = 0; \alpha_{-1} = 0$

$s = L^{-1}r^{(0)}$

$\rho_0 = s^T s$

for $i = 0, 1, 2, \dots$

$w_i = L^{-T} s$

$p_i = w_i + \beta_{i-1} p_{i-1}$

$q_i = A p_i$

$\gamma = p_i^T q_i$

$x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$

$\alpha_i = \rho_i / \gamma_i$

$r_{i+1} = r_i - \alpha q_i$

$s = L^{-1} r_{i+1}$

$\rho_{i+1} = s^T s$

Check convergence ($\|r_{i+1}\|$)

$\beta_i = \rho_{i+1} / \rho_i$

end

Split $z = M^{-1}r$ into s, w_i

Overlap

- ▶ $p_i^T q_i$ with x update
- ▶ $s^T s$ with w_i eval
- ▶ Computing p_i, q_i, γ
- ▶ Pipeline $r_{i+1}, s?$
- ▶ Pipeline $p_i, w_i?$

Parallel Numerical LA,
Demmel, Heath, van der Vorst

Tuning PCG

Can also tune

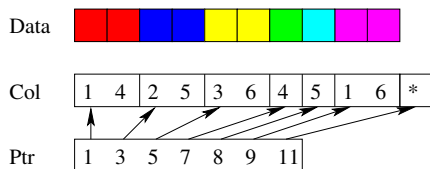
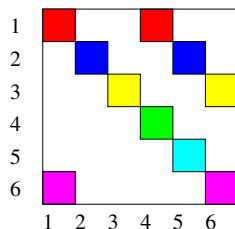
- ▶ Preconditioner solve (hooray!)
- ▶ Matrix multiply
 - ▶ Represented implicitly (regular grids)
 - ▶ Or explicitly (e.g. compressed sparse column)

Or further rearrange algorithm (Hoemmen, Demmel).

Tuning sparse matvec

- ▶ Sparse matrix blocking and reordering (Im, Vuduc, Yelick)
 - ▶ Packages: Sparsity (Im), OSKI (Vuduc)
 - ▶ Available as PETSc extension
- ▶ Optimizing stencil operations (Datta)

Reminder: Compressed sparse row storage



```
for i = 1:n
    y[i] = 0;
    for jj = ptr[i] to ptr[i+1]-1
        y[i] += A[jj]*x[col[jj]];
    end
end
```

Problem: $y[i] += A[jj]*x[\text{col}[j]];$

Memory traffic in CSR multiply

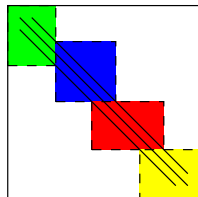
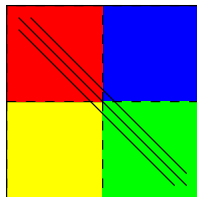
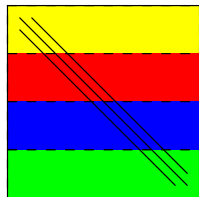
Memory access patterns:

- ▶ Elements of y accessed sequentially
- ▶ Elements of A accessed sequentially
- ▶ Access to x are all over!

Can help by switching to block CSR.

Switching to single precision, short indices can help memory traffic, too!

Parallelizing matvec



- ▶ Each processor gets a piece
- ▶ Many partitioning strategies
- ▶ Idea: re-order so one of these strategies is “good”

Reordering for matvec

SpMV performance goals:

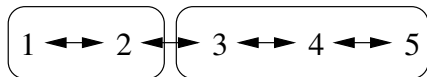
- ▶ Balance load?
- ▶ Balance storage?
- ▶ Minimize communication?
- ▶ Good cache re-use?

Also reorder for

- ▶ Stability of Gauss elimination,
- ▶ Fill reduction in Gaussian elimination,
- ▶ Improved performance of preconditioners...

