Lecture 15: More Iterative Ideas

David Bindel

15 Mar 2010
Logistics

- HW 2 due!
- Some notes on HW 2.
- Where we are / where we’re going
- More iterative ideas.
- Intro to HW 3.
More HW 2 notes

See solution code!
Life lessons from HW 2?

- Where an error occurs may not be where you observe it!
- Check against a slow, naive, obvious calculation.
- `assert` is your friend.
- Use version control (`git`, `cvs`, `svn`, ...).
Where we’ve been

So far: some basic technology and algorithmic ideas

- Architectural ideas (serial and parallel)
- Parallel programming models
- OpenMP and MPI programming
- Overview of parallel simulation ideas
- Overview of dense and sparse linear algebra
Where we’re going

Still up (technology side):
  ► UPC programming
  ► CUDA programming
  ► Performance analysis tools
  ► Scripting, code generation, mixed-language coding
  ► Library use and PETSc/Trilinos
Where we’re going

Still up (algorithmic side):

- Sparse direct methods
- FFT and spectral methods
- Multigrid and domain decomposition
- Hierarchical methods for $N$-body
- Graph partitioning
- Parallel sort

... and some applications as time permits.
Reminder: Conjugate Gradients

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

$$K_k(A, b) = \text{span} \left\{ b, Ab, A^2b, \ldots, A^{k-1}b \right\}.$$ 

Gives surprisingly useful information!

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

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b$$

$$\nabla \phi(x) = Ax - b.$$ 

Idea: Minimize $\phi(x)$ over $K_k(A, b)$. 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?
Compute $r^{(0)} = b - Ax$
for $i = 1, 2, \ldots$
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

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

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

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

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

dend

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:

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

Describes how elongated the level surfaces of $\phi$ 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 \textit{ghost cell data}
- Solve \textit{everything} local (including neighbor data)
- Update \textit{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.
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 \( i = 1, 2, \ldots \)

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 \( i = 0, 1, 2, \ldots \)
\( w_i = L^{-T} s \)
\( p_i = w_i + \beta_{i-1} p_{i-1} \)
\( q_i = A p_i \)
\( \gamma = p_i^T q_i \)
\( x_i = x_{i-1} + \alpha_{i-1} p_{i-1} \)
\( \alpha_i = \rho_i / \gamma_i \)
\( r_{i+1} = r_i - \alpha q_i \)
\( s = L^{-1} r_{i+1} \)
\( \rho_{i+1} = s^T s \)
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 \( p_i, q_i, \gamma \)
- 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)
for $i = 1:n$
  $y[i] = 0$;
  for $jj = ptr[i]$ to $ptr[i+1]-1$
    $y[i] += A[jj] \times x[col[j]]$;
  end
end

Problem: $y[i] += A[jj] \times x[col[j]]$;
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

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
HW 3 preview

Given serial implementation of:

- 3D elastic finite element code
- Regular mesh, variable material properties
- PCG solver with an RAS preconditioner

Wanted:

- Parallelized CG solver (MPI or OpenMP)
- Study of scaling with $n, p$
- Some fun tuning (e.g. matrix layout)