Sparse Direct Methods on High Performance Computers

X. Sherry Li
xsli@lbl.gov
http://crd.lbl.gov/~xiaoye

CS267/E233: Applications of Parallel Computing
March 14, 2007
Review of Gaussian Elimination (GE)

- Solving a system of linear equations $Ax = b$

- First step of GE:

$$A = \begin{bmatrix} \alpha & w^T \\ v & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/\alpha & I \end{bmatrix} \cdot \begin{bmatrix} \alpha & w^T \\ 0 & C \end{bmatrix}$$

$$C = B - \frac{v \cdot w^T}{\alpha}$$

- Repeats GE on $C$

- Results in LU factorization ($A = LU$)
  - $L$ lower triangular with unit diagonal, $U$ upper triangular

- Then, $x$ is obtained by solving two triangular systems with $L$ and $U$
Sparse matrices are ubiquitous
- Example: A of dimension $10^5$, only 10~100 nonzeros per row
- Goal: Store only nonzeros and perform operations only on nonzeros
- Fill-in: original zero entry $a_{ij}$ becomes nonzero in L and U
  
  Natural order: nonzeros = 233  
  Min. Degree order: nonzeros = 207
Compressed Column Storage (CCS)

- Also known as Harwell-Boeing format
- Store nonzeros columnwise contiguously
- 3 arrays:
  - Storage: NNZ reals, NNZ+N+1 integers
- Efficient for columnwise algorithms

\[
\begin{bmatrix}
1 & a \\
2 & b \\
c & d & 3 \\
e & 4 & f \\
5 & g \\
h & i & 6 & j \\
k & l & 7
\end{bmatrix}
\]

nzval
\[
\begin{array}{cccccccc}
1 & c & 2 & d & e & 3 & k & a & 4 & h & b & f & 5 & i & l & 6 & g & j & 7
\end{array}
\]

rowind
\[
\begin{array}{cccccccc}
1 & 3 & 2 & 3 & 4 & 3 & 7 & 1 & 4 & 6 & 2 & 4 & 5 & 6 & 7 & 6 & 5 & 6 & 7
\end{array}
\]

colptr
\[
\begin{array}{cccccccc}
1 & 3 & 6 & 8 & 11 & 16 & 17 & 20
\end{array}
\]

- Ref: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, R. Barrett et al.
Numerical Stability: Need for Pivoting

- One step of GE:

\[
A = \begin{bmatrix} \alpha & w^T \\ v & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/\alpha & I \end{bmatrix} \cdot \begin{bmatrix} \alpha & w^T \\ 0 & C \end{bmatrix}
\]

- \( C = B - \frac{v \cdot w^T}{\alpha} \)

  - If \( \alpha \) is small, some entries in \( B \) may be lost from addition

- Pivoting: swap the current diagonal entry with a larger entry from the other part of the matrix

- Goal: prevent \( C \) from getting too large
Dense versus Sparse GE

- **Dense GE:** $P_r A P_c = LU$
  - $P_r$ and $P_c$ are permutations chosen to maintain stability
  - Partial pivoting suffices in most cases: $P_r A = LU$
- **Sparse GE:** $P_r A P_c = LU$
  - $P_r$ and $P_c$ are chosen to maintain stability and preserve sparsity
  - Dynamic pivoting causes dynamic structural change
    - Alternatives: threshold pivoting, static pivoting, . . .
Algorithmic Issues in Sparse GE

- Minimize number of fill-ins, maximize parallelism
  - Sparsity structure of L & U depends on that of A, which can be changed by row/column permutations (vertex re-labeling of the underlying graph)
  - Ordering (combinatorial algorithms; NP-complete to find optimum [Yannakis '83]; use heuristics)

- Predict the fill-in positions in L & U
  - Symbolic factorization (combinatorial algorithms)

- Perform factorization and triangular solutions
  - Numerical algorithms (F.P. operations only on nonzeros)
    - How and when to pivot?
  - Usually dominate the total runtime
Local greedy: minimize upper bound on fill-in

\[
\begin{bmatrix}
1 & i & j & k & l \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
i & x & x & x & x & x \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
i & x \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]
Minimum Degree Ordering (2/3)

- **Greedy approach:** do the best locally
  - Best for modest size problems
  - Hard to parallelize

- **At each step**
  - Eliminate the vertex with the smallest degree
  - Update degrees of the neighbors

- **Straightforward implementation** is slow and requires too much memory
  - Newly added edges are more than eliminated vertices
Minimum Degree Ordering (3/3)