Reminder: Sparsity and partitioning

$$A = \begin{bmatrix} * & * & & & & & \\ * & * & * & & & & \\ & & & & & & \\ & * & * & * & & & \\ & & * & * & * & & \\ & & & * & * & & \\ & & & & * & * & \end{bmatrix}$$

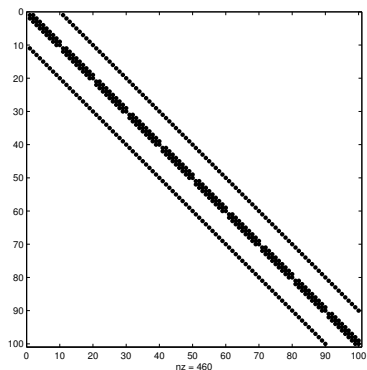


Want to partition sparse graphs so that

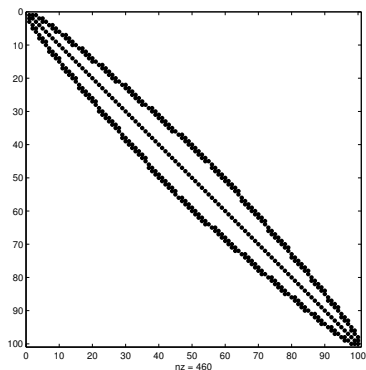
- ▶ Subgraphs are same size (load balance)
- ▶ Cut size is minimal (minimize communication)

Matrices that are “almost” diagonal are good?

Reordering for bandedness



Natural order



RCM reordering

Reverse Cuthill-McKee

- ▶ Select “peripheral” vertex v
- ▶ Order according to breadth first search from v
- ▶ Reverse ordering

From iterative to direct

- ▶ RCM ordering is great for SpMV
- ▶ But isn't narrow banding good for solvers, too?
 - ▶ LU takes $O(nb^2)$ where b is bandwidth.
 - ▶ Great if there's an ordering where b is small!

Skylines and profiles

- ▶ *Profile* solvers generalize band solvers
- ▶ Use skyline storage; if storing lower triangle, for each row i :
 - ▶ Start and end of storage for nonzeros in row.
 - ▶ *Contiguous* nonzero list up to main diagonal.
- ▶ In each column, first nonzero defines a profile.
- ▶ All fill-in confined to profile.
- ▶ RCM is again a good ordering.

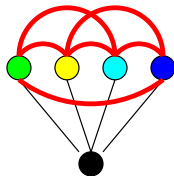
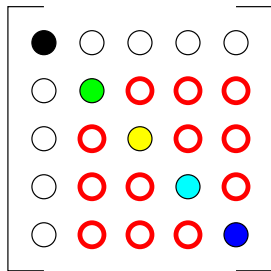
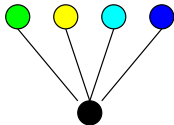
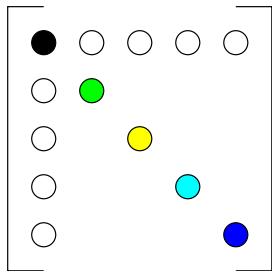
Beyond bandedness

- ▶ Bandedness only takes us so far
 - ▶ Minimum bandwidth for 2D model problem? 3D?
 - ▶ Skyline only gets us so much farther
- ▶ But more general solvers have similar structure
 - ▶ Ordering (minimize fill)
 - ▶ Symbolic factorization (where will fill be?)
 - ▶ Numerical factorization (pivoting?)
 - ▶ ... and triangular solves

Reminder: Matrices to graphs

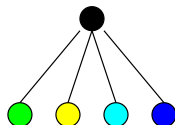
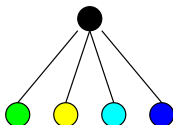
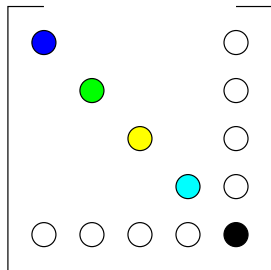
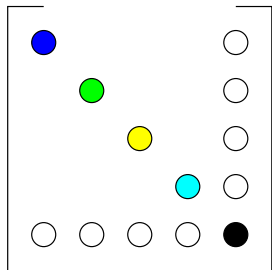
- ▶ $A_{ij} \neq 0$ means there is an edge between i and j
- ▶ Ignore self-loops and weights for the moment
- ▶ Symmetric matrices correspond to undirected graphs

Troublesome Trees



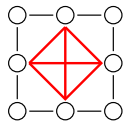
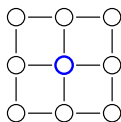
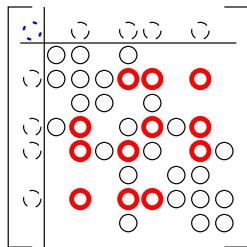
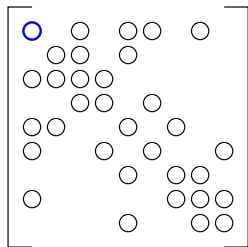
One step of Gaussian elimination *completely* fills this matrix!

