Numerical Methods for Solving Large Linear Systems

Paolo Bientinesi

AICES, RWTH Aachen
pauldj@aices.rwth-aachen.de

3rd LHC Detector Alignment Workshop
June 15-16, 2009
CERN, Switzerland
Modularity and Data Movement

Linear Algebra operations decomposed into simpler operations.
Linear Algebra operations decomposed into simpler operations.

**BLAS-1:**

\[
\begin{align*}
    y & := y + \alpha x \\
    \text{dot} & := \alpha + x^T y
\end{align*}
\]

\(x, y \in \mathbb{R}^n\)

**BLAS-2:**

\[
\begin{align*}
    y & := y + Ax \\
    y & := A^{-1} x
\end{align*}
\]

\(L \in \mathbb{R}^{n \times n}, x, y \in \mathbb{R}^n\)

\(L \in \mathbb{R}^{n \times n} \wedge \text{triangular}\)

**BLAS-3:**

\[
\begin{align*}
    C & := C + AB \\
    C & := L^{-1} B
\end{align*}
\]

\(A, B, C \in \mathbb{R}^{n \times n}\)

\(L \in \mathbb{R}^{n \times n} \wedge \text{triangular}\)
Modularity and Data Movement

Linear Algebra operations decomposed into simpler operations.

**BLAS-1:**
\[ y := y + \alpha x \quad x, y \in \mathbb{R}^n \]
\[ \text{dot} := \alpha + x^T y \]

**BLAS-2:**
\[ y := y + Ax \quad L \in R^{n \times n}, x, y \in \mathbb{R}^n \]
\[ y := A^{-1}x \quad L \in R^{n \times n} \wedge \text{triangular} \]

**BLAS-3:**
\[ C := C + AB \quad A, B, C \in R^{n \times n} \]
\[ C := L^{-1}B \quad L \in R^{n \times n} \wedge \text{triangular} \]

<table>
<thead>
<tr>
<th>BLAS</th>
<th>#FLOPS</th>
<th>Mem. refs.</th>
<th>Ratio</th>
<th>Proc. use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>2n</td>
<td>3n</td>
<td>2/3</td>
<td>low</td>
</tr>
<tr>
<td>Level 2</td>
<td>2n^2</td>
<td>n^2</td>
<td>2</td>
<td>medium-low</td>
</tr>
<tr>
<td>Level 3</td>
<td>2n^3</td>
<td>4n^2</td>
<td>n/2</td>
<td>very high</td>
</tr>
</tbody>
</table>
High Performance Computing

Sparse Matrices

Linear Systems

Error Analysis

Eigensolvers
What is a Sparse Matrix?

- Sparse matrix: concept of convenience.
- No formal definition in terms of number of non-zeros, patterns, properties.
- Practical definition in terms of cost: operation count, storage requirement, ...
What is a Sparse Matrix?

- Sparse matrix: concept of convenience.
- No formal definition in terms of number of non-zeros, patterns, properties.
- Practical definition in terms of cost: operation count, storage requirement, ... 

A matrix is sparse when it has enough zeros that pays off to exploit them \(\text{(Wilkinson)}\)

Objectives:
- Storage space.
- Accessing, inserting matrix elements.
- Matrix operations and fill in.
Sparse Matrices — Structured Matrices — Dense Matrices

Structured matrices:
- bidiagonal, tridiagonal, banded, blocked, etc.

Small number of non-zeros (NNZ), but known structure!
Sparse Matrices

Sparsity:
interactions between particles, components, neighbors, degrees of freedom.

The finer the discretization, the higher the sparsity.

- 50% of NNZ → NOT a sparse matrix.
- 10% of NNZ: if \( n = 100,000 \), then 10,000 iterations per row.
  Still not very sparse.
- Large sparse matrices: \( \text{NNZ} \ll 1\% \).
Direct methods: LU factorization, Cholesky, . . . .
Iterative methods: Gauss-Seidel, Conjugate Gradient, GMRES, . . . .
Direct methods: LU factorization, Cholesky, . . .
Iterative methods: Gauss-Seidel, Conjugate Gradient, GMRES, . . .

