Lecture 16:
Iterative Methods and Sparse Linear Algebra

David Bindel

25 Oct 2011
Logistics

- Send me a project title and group (today, please!)
- Project 2 due next Monday, Oct 31
<aside topic="proj2">
// x bin and interaction range (y similar)
int ix = (int) (x / (2*h));
int ixlo = (int) ((x-h)/(2*h));
int ixhi = (int) ((x+h)/(2*h));
Spatial binning and hashing

- Simplest version
  - One linked list per bin
  - Can include the link in a particle \texttt{struct}
  - Fine for this project!

- More sophisticated version
  - Hash table keyed by bin index
  - Scales even if occupied volume $\ll$ computational domain
Partitioning strategies

Can make each processor responsible for
  ▶ A region of space
  ▶ A set of particles
  ▶ A set of interactions

Different tradeoffs between load balance and communication.
To use symmetry, or not to use symmetry?

- Simplest version is prone to race conditions!
- Can not use symmetry (and do twice the work)
- Or update bins in two groups (even/odd columns?)
- Or save contributions separately, sum later
Logistical odds and ends

- Parallel performance starts with serial performance
  - Use flags — let the compiler help you!
  - Can refactor memory layouts for better locality
- You will need more particles to see good speedups
  - Overheads: open/close parallel sections, barriers.
  - Try -s 1e-2 (or maybe even smaller)
- Careful notes and a version control system *really* help
  - I like Git’s lightweight branches here!
Reminder: World of Linear Algebra

- **Dense methods**
  - Direct representation of matrices with simple data structures (no need for indexing data structure)
  - Mostly $O(n^3)$ factorization algorithms

- **Sparse direct methods**
  - 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

- **Iterative methods**
  - 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
% Dense (LAPACK)
[L,U] = lu(A);
x = U\(L\b);

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

% Sparse iterative (PCG + incomplete Cholesky)
tol = 1e-6;
maxit = 500;
R = cholinc(A,'0');
x = pcg(A,b,tol,maxit,R',R);
Linear Algebra Software: the Wider World

- Dense: LAPACK, ScaLAPACK, PLAPACK
- Sparse direct: UMFPACK, TAUCS, SuperLU, MUMPS, Pardiso, SPOOLES, ...
- Sparse iterative: too many!
- Sparse mega-libraries
  - PETSc (Argonne, object-oriented C)
  - Trilinos (Sandia, C++)
- Good references:
  - *Templates for the Solution of Linear Systems* (on Netlib)
  - Survey on “Parallel Linear Algebra Software” (Eijkhout, Langou, Dongarra – look on Netlib)
  - ACTS collection at NERSC
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
Software Strategies: Sparse Direct Case

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
Software Strategies: Sparse Iterative Case

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

Frameworks (e.g. PETSc or Trilinos) speed experimentation.
(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

First three are high level — I used a scripting language (Lua).
Iterative Idea

- $f$ is a contraction if $\|f(x) - f(y)\| < \|x - y\|$.  
- $f$ 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 $\alpha$ not too near 1...
Stationary Iterations

Write \( Ax = b \) as \( A = M - K \); get fixed point of

\[
Mx_{k+1} = Kx_k + b
\]

or

\[
x_{k+1} = (M^{-1}K)x_k + M^{-1}b.
\]

- Convergence if \( \rho(M^{-1}K) < 1 \)
- Best case for convergence: \( M = A \)
- Cheapest case: \( M = I \)
- Realistic: choose something between
  - Jacobi \( M = \text{diag}(A) \)
  - Gauss-Seidel \( M = \text{tril}(A) \)
Reminder: Discretized 2D Poisson Problem

\[(Lu)_{i,j} = h^{-2} (4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1})\]
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
    end

    u = u_next;
end

Basically do some averaging at each step.
Parallel version (5 point stencil)

Boundary values: white
Data on P0: green
Ghost cell data: blue
Parallel version (9 point stencil)

Boundary values: white
Data on P0: green
Ghost cell data: blue
Parallel version (5 point stencil)

Communicate ghost cells before each step.
Parallel version (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 sketch

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 Gauss-Seidel direction
- Block Jacobi: let $M$ be a block diagonal matrix from $A$
  - Other block variants similar
- Alternating Direction Implicit (ADI): alternately solve on vertical lines and horizontal lines
- Multigrid

These are mostly just the opening act for...
Krylov Subspace Methods

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

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

Gives surprisingly useful information!
Example: Conjugate Gradients

If $A$ is symmetric and positive definite, $Ax = b$ solves a minimization:

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

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

Idea: Minimize $\phi(x)$ over $\mathcal{K}_k(A, b)$. Basis for the *method of conjugate gradients*
Example: GMRES

Idea: Minimize $\|Ax - b\|^2$ over $\mathcal{K}_k(A, b)$.
Yields \textit{Generalized Minimum RESidual} (GMRES) method.
Convergence of Krylov Subspace Methods

- 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 $i = 1, 2, \ldots$
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

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 $A$?