Terrific Trees



Full Gaussian elimination generates *no* fill in this matrix!

Graphic Elimination



Eliminate a variable, connect all neighbors.

Graphic Elimination

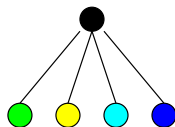
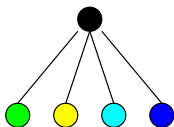
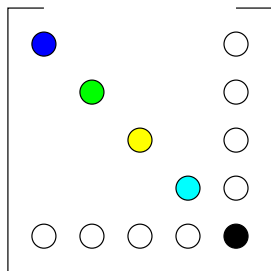
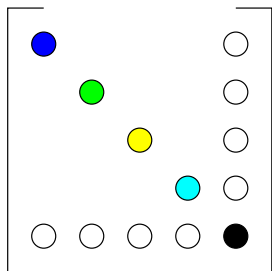
Consider first steps of GE

$$\begin{aligned}A(2:\text{end}, 1) &= A(2:\text{end}, 1) / A(1, 1); \\A(2:\text{end}, 2:\text{end}) &= A(2:\text{end}, 2:\text{end}) - \dots \\ &\quad A(2:\text{end}, 1) * A(1, 2:\text{end});\end{aligned}$$

Nonzero in the outer product at (i, j) if $A(i, 1)$ and $A(j, 1)$ both nonzero — that is, if i and j are both connected to 1.

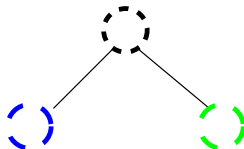
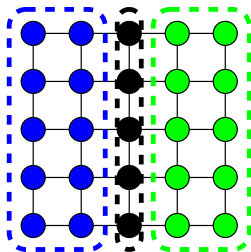
General: Eliminate variable, connect remaining neighbors.

Terrific Trees Redux



Order leaves to root \implies
on eliminating i , parent of i is only remaining neighbor.

Nested Dissection



- ▶ Idea: Think of *block* tree structures.
- ▶ Eliminate block trees from bottom up.
- ▶ Can recursively partition at leaves.
- ▶ Rough cost estimate: how much just to factor dense Schur complements associated with separators?
- ▶ Notice graph partitioning appears again!
 - ▶ And again we want small separators!

Nested Dissection

Model problem: Laplacian with 5 point stencil (for 2D)

- ▶ ND gives optimal complexity in exact arithmetic (George 73, Hoffman/Martin/Rose)
- ▶ 2D: $O(N \log N)$ memory, $O(N^{3/2})$ flops
- ▶ 3D: $O(N^{4/3})$ memory, $O(N^2)$ flops

Minimum Degree

- ▶ Locally greedy strategy
 - ▶ Want to minimize upper bound on fill-in
 - ▶ Fill $\leq (\text{degree in remaining graph})^2$
- ▶ At each step
 - ▶ Eliminate vertex with smallest degree
 - ▶ Update degrees of neighbors
- ▶ Problem: Expensive to implement!
 - ▶ But better variants via *quotient graphs*
 - ▶ Variants often used in practice

Elimination Tree

- ▶ Variables (columns) are nodes in trees
- ▶ j a descendant of k if eliminating j updates k
- ▶ Can eliminate disjoint subtrees in parallel!

Cache locality

Basic idea: exploit “supernodal” (dense) structures in factor

- ▶ e.g. arising from elimination of separator Schur complements in ND
- ▶ Other alternatives exist (multifrontal solvers)

Pivoting

Pivoting is a tremendous pain, particularly in distributed memory!

- ▶ Cholesky — no need to pivot!
- ▶ Threshold pivoting — pivot when things look dangerous
- ▶ Static pivoting — try to decide up front

What if things go wrong with threshold/static pivoting?

Common theme: Clean up sloppy solves with good residuals

Direct to iterative

Can improve solution by *iterative refinement*:

$$PAQ \approx LU$$

$$x_0 \approx QU^{-1}L^{-1}Pb$$

$$r_0 = b - Ax_0$$

$$x_1 \approx x_0 + QU^{-1}L^{-1}Pr_0$$

Looks like approximate Newton on $F(x) = Ax - b = 0$.

This is just a stationary iterative method!

Nonstationary methods work, too.

Variations on a theme

If we're willing to sacrifice some on factorization,

- ▶ Single precision + refinement on double precision residual?
- ▶ Sloppy factorizations (marginal stability) + refinement?
- ▶ Modify m small pivots as they're encountered (low rank updates), fix with m steps of a Krylov solver?