Dense vs. Sparse (vs. Structured)

Dense matrices

Sparse matrices

Direct methods

Iterative methods
## Direct Methods vs. Iterative Methods

<table>
<thead>
<tr>
<th>Direct</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy: fixed (cond num)</td>
<td>Variable accuracy</td>
</tr>
<tr>
<td>Matrix-matrix operations</td>
<td>Matrix-vector operations</td>
</tr>
<tr>
<td>Cost: $O(n^3)$, predictable</td>
<td>Cost not known: $k\ O(n^2)$</td>
</tr>
<tr>
<td>Factorization re-use: multiple right-hand sides</td>
<td>Convergence (spectrum)</td>
</tr>
<tr>
<td>Fill-in, reordering</td>
<td>Preconditioning</td>
</tr>
<tr>
<td></td>
<td>Stopping criteria</td>
</tr>
<tr>
<td></td>
<td>Single/few right-hand sides</td>
</tr>
<tr>
<td></td>
<td>Exploit sparsity</td>
</tr>
</tbody>
</table>
## Direct Methods vs. Iterative Methods

<table>
<thead>
<tr>
<th>Direct</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy: fixed (cond num)</td>
<td>Variable accuracy</td>
</tr>
<tr>
<td>Matrix-matrix operations</td>
<td>Matrix-vector operations</td>
</tr>
<tr>
<td>Cost: $O(n^3)$, predictable</td>
<td>Cost not known: $k O(n^2)$</td>
</tr>
<tr>
<td>Factorization re-use: multiple right-hand sides</td>
<td>Convergence (spectrum)</td>
</tr>
<tr>
<td>Fill-in, reordering</td>
<td>Preconditioning</td>
</tr>
<tr>
<td></td>
<td>Stopping criteria</td>
</tr>
<tr>
<td>BLAS, LAPACK, FLAME &amp; PETSc</td>
<td>Single/few right-hand sides</td>
</tr>
<tr>
<td>HSL, MUMPS, UMFPACK, . . .</td>
<td>Exploit sparsity</td>
</tr>
<tr>
<td></td>
<td>PETSc, Trilinos, . . .</td>
</tr>
</tbody>
</table>

**Google: “Linear Algebra Software”**

Survey of freely available libraries
### 30k, 1 core

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>7GB</td>
</tr>
<tr>
<td>LU fact</td>
<td>1640 secs (28m)</td>
</tr>
<tr>
<td>Ax = b</td>
<td>1.9 secs (1 rhs)</td>
</tr>
<tr>
<td>AX = B</td>
<td>172 secs (2k rhs)</td>
</tr>
</tbody>
</table>

---

Paolo Bientinesi (AICES, RWTH Aachen)

Solving Large Linear Systems

June 15th, 2009
<table>
<thead>
<tr>
<th></th>
<th>Memory:</th>
<th>LU fact:</th>
<th>Ax = b:</th>
<th>AX = B:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>30k, 1 core</strong></td>
<td>7GB</td>
<td>1640 secs (28m)</td>
<td>1.9 secs (1 rhs)</td>
<td>172 secs (2k rhs)</td>
</tr>
<tr>
<td><strong>40k, 8-core</strong></td>
<td>12GB</td>
<td>513 secs (9m)</td>
<td>2.1secs</td>
<td>93.5secs</td>
</tr>
</tbody>
</table>

**Parallelism — Multi-cores**

Paolo Bientinesi (AICES, RWTH Aachen)

Solving Large Linear Systems

June 15th, 2009 12 / 17
### Parallelism — Multi-cores

<table>
<thead>
<tr>
<th>30k, 1 core</th>
<th>40k, 8-core</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory:</strong> 7GB</td>
<td><strong>Memory:</strong> 12GB</td>
</tr>
<tr>
<td><strong>LU fact:</strong> 1640 secs (28m)</td>
<td><strong>LU fact:</strong> 513 secs (9m)</td>
</tr>
<tr>
<td><strong>Ax = b:</strong> 1.9 secs (1 rhs)</td>
<td><strong>Ax = b:</strong> 2.1 secs</td>
</tr>
<tr>
<td><strong>AX = B:</strong> 172 secs (2k rhs)</td>
<td><strong>AX = B:</strong> 93.5 secs</td>
</tr>
</tbody>
</table>

### 100k — extrapolating:

| Memory | 80GB |
| # of 8-cores | 8 with 10-12GB each |
| LU fact | 20 minutes |
High Performance Computing

Sparse Matrices

Linear Systems

Error Analysis

Eigen solvers
Perturbation Results

\[ Ax = b \]

- Acquisition and representation errors:

\[ A \rightarrow \hat{A} = A + \delta A \quad b \rightarrow \hat{b} = b + \delta b \]

\[(A + \delta A)\hat{x} = b + \delta b\]