- Use **quotient graph** as a compact representation [George/Liu '78]

- Collection of cliques resulting from the eliminated vertices affects the degree of an uneliminated vertex

- Represent each connected component in the eliminated subgraph by a single “supervertex”

- Storage required to implement QG model is bounded by size of $A$

- Large body of literature on implementation variants
  - Tinney/Walker `67, George/Liu `79, Liu `85,
    Amestoy/Davis/Duff `94, Ashcraft `95, Duff/Reid `95, et al., . .
Model problem: discretized system $Ax = b$ from certain PDEs, e.g., 5-point stencil on $n \times n$ grid, $N = n^2$

- Factorization flops: $O(n^3) = O(N^{3/2})$

Theorem: ND ordering gave optimal complexity in exact arithmetic \cite{George '73, Hoffman/Martin/Ross}
**ND Ordering (2/3)**

- **Generalized nested dissection** [Lipton/Rose/Tarjan '79]
  - Global graph partitioning: top-down, divide-and-conquer
  - Best for largest problems
  - Parallel code available: e.g., ParMETIS
  - First level

  \[
  \begin{bmatrix}
  A & 0 & x \\
  0 & B & x \\
  x & x & S
  \end{bmatrix}
  \]

  - Recurse on A and B

- **Goal:** find the smallest possible separator S at each level
  - Multilevel schemes:
    - Chaco [Hendrickson/Leland `94], Metis [Karypis/Kumar `95]
    - Spectral bisection [Simon et al. `90-`95]
    - Geometric and spectral bisection [Chan/Gilbert/Teng `94]
ND Ordering (3/3)


**Ordering for LU (unsymmetric)**

- Can use a symmetric ordering on a symmetrized matrix.
- **Case of partial pivoting (sequential SuperLU):**
  Use ordering based on $A^TA$
  - If $RTR = A^TA$ and $PA = LU$, then for any row permutation $P$, $\text{struct}(L+U) \subseteq \text{struct}(R^TR)$ [George/Ng `87]
  - Making $R$ sparse tends to make $L$ & $U$ sparse . . .

- **Case of static pivoting (SuperLU_DIST):**
  Use ordering based on $A^{T+A}$
  - If $RTR = A^{T+A}$ and $A = LU$, then $\text{struct}(L+U) \subseteq \text{struct}(R^TR)$
  - Making $R$ sparse tends to make $L$ & $U$ sparse . . .
  
  - Can find better ordering based solely on $A$, without symmetrization [Amestoy/Li/Ng `03]
Ordering for LU

- Still wide open . . .

- Simple extension: symmetric ordering using $A'+A$
  - Greedy algorithms, graph partitioning, or hybrid
- Problem: unsymmetric structure is not respected!

- We developed an unsymmetric variant of “Min Degree” algorithm based solely on $A$ [Amestoy/Li/Ng ’02]
  (a.k.a. Markowitz scheme)
**Structural Gaussian Elimination - Unsymmetric Case**

- Bipartite graph
- After a vertex is eliminated, all the row & column vertices adjacent to it become fully connected - “bi-clique” (assuming diagonal pivot)
- The edges of the bi-clique are the potential fills (upper bound !)
Results of Markowitz Ordering

- Extend the QG model to bipartite quotient graph
- Same asymptotic complexity as symmetric MD
  - Space is bounded by $2*(m + n)$
  - Time is bounded by $O(n \times m)$

- For 50+ unsym. matrices, compared with MD on $A'+A$:
  - Reduction in fill: average 0.88, best 0.38
  - Reduction in f.p. operations: average 0.77, best 0.01

- How about graph partitioning? (open problem)
  - Use directed graph
High Performance Issues: Reduce Cost of Memory Access & Communication

- Blocking to increase number of floating-point operations performed for each memory access

- Aggregate small messages into one larger message
  - Reduce cost due to latency

- Well done in LAPACK, ScaLAPACK
  - Dense and banded matrices

- Adopted in the new generation sparse software
  - Performance much more sensitive to latency in sparse case
General Sparse Solver

- Use (blocked) CRS or CCS, and any ordering method
  - Leave room for fill-ins! (symbolic factorization)
- Exploit “supernodal” (dense) structures in the factors
  - Can use Level 3 BLAS
  - Reduce inefficient indirect addressing (scatter/gather)
  - Reduce graph traversal time using a coarser graph
CS267

