Overview

- Dense matrix
  - BLAS (serial)
  - ATLAS (serial/threaded)
  - LAPACK (serial)
  - Vendor-tuned LAPACK (shared memory parallel)
  - ScaLAPACK/PLAPACK (distributed memory parallel)
  - FLAME (an algorithm derivation framework)

- Sparse matrix
  - PETSc

- Further reading
The Basic Linear Algebra Subprograms (BLAS) consist of a set of lower-level linear algebra operations

- Level 1: vector-vector
  - O(n) operations on O(n) data
  - Bandwidth to memory is a limiting factor

- Level 2: matrix-vector
  - O(n^2) operations on O(n^2) data
  - Vectors kept in cache

- Level 3: matrix-matrix
  - O(n^3) operations on O(n^2) data
  - Blocked matrices kept in cache

- Netlib’s BLAS is a reference implementation

Examples

\[ y \leftarrow \alpha x + y \]

\[ y \leftarrow \alpha A x + \beta y \]

\[ T x = y \quad \text{(Triangular T)} \]

\[ C \leftarrow \alpha A B + \beta C \]

\[ B \leftarrow \alpha T^{-1} B \quad \text{(Triangular T)} \]
GotoBlas and Vendor-Tuned BLAS

- Implemented by Kazushige Goto
- Optimized for cache and Translation Lookaside Buffer (TLB)
- Restrictive open-source license
- Licensed to vendors for vendor-tuned BLAS libraries

Vendor-tuned BLAS
- Accelerate framework (Apple)
- MLK (Intel)
- ACML (AMD)
- ESSL (IBM)
- MLIB (HP)
- Sun performance library
The Automatically Tuned Linear Algebra Software (ATLAS) is a self-tuned BLAS version. Installation tests numerical kernels and (other parts of) the code to determine which parameters are best for a particular machine, e.g. blocking, loop unrolling, ... Faster than the reference implementation. Freely available.
DGEMM

Pentium4 (3.6 GHz)

Image source: Robert van de Geijn (TACC)

11/29/10  HPC Fall 2010
DGEMM

Image source: Robert van de Geijn (TACC)
11/29/10
DGEMM

Power 5 (1.9 GHz)

Image source: Robert van de Geijn (TACC)
LAPACK

- Linear Algebra PACKage (LAPACK) written in Fortran
- Built on BLAS
- Like BLAS, has standard set of APIs (Application Programming Interfaces)
  - Data type: real and complex, single and double precision
  - Matrix shapes: general dense, diagonal, bidiagonal, tridiagonal, banded, trapeziodal, Hessenberg
  - Matrix properties: general, orthogonal, positive definite, Hermitian, symmetric
- Reference implementation from Netlib
- Vendor-tuned versions available
  - Some for shared memory parallel
ScaLAPACK/PLAPACK

- ScaLAPACK/PLAPACK are versions of LAPACK for distributed memory MIMD parallel machines
  - Subset of LAPACK routines
- ScaLAPACK is built on BLAS and MPI
- ScaLAPACK reference implementation from Netlib

- PLAPACK is a project at UT Austin (TACC)
FLAME

- Formal Linear Algebra Methods Environment (FLAME)
- LAPACK code is hard to write/read/maintain/alter
- “Transform the development of dense linear algebra libraries from an art reserved for experts to a science that can be understood by novice and expert alike”
  - Notation for expressing algorithms
  - A methodology for systematic derivation of algorithms using loop invariants
  - Application Program Interfaces (APIs) for representing the algorithms in code
  - Tools for mechanical derivation, implementation and analysis of algorithms and implementations
Algorithm: \[ A := LU\text{-}BLK\text{-}VAR5(A) \]

Partition \[ A \rightarrow \begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \]

where \( A_{TL} \) is \( 0 \times 0 \)

while \( m(A_{TL}) < m(A) \) do

Determine block size \( b \)

Repartition

\[
\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}
\]

where \( A_{11} \) is \( b \times b \)

\[
A_{11} = LU(A_{11})
\]

\[
A_{12} = \text{TRILU}(A_{11})^{-1}A_{12}
\]

\[
A_{21} = A_{21} \text{TRIU}(A_{11})^{-1}
\]

\[
A_{22} = A_{22} - A_{21}A_{12}
\]