- \[ \hat{x} = x + \delta x \]

\[ \frac{\|\delta x\|}{\|x\|} = \mu(A) \frac{\|\delta A\|/\|A\| + \|\delta b\|/\|b\|}{1 - \mu(A)\|\delta A\|/\|A\|} \]

- \( \mu(A) = \|A\|\|A^{-1}\| \) is the **condition number** of \( A \)

Sensitivity to perturbations. Independent of the solution method. Well vs. ill conditioned problems.
Backward/Forward Stability

\[ f : X \rightarrow Y \quad \hat{f} \text{ is an implementation of } f \]

\[ \text{Question: } |f - \hat{f}| ? \]

Exact arithmetic

\[ x \rightarrow f(x) \]

Floating point arithmetic

\[ x \rightarrow \hat{f}(x) \]

\[ (\hat{x} \rightarrow \hat{f}(\hat{x})) \]
Backward/Forward Stability

\[ f : X \rightarrow Y \]
\[ \hat{f} \text{ is an implementation of } f \]

- Question: \[ |f - \hat{f}| \]?

- Exact arithmetic
  \[ x \rightarrow f(x) \]

- Floating point arithmetic
  \[ x \rightarrow \hat{f}(x) \]
  \[ (\hat{x} \rightarrow \hat{f}(\hat{x})) \]

- **Forward stability:** \[ \forall x \ | f(x) - \hat{f}(x) | \text{ is small.} \]

- Let \( \bar{x} \) be such that \( \hat{f}(x) = f(\bar{x}) \). Exact sol. to a different probl.

- **Backward stability:** \[ \forall x \ \exists \bar{x} . \ | | x - \bar{x} | \text{ is small.} \]

Factorizations are backward stable.
Iterative methods \( \rightarrow \) convergence & convergence rate.
Symmetric Eigenproblem

\[ AV = V\Lambda \]
Symmetric Eigenproblem

\[ AV = V\Lambda \]

Three stages:
1) Reduction to tridiagonal form
2) **Tridiagonal eigensolver** \((TZ = Z\Lambda)\)
3) Backtransformation

Reduction: \(O(n^3)\), perfectly stable, destroys sparsity.

Backtransformation: matrix-matrix multiplication, \(O(n^3)\), perfectly stable.

Tridiagonal eigensolvers: MR, QR, D&C, etc.

Cost: \(O(n^2) - O(n^3)\)

Accuracy:
\[ \|Z^T Z - I\| \leq cn\epsilon \wedge \|TZ - Z\Lambda\| \leq cn\epsilon \|T\| \]
Symmetric Eigenproblem

\[ AV = V \Lambda \]

- Three stages:
  1) Reduction to tridiagonal form
  2) **Tridiagonal eigensolver** \((TZ = Z \Lambda)\)
  3) Backtransformation

- Reduction: \(O(n^3)\), perfectly stable, destroys sparsity.
- Backtransformation: matrix-matrix multiplication, \(O(n^3)\), perfectly stable.
- Tridiagonal eigensolvers: \(MR^3\), QR, D&C, \ldots

Cost: \(O(n^2)\) — \(O(n^3)\)

Accuracy: \[\|Z^T Z - I\| \leq c n \epsilon \quad \land \quad \|TZ - Z \Lambda\| \leq c n \epsilon \|T\|\]
Symmetric Eigenproblem

\[ AV = V\Lambda \]

- Three stages:
  1) Reduction to tridiagonal form
  2) **Tridiagonal eigensolver** \((TZ = Z\Lambda)\)
  3) Backtransformation

- Reduction: \(O(n^3)\), perfectly stable, destroys sparsity.
- Backtransformation: matrix-matrix multiplication, \(O(n^3)\), perfectly stable.
- Tridiagonal eigensolvers: MR\(^3\), QR, D&C, . . .

- Cost: \(O(n^2) \rightarrow O(n^3)\)
- Accuracy: \(\|Z^T Z - I\| \leq c n\epsilon \land \|TZ - Z\Lambda\| \leq c n\epsilon\|T\|\)

- Eigenvalues AND(?) eigenvectors? How many? Accuracy?
- LAPACK, PMR3, ScaLAPACK. Sparse solver: ARPACK.
Future?

- Exploiting structures, properties.
- Knowledge from applications.
- Massive parallelism: hybrid multi-core + distributed architectures.

Thank you!

For more information: pauldj@aices.rwth-aachen.de
Future?

- Exploiting structures, properties.
- Knowledge from applications.
- Massive parallelism: hybrid multi-core + distributed architectures.

Thank you!

For more information: pauldj@aices.rwth-aachen.de