Speedup Over Un-blocked Code

- Sorted in increasing “reuse ratio” = \#Flops/nonzeros
- Up to 40% of machine peak on large sparse matrices on IBM RS6000/590, MIPS R8000, 25% on Alpha 21164
Parallel Task Scheduling for SMPs (in SuperLU_MT)

- **Elimination tree** exhibits parallelism and dependencies

```
Shared task queue initialized by leaves
While ( there are more panels ) do
    panel := GetTask( queue )
    (1) panel_symbolic_factor( panel )
        Skip all BUSY descendant supernodes
    (2) panel_numeric_factor( panel )
        Perform updates from all DONE
        Wait for BUSY supernodes to become DONE
    (3) inner_factor( panel )
End while
```

- Up to 25-30% machine peak, 20 processors, Cray C90/J90, SGI Origin
- Model speedup by critical path: 10~100
  - [Demmel/Gilbert/Li '99]
Parallelism from Separator Tree

- Ordering using graph partitioning
Matrix Distribution on Large Distributed-memory Machine

- 2D block cyclic recommended for many linear algebra algorithms
  - Better load balance, less communication, and BLAS-3
2D Block Cyclic Layout for Sparse L and U (in SuperLU_DIST)

- Better for GE scalability, load balance
Scalability and Isoefficiency Analysis

- Model problem: matrix from 11 pt Laplacian on $k \times k \times k$ (3D) mesh; Nested dissection ordering
  - $N = k^3$
  - Factor nonzeros: $O(N^{4/3})$
  - Number of floating-point operations: $O(N^2)$
  - Total communication overhead: $O(N^{4/3} \sqrt{P})$
    (assuming $P$ processors arranged as $\sqrt{P} \times \sqrt{P}$ grid)

- Isoefficiency function: Maintain constant efficiency if “Work” increases proportionally with “Overhead”:
  \[ N^2 = c \cdot N^{4/3} \sqrt{P}, \text{ for some const. } c \]

  This is equivalent to:
  - $N^{4/3} = c^2 \cdot P$ (Memory-processor relation)
  - Parallel efficiency can be kept constant if the memory-per-processor is constant, same as dense LU in ScALPAPACK
  - $N^2 = c^3 \cdot P^{3/2}$ (Work-processor relation)
Scalability

- 3D KxKxK cubic grids, scale $N^2 = K^6$ with $P$ for constant work per processor
- Achieved 12.5 and 21.2 Gflops on 128 processors
- Performance sensitive to communication latency
  - Cray T3E latency: 3 microseconds (~ 2702 flops)
  - IBM SP latency: 8 microseconds (~ 11940 flops)
### Irregular Matrices

| Name   | Application                          | Datatype | N      | |A| / N Sparsity | |L\U| (10^6) | Fill-ratio |
|--------|--------------------------------------|----------|--------|-----------------|--------------|------------|
| g500   | Quantum Mechanics (LBL)               | Complex  | 4,235,364 | 13              | 3092.6       | 56.2       |
| matrix181 | Fusion, MHD eqns (PPPL)               | Real     | 589,698  | 161             | 888.1        | 9.3        |
| dds15  | Accelerator, Shape optimization (SLAC)| Real     | 834,575  | 16              | 526.6        | 40.2       |
| matick | Circuit sim. MNA method (IBM)        | Complex  | 16,019  | 4005            | 64.3         | 1.0        |

- **Sparsity-preserving ordering:** MeTis applied to structure of $A' + A$
Performance on IBM Power5 (1.9 GHz)

- Up to 454 Gflops factorization rate
Performance on IBM Power3 (375 MHz)

Quantum mechanics, complex: $N = 2$ million
Summary

- Important kernel for science and engineering applications, used in practice on a regular basis
- Good implementation on high-performance machines requires a large set of tools from CS and NLA
- Performance more sensitive to latency than dense case

Survey of other sparse direct solvers:
- $LL^T$, $LDL^T$, LU, QR
- Platforms: sequential, shared-memory, distributed-memory, out-of-core
Open Problems

- Much room for optimizing parallel performance
  - Automatic tuning of blocking parameters
  - Use of modern programming language to hide latency (e.g., UPC)

- Graph partitioning ordering for unsymmetric LU

- Scalability of sparse triangular solve
  - Switch-to-dense, partitioned inverse

- Efficient incomplete factorization (ILU preconditioner) - both sequential and parallel

- Optimal complexity sparse factorization
  - In the spirit of fast multipole method, but for matrix inversion
  - J. Xia’s dissertation (May 2006)

- New latency-avoidance LU and QR factorizations
Adoptions of SuperLU

- Over 6,000 downloads each year, 2004-2006
- Industrial
  - FEMLAB
  - HP Mathematical Library
  - NAG
  - Numerical Python
  - Visual Numerics: IMSL
- Academic/Lab:
  - In other ACTS Tools: PETSc, Hypre
  - M3D, NIMROD (simulate fusion reactor plasmas)
  - Omega3P (accelerator design, SLAC)
  - OpenSees (earthquake simulation, UCB)
  - DSpice (parallel circuit simulation, SNL)
  - Trilinos (object-oriented framework encompassing various solvers, SNL)
  - NIKE (finite element code for structural mechanics, LLNL)
Extra Slides
Numerical Pivoting

- **Goal of pivoting** is to control element growth in \( L \) & \( U \) for stability
  - For sparse factorizations, often relax the pivoting rule to trade with better sparsity and parallelism (e.g., threshold pivoting, static pivoting)

- **Partial pivoting** used in sequential SuperLU (GEPP)
  - Can force diagonal pivoting (use diagonal threshold)
  - Hard to implement scalably for sparse factorization

- **Static pivoting** used in SuperLU_DIST (GESP)
  - Before factor, scale and permute \( A \) to maximize diagonal: \( P_r D_r A D_c = A' \)
  - \( P_r \) is found by a weighted bipartite matching algorithm on \( G(A) \)
  - During factor \( A' = LU \), replace tiny pivots by \( \sqrt{\varepsilon \| A \|} \), without changing data structures for \( L \) & \( U \)
  - If needed, use a few steps of iterative refinement to improve the first solution
  - Quite stable in practice
Static Pivoting via Weighted Bipartite Matching

- Maximize the diag. entries: sum, or product (sum of logs)
- Hungarian algo. or the like (MC64): $O(n^*(m+n)*\log n)$
- Auction algo. (more parallel): $O(n^*m^*\log(n^*C))$
  - J. Riedy's dissertation (expected Dec. 2006?)
Numerical Accuracy: GESP versus GEPP
**Blocking in Sparse GE**

- Exploit dense submatrices in L & U factors

- **Benefits of Supernodes:**
  - Permit use of Level 3 BLAS (e.g., matrix-matrix mult.)
  - Reduce inefficient indirect addressing.
  - Reduce symbolic time by traversing supernodal graph.
Parallel Symbolic Factorization [Grigori/Demmel/Li ’06]

- Parallel ordering with ParMETIS on $G(A'+A)$
- Separator tree (binary) to guide computation
  - Each step: one row of $U$, one column of $L$
  - Within each separator: 1D block cyclic distribution
  - Send necessary contribution to parent processor

- Results:
  - Reasonable speedup: up to $6\times$
  - $5\times$ reduction in maximum per-processor memory needs
  - Need improve memory balance
Application 1: Quantum Mechanics

- Scattering in a quantum system of three charged particles

- Simplest example is ionization of a hydrogen atom by collision with an electron:
  \[ e^- + H \rightarrow H^+ + 2e^- \]

- Seek the particles’ wave functions represented by the time-independent Schrodinger equation

- First solution to this long-standing unsolved problem [Recigno, McCurdy, et al. Science, 24 Dec 1999]
Finite difference leads to complex, unsymmetric systems, very ill-conditioned
- Diagonal blocks have the structure of 2D finite difference Laplacian matrices
  - Very sparse: nonzeros per row $\leq 13$
- Off-diagonal block is a diagonal matrix
- Between 6 to 24 blocks, each of order between 200K and 350K
- Total dimension up to 8.4 M

- Too much fill if use direct method . . .
SuperLU_DIST as Preconditioner

- SuperLU_DIST as block-diagonal preconditioner for CGS iteration
  \[ M^{-1}A \mathbf{x} = M^{-1}\mathbf{b} \]
  \[ M = \text{diag}(A_{11}, A_{22}, A_{33}, \ldots) \]
- Run multiple SuperLU_DIST simultaneously for diagonal blocks
- No pivoting, nor iterative refinement

- 12 to 35 CGS iterations @ 1 ~ 2 minute/iteration using 64 IBM SP processors
  \[ \rightarrow \text{Total time: 0.5 to a few hours} \]
Complex, unsymmetric

- $N = 2\ M$, $NNZ = 26\ M$
- Fill-ins using Metis: $1.3\ G$ (50x fill)
- Factorization speed
  - 10x speedup (4 to 128)
  - Up to 30 Gflops
Application 2: Accelerator Cavity Design

- Calculate cavity mode frequencies and field vectors
- Solve Maxwell equation in electromagnetic field
- Omega3P simulation code developed at SLAC

Omega3P model of a 47-cell section of the 206-cell Next Linear Collider accelerator structure

Individual cells used in accelerating structure
Finite element methods lead to large sparse generalized eigensystem $K \mathbf{x} = \lambda M \mathbf{x}$

- Real symmetric for lossless cavities; Complex symmetric when lossy in cavities
- Seek interior eigenvalues (tightly clustered) that are relatively small in magnitude
Speed up Lanczos convergence by shift-invert

- Seek largest eigenvalues, well separated, of the transformed system
  \[ M (K - \sigma M)^{-1} x = \mu M x \]
  \[ \mu = 1 / (\lambda - \sigma) \]

- The Filtering algorithm [Y. Sun]
  - Inexact shift-invert Lanczos + JOCC (Jacobi Orthogonal Component Correction)

- We added exact shift-invert Lanczos (ESIL)
  - PARPACK for Lanczos
  - SuperLU_DIST for shifted linear system
  - No pivoting, nor iterative refinement
• Total eigensolver time: $N = 1.3\ M$, $NNZ = 20\ M$
Largest Eigen Problem Solved So Far

- DDS47, quadratic elements
  - N = 7.5 M, NNZ = 304 M
  - 6 G fill-ins using Metis

- 24 processors (8x3)
  - Factor: 3,347 s
  - 1 Solve: 61 s
  - Eigensolver: 9,259 s (~2.5 hrs)
    - 10 eigenvalues, 1 shift, 55 solves
Model Problem

- Discretized system $Ax = b$ from certain PDEs, e.g., 5-point stencil on $n \times n$ grid, $N = n^2$
- Nested dissection ordering gave optimal complexity in exact arithmetic [Hoffman/Martin/Ross]
  - Factorization cost: $O(n^3)$
Superfast Factorization: Exploit Low-rank Property

- Consider top-level dissection:
  - S is full
    - Needs \(O(n^3)\) to find \(u_3\)
  - But, off-diagonal blocks of S has low numerical ranks (e.g. 10~15)
    - \(U_3\) can be computed in \(O(n)\) flops
- Generalizing to multilevel dissection: all diagonal blocks corresp. to the separators have the similar low rank structure
- Low rank structures can be represented by hierarchical semi-separable (HSS) matrices [Gu et al.] (... think about SVD)
- Factorization complexity ... essentially linear
  - 2D: \(O(p\ n^2)\), \(p\) is related to the problem and tolerance (numerical rank)
  - 3D: \(O(c(p)\ n^3)\), \(c(p)\) is a polynomial of \(p\)

\[
\begin{pmatrix}
A_{11} & 0 & A_{13} \\
0 & A_{22} & A_{23} \\
A_{31} & A_{32} & A_{33}
\end{pmatrix}
\begin{pmatrix}
\mathbf{u}_1 \\
\mathbf{u}_2 \\
\mathbf{u}_3
\end{pmatrix}
=
\begin{pmatrix}
f_1 \\
f_2 \\
f_3
\end{pmatrix}
\]

\[
S\ u_3 = f_3 - A_{31}A_{11}^{-1}f_1 - A_{32}A_{22}^{-1}f_2
\]
Results for the Model Problem

- Flops and times comparison

![Graphs showing flops and times comparison for Traditional multifrontal and Superfast multifrontal methods.](chart)

Scaling factors (flops_{2n}/flops_n):
- 7.49, 7.70, 7.83, 7.91
- 5.44, 4.81, 4.44, 4.24

Scaling factors (time_{2n}/time_n):
- 6.16, 6.40, 6.45, 6.87
- 5.27, 4.75, 4.41, 4.20
Research Issues

- Analysis of 3D problems, and complex geometry
- Larger tolerance $\rightarrow$ preconditioner (another type of ILU)
  - If SPD, want all the low rank structures to remain SPD
    (“Schur-monotonic” talk by M. Gu, Wed, 5/3)
- Performance tuning for many small dense matrices (e.g. 10~20)
- Switching level in a hybrid solver
  - Benefits show up only for large enough mesh
- Local ordering of unknowns
  - E.g.: node ordering within a separator affects numerical ranks
- Parallel algorithm and implementation