Continue with

\[
\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}
\]

endwhile

\[
\text{FLA\_Part\_2x2}( A, \ &ATL, \ &ATR, \\
&ABL, \ &ABR, \ 0, 0, \ FLA\_TL )
\]

while \( \text{FLA\_Obj\_length}( ATL ) < \text{FLA\_Obj\_length}( A ) \) do

\[
b = \min( \text{FLA\_Obj\_length}( ABR ), \ nb\_alg )
\]

\[
\text{FLA\_Repart\_2x2\_to\_3x3}( ATL, \ &/\!/ \ ATR, \ &A00, \ &/\!/ \ &A01, \ &A02, \\
&/\!/ \ &A10, \ &/\!/ \ &A11, \ &A12, \\
&ABL, \ &/\!/ \ &ABR, \ &A20, \ &/\!/ \ &A21, \ &A22, \\
b, \ b, \ FLA\_BR )
\]

endwhile

\[
\text{FLA\_Trsm}( \text{FLA\_LEFT}, \text{FLA\_LOWER\_TRIANGULAR}, \\
\text{FLA\_NO\_TRANSPOSE}, \text{FLA\_UNIT\_DIAG}, \\
\text{FLA\_ONE}, \ A11, \ A12 )
\]

\[
\text{FLA\_Trsm}( \text{FLA\_RIGHT}, \text{FLA\_UPPER\_TRIANGULAR}, \\
\text{FLA\_NO\_TRANSPOSE}, \text{FLA\_NONUNIT\_DIAG}, \\
\text{FLA\_ONE}, \ A11, \ A21 )
\]

\[
\text{FLA\_Gemm}( \text{FLA\_NO\_TRANSPOSE}, \text{FLA\_NO\_TRANSPOSE}, \\
\text{FLA\_MINUS\_ONE}, \ A21, \ A12, \text{FLA\_ONE}, \ A22 )
\]

endwhile

\[
\text{FLA\_Cont\_with\_3x3\_to\_2x2}( \ &ATL, \ &/\!/ \ &ATR, \ A00, \ A01, \ &/\!/ \ A02, \\
&/\!/ \ A10, \ A11, \ &/\!/ \ A12, \\
&ABL, \ &/\!/ \ &ABR, \ A20, \ A21, \ &/\!/ \ A22, \\
\text{FLA\_TL} )
\]
Operation: \([L] = \text{TrinvLVar1}(L)\)

Partition

\[
L = \begin{pmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{pmatrix}
\]

such that \(L_{22}\) is empty

Loop invariant:

\[
\begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} = \begin{pmatrix} \text{inv} \\\ \\ \text{inv} \end{pmatrix}
\]

while \(L_{22} \neq I\)

Simplify:

\[
\begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} = \begin{pmatrix} \text{inv} \\\ \\ \text{inv} \end{pmatrix}
\]

Loop invariant before the update:

\[
\begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} = \begin{pmatrix} \text{inv} \\\ \\ \text{inv} \end{pmatrix}
\]

\[
L_{11} := \text{inv}
\]

\[
L_{12} := -L_{12}.L_{21}.L_{11}
\]

Continue with

\[
\begin{pmatrix} \text{inv} \\ \text{inv} \end{pmatrix}
\]

Loop invariant after the update:

\[
\begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} = \begin{pmatrix} \text{inv} \\\ \\ \text{inv} \end{pmatrix}
\]

end while
LU w/ Pivoting on 8 Cores
4 x AMD 2.4GHz dual-core Opteron 880

LU (with pivoting) performance with various libraries ($m = p$, $n = p$)

- ○ ACML 3.60
- ▲ GotoBLAS 1.09 + LAPACK 3.0
- □ LAPACK 3.0 + GotoBLAS 1.09
- ● FLAME + ACML 3.60
- ○ FLAME + GotoBLAS 1.09 + LAPACK 3.0
- + FLAME + LAPACK 3.0 + GotoBLAS 1.09

GotoBLAS
FLAME
LAPACK

Image source: Robert van de Geijn (TACC)
11/29/10 HPC Fall 2010
QR Factorization on 8 Cores
4 x AMD 2.4GHz dual-core Opteron 880

Image source: Robert van de Geijn (TACC)

11/29/10
Cholesky on 8 Cores
4 x AMD 2.4GHz dual-core Opteron 880

Image source: Robert van de Geijn (TACC)

11/29/10
PETSc

- Portable, Extensible Toolkit for Scientific Computation (PETSc) for distributed memory MIMD parallel machines
  - Vector/matrix formats and array operations (serial and parallel)
  - Linear and nonlinear solvers
  - Limited ODE integrators
  - Limited grid/data management (serial and parallel)

- Built on BLAS, LAPACK, and MPI

- Basically a solver library for general sparse matrices
  - User writes main() program
  - User orchestrates computation via object creations
  - User controls the basic flow of the PETSc program
  - PETSc propagates errors from underlying libs
**PETSc Numerical Components**

<table>
<thead>
<tr>
<th><strong>Nonlinear Solvers (SNES)</strong></th>
<th><strong>Time Steppers (TS)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-based Methods</td>
<td>Euler</td>
</tr>
<tr>
<td>Line Search</td>
<td>Backward Euler</td>
</tr>
<tr>
<td>Trust Region</td>
<td>Pseudo Time Stepping</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Krylov Subspace Methods (KSP)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>GMRES</td>
</tr>
<tr>
<td>CG</td>
</tr>
<tr>
<td>CGS</td>
</tr>
<tr>
<td>Bi-CG-STAB</td>
</tr>
<tr>
<td>TFQMR</td>
</tr>
<tr>
<td>Richardson</td>
</tr>
<tr>
<td>Chebychev</td>
</tr>
<tr>
<td>Other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Preconditioners (PC)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Additive Schwartz</td>
</tr>
<tr>
<td>Block Jacobi</td>
</tr>
<tr>
<td>Jacobi</td>
</tr>
<tr>
<td>ILU</td>
</tr>
<tr>
<td>ICC</td>
</tr>
<tr>
<td>LU (Sequential only)</td>
</tr>
<tr>
<td>Others</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Matrices (Mat)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compressed Sparse Row (AIJ)</td>
</tr>
<tr>
<td>Blocked Compressed Sparse Row (BAIJ)</td>
</tr>
<tr>
<td>Block Diagonal (BDIAG)</td>
</tr>
<tr>
<td>Dense</td>
</tr>
<tr>
<td>Matrix-free</td>
</tr>
<tr>
<td>Other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Distributed Arrays (DA)</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Vectors (Vec)</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Index Sets (IS)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Indices</td>
</tr>
<tr>
<td>Block Indices</td>
</tr>
<tr>
<td>Stride</td>
</tr>
<tr>
<td>Other</td>
</tr>
</tbody>
</table>

*Image source: PETSc project*
PETSc Flow of Control for PDEs

Image source: PETSc project
11/29/10

HPC Fall 2010
PETSc Linear Solver Example

\[ Ax = b \]

```c
KSP  ksp; /* linear solver context */
Mat   A; /* matrix */
Vec  x, b; /* solution, RHS vectors */
int n;    /* problem dimension */

MatCreate(PETSC_COMM_WORLD, PETSC_DECIDE, PETSC_DECIDE, n, n, &A);
MatSetFromOptions(A);
/* (code to assemble matrix A not shown) */
VecCreate(PETSC_COMM_WORLD, &x);
VecSetSizes(x, PETSC_DECIDE, n);
VecSetFromOptions(x);
VecDuplicate(x, &b);
/* (code to assemble RHS vector not shown)*/
KSPCreate(PETSC_COMM_WORLD, &ksp);
KSPSetOperators(ksp, A, A, DIFFERENT_NONZERO_PATTERN);
KSPSetFromOptions(ksp);
KSPSolve(ksp, b, x);
KSPDestroy(ksp);
```
PETSc Nonlinear Solver Interface: SNES

- For problems arising from PDEs
- Uses Newton-based methods
  - (Approximately) solve $F'(u_k) = -F(u_k)$
  - Update $u_{k+1} = u_k + \Delta u_k$
- Support the general solution to $F(u) = 0$
- User provides:
  - Code to evaluate $F(u)$
  - Code to evaluate Jacobian of $F(u)$
    - Or use (built-in) first-order sparse finite difference approximation
    - Or use automatic differentiation, e.g. ADIFOR and ADIC
PETSc Nonlinear Solver Example

SNES snes; /* nonlinear solver context */
Mat J; /* Jacobian matrix */
Vec x, f; /* solution, RHS vectors */
int n, its; /* problem dimension, number of iterations */
ApptCtx uc; /* user-defined application context */

MatCreate(PETSC_COMM_WORLD, n, n, &J);
VecCreate(PETSC_COMM_WORLD, n, &x);
VecDuplicate(x, &f);

SNESCreate(PETSC_COMM_WORLD, SNES_NONLINEAR_EQUATIONS, &snes);
SNESSetFunction(snes, f, EvaluateFunction, uc);
SNESSetJacobian(snes, J, EvaluateJacobian, uc);
SNESSetFromOptions(snes);

SNESsolve(snes, x, &its);

SNESDestroy(snes);
PETSc Meshes

Image source: PETSc project

11/29/10
PETSc Global vs Local Meshes

Global: each process stores a unique local set of vertices (and each vertex is owned by exactly one process)

Local: each process stores a unique local set of vertices as well as ghost nodes from neighboring processes

Image source: PETSc project
PETSc Distributed Arrays

- Form a DA:
  - DACreate1d(..., DA*)
  - DACreate2d(..., DA*)
  - DACreate3d(..., DA*)

- Create the corresponding PETSc vectors
  - DACreateGlobalVector(DA, Vec*)
  - DACreateLocalVector(DA, Vec*)

- Update ghost points (scatter global vector into local parts, including ghost points)
  - DAGlobalToLocalBegin(DA, ...)
  - DAGlobalToLocalEnd(DA, ...)
Further Reading

- Optional: [SRC] pages 621-647
- Netlib organization: www.netlib.org
- FLAME project: www.cs.utexas.edu/users/flame
- PETSc project: www.mcs.anl.gov/petsc
- Linear algebra Wiki: www.linearalgebrawiki.org