Sparse Matrices and Numerical Linear Algebra Software

Last Lecture (not Last Supper)

Radu Trîmbiţaş

“Babeş-Bolyai” University

May 20, 2008
Sparse vs. Dense Matrices

- A *sparse matrix* is a matrix with enough zeros that it is worth taking advantage of them [Wilkinson]
- A *structured matrix* has enough structure that it is worthwhile to use it (e.g. Toeplitz)
- A *dense matrix* is neither sparse nor structured
Most operations should give the same results for sparse and full matrices.

Sparse matrices are never created automatically, but once created they propagate.

Performance is important – but usability, simplicity, completeness, and robustness are more important.

Storage for a sparse matrix should be $O(\text{nonzeros})$.

Time for a sparse operation should be close to $O(\text{flops})$. 
Data Structures for Matrices

Full:
- Storage: Array of real (or complex) numbers
- Memory: $n_{rows} \times n_{cols}$

Sparse:
- Compressed column storage
- Memory: About $1.5 \times n_{nz} + 0.5 \times n_{cols}$
Compressed Column Format - Observations

- Element look-up: $O(\log \ #\text{elements in column})$ time
- Insertion of new nonzero very expensive
- Sparse vector = Column vector (not Row vector)
Graphs and Sparsity: Cholesky Factorization

- **Fill**: New nonzeros in factor
- **Symmetric Gaussian Elimination**:

  ```latex
  \text{for } j=1 \text{ to } N
  \text{Add edges between } j \text{’s higher numbered neighbors}
  ```
Permutations of the 2-D Model Problem

- 2-D Model Problem: Poisson’s Equation on $n \times n$ finite difference grid
- Total number of unknowns $n^2 = N$
- Theoretical results for the fill-in:
  - With natural permutation: $O(N^{3/2})$ fill
  - With any permutation: $\Omega(N \log N)$ fill
  - With a nested dissection permutation: $O(N \log N)$ fill
A separator in a graph $G$ is a set $S$ of vertices whose removal leaves at least two connected components.

A nested dissection ordering for an $N$-vertex graph $G$ numbers its vertices from 1 to $N$ as follows:

1. Find a separator $S$, whose removal leaves connected components $T_1$, $T_2$, $\ldots$, $T_k$.
2. Number the vertices of $S$ from $N - |S| + 1$ to $N$.
3. Recursively, number the vertices of each component: $T_1$ from 1 to $|T_1|$, $T_2$ from $|T_1| + 1$ to $|T_1| + |T_2|$, etc.
4. If a component is small enough, number it arbitrarily.

It all boils down to finding good separators!
Banded orderings (Reverse Cuthill-McKee, Sloan, etc):
- Try to keep all nonzeros close to the diagonal
- Theory, practice: Often wins for “long, thin” problems

Minimum degree:
- Eliminate row/col with fewest nonzeros, add fill, repeat
- Hard to implement efficiently – current champion is “Approximate Minimum Degree” [Amestoy, Davis, Duff]
- Theory: Can be suboptimal even on 2-D model problem
- Practice: Often wins for medium-sized problems
Heuristic Fill-Reducing Matrix Permutations

- Nested dissection:
  - Find a separator, number it last, proceed recursively
  - Theory: Approximately optimal separators $\implies$ approximately optimal fill and flop count
  - Practice: Often wins for very large problems

- The best modern general-purpose orderings are ND/MD hybrids
Fill-Reducing Permutations in MATLAB

- **Reverse Cuthill-McKee:**
  - \( p = \text{symrcm}(A); \)
  - Symmetric permutation: \( A(p,p) \) often has smaller bandwidth than \( A \)

- **Symmetric approximate minimum degree:**
  - \( p = \text{symamd}(A); \)
  - Symmetric permutation: \( \text{chol}(A(p,p)) \) sparser than \( \text{chol}(A) \)

- **Nonsymmetric approximate minimum degree:**
  - \( p = \text{colamd}(A); \)
  - Column permutation: \( \text{lu}(A(:,p)) \) sparser than \( \text{lu}(A) \)

- **Symmetric nested dissection:**
  - Not built into MATLAB, several versions in the MESHPART toolbox
Complexity of Direct Methods

- Time and space to solve any problem on any well-shaped finite element mesh with $N$ nodes:

<table>
<thead>
<tr>
<th></th>
<th>1-D</th>
<th>2-D</th>
<th>3-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>$O(N)$</td>
<td>$O(N \log N)$</td>
<td>$O(N^{4/3})$</td>
</tr>
<tr>
<td>Time</td>
<td>$O(N)$</td>
<td>$O(N^{3/2})$</td>
<td>$O(N^2)$</td>
</tr>
</tbody>
</table>
**Basic Linear Algebra Subroutines (BLAS)**

- Standardized interface for simple vector and matrix operations
- Manufacturers provide optimized implementations for their machines

**History:**

- BLAS1 (1970s) – Vector operations: \( \alpha = x^T y, \ y = \alpha x + y \)
- BLAS2 (mid 1980s) – Matrix-vector operations: \( y = Ax + y \)
- BLAS3 (late 1980s) – Matrix-matrix operations: \( C = AB + C \)

Efficient cache-aware implementations give almost peak performance for BLAS3 operations

High level algorithms (Gaussian elimination, etc) use BLAS but no other machine dependent code
- Performance and portability
Modern computers use a memory hierarchy
From fast/expensive to cheap/slow: Registers, L1 cache, L2 cache, local memory, remote memory, secondary memory
Fast algorithms perform many operations on each memory block to minimize memory access (cache reuse)
Only BLAS3 has potential for very high performance

<table>
<thead>
<tr>
<th>BLAS</th>
<th>Memory Refs</th>
<th>Flops</th>
<th>Flops/Memory Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1 ( (y = \alpha x + y) )</td>
<td>3n</td>
<td>2n</td>
<td>2/3</td>
</tr>
<tr>
<td>Level 2 ( (y = Ax + y) )</td>
<td>( n^2 )</td>
<td>2( n^2 )</td>
<td>2</td>
</tr>
<tr>
<td>Level 3 ( (C = AB + C) )</td>
<td>4( n^2 )</td>
<td>2( n^3 )</td>
<td>( n/2 )</td>
</tr>
</tbody>
</table>
For high performance write algorithms in terms of BLAS3 operations.
BLAS Implementations

- **Vendor provided:**
  - Intel Math Kernel Library (MKL), AMD Core Math Library (ACML)
  - Sun Performance Library
  - SGI Scientific Computing Software Library

- **Automatically Tuned Linear Algebra Software (ATLAS)**
  - Analyzes hardware to produce BLAS libraries for any platform
  - Used in MATLAB, precompiled libraries freely available
  - Sometimes outperforms vendor libraries

- **GOTO BLAS (mainly for Intel processors)**
  - Manually optimized assembly code, currently the fastest implementation
Calling BLAS from C

- BLAS standard based on Fortran 77:
  - All memory must be preallocated
  - All variables are passed by reference

- Example: Double precision matrix-matrix multiply ($C = AB + C$):
  
  ```c
  dgemm_(&transa,&transb,&m,&n,&k,&alpha,A,&lda,
          B,&ldb,&beta,C,&ldc);
  ```
  
  - transa, etc: Matrix transpose ('T') or not ('N')
  - lda, etc: Leading dimensions of matrices
  - Some platforms/compilers do not require the trailing underscore
  - In C++, declare functions with extern "C"

- See also C BLAS interface in ATLAS
LAPACK

- Standard library for dense/banded linear algebra
  - Linear systems: $Ax = b$
  - Least squares problems: $\min_x \|Ax - b\|_2$
  - Eigenvalue problems: $Ax = \lambda x$, $Ax = \lambda Bx$
  - Singular value decomposition (SVD): $A = U\Sigma V^T$

- Algorithms use BLAS3 as much as possible
- Used by MATLAB (since version 6)
- LAPACK Search Engine useful for finding routines
LAPACK Performance

- Matrix-matrix multiply and LU factorization as function of matrix size
- About 80% of peak performance for LU factorization of large matrices
Sparse Solver Packages

- **UMFpack** (Unsymmetric MultiFrontal method)
  - Used in MATLAB (since version 7.1), no parallel version

- **PARADISO**
  - Serial and shared memory, used in Intel MKL

- **SuperLU**
  - Versions for serial and parallel computers (shared/distributed)
  - “Static pivoting” for distributed machines (increase small pivots, iterative refinement for accuracy)

- **MUMPS** (MUltifrontal Massively Parallel sparse direct Solver)
  - Versions for serial and parallel computers (distributed)