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
BLAS

- 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**

- $y \leftarrow \alpha x + y$
- $y \leftarrow \alpha Ax + \beta y$
- $Tx = y$ (Triangular $T$)
- $C \leftarrow \alpha AB + \beta C$
- $B \leftarrow \alpha T^{-1}B$ (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
ATLAS

- 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)
DGEMM

Image source: Robert van de Geijn (TACC)

11/26/12 HPC Fall 2012
DGEMM

Image source: Robert van de Geijn (TACC)
LAPACK

- Linear Algebra PACKage (LAPACK) written in Fortran
- Built on BLAS
- Standard API (Application Programming Interface)
  - 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
  - Linear least squares, eigenvalue problems, singular value decomposition, matrix factorizations (LU, QR, Cholesky, Schur)

- 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_BLK_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 \( b \times b \)

\[ A_{11} = LU(A_{11}) \]
\[ A_{12} = TRILU(A_{11})^{-1} A_{12} \]
\[ A_{21} = A_{21} \text{TRIU}(A_{11})^{-1} \]
\[ A_{22} = A_{22} - A_{21} A_{12} \]

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

\[
\text{FLA_Part_2x2(} A, &ATL, &ATR, \\
&ABL, &ABR, 0, 0, \text{FLA_TL})
\]

while \( \text{FLA_Obj_length( ATL }) < \text{FLA_Obj_length( A ))} \{

\[ b = \min( \text{FLA_Obj_length( ABR }) \), \text{nb_alg} \);

\[
\text{FLA_Repart_2x2_to_3x3(} ATL, /**/ ATR, &A00, /**/ &A01, &A02, \\
/* ************* */ /* *********************** */ &A10, /**/ &A11, &A12, \\
&ABL, /**/ &ABR, &A20, /**/ &A21, &A22, \\
b, b, \text{FLA_BR})
\]

/*-----------------------------------------------*/

\[
\text{LU_unb_var5(} A11)\)
\]

\[
\text{FLA_Trsm(} \text{FLA_LEFT, FLA_LOWER_TRIANGULAR}, \\
\text{FLA_NO_TRANSPOSE, FLA_UNIT_DIAG,} \\
\text{FLA_ONE, A11, A12})
\]

\[
\text{FLA_Trsm(} \text{FLA_RIGHT, FLA_UPPER_TRIANGULAR}, \\
\text{FLA_NO_TRANSPOSE, FLA_NONUNIT_DIAG,} \\
\text{FLA_ONE, A11, A21})
\]

\[
\text{FLA_Gemm(} \text{FLA_NO_TRANSPOSE, FLA_NO_TRANSPOSE,} \\
\text{FLA_MINUS_ONE, A21, A12, FLA_ONE, A22})
\]

/*-----------------------------------------------*/

\[
\text{FLA_Cont_with_3x3_to_2x2(} &ATL, /**/ &ATR, \\
A00, A01, /**/ A02, \\
A10, A11, /**/ A12, \\
&ABL, /**/ &ABR, A20, A21, /**/ A22, \\
\text{FLA_TL})
\]

}
AutoFLAME

Operation: \[ (L) = \text{TrinvLVart}(L) \]

Partition:
\[ L = \begin{pmatrix} \ell_{1m} & 0 \\ \ell_{m1} & \ell_{mm} \end{pmatrix} \]

where \( \ell_{mm} \) is empty.

Loop invariant:
\[ \begin{pmatrix} \ell_{1m} \\ \ell_{m1} \end{pmatrix} = \begin{pmatrix} \ell_{1m} \\ \ell_{m1} \end{pmatrix} \]

While \( L_m \to L \)

Repartition:
\[ \begin{pmatrix} \ell_{1m} \\ \ell_{m1} \end{pmatrix} = \begin{pmatrix} \ell_{1m} \\ \ell_{m1} \end{pmatrix} \]

Loop invariant before the updates:
\[ \begin{pmatrix} \ell_{1m} \\ \ell_{m1} \end{pmatrix} = \begin{pmatrix} \ell_{1m} \\ \ell_{m1} \end{pmatrix} \]

\[ L_{11} := L_{11}^{-1} \]
\[ L_{12} := \frac{L_{12}}{L_{11}} \cdot \frac{L_{21}}{L_{22}} \]

Continue with:
\[ \begin{pmatrix} \ell_{1m} \\ \ell_{m1} \end{pmatrix} = \begin{pmatrix} \ell_{1m} \\ \ell_{m1} \end{pmatrix} \]

Loop invariant after the updates:
\[ \begin{pmatrix} \ell_{1m} \\ \ell_{m1} \end{pmatrix} = \begin{pmatrix} \ell_{1m} \\ \ell_{m1} \end{pmatrix} \]
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)

11/26/12
QR Factorization on 8 Cores

4 x AMD 2.4GHz dual-core Opteron 880

QR performance with various libraries (m = p, n = p)

Image source: Robert van de Geijn (TACC)

11/26/12
Cholesky on 8 Cores
4 x AMD 2.4GHz dual-core Opteron 880

Image source: Robert van de Geijn (TACC)

11/26/12
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

Nonlinear Solvers (SNES)
- Newton-based Methods
- Line Search
- Trust Region
- Other

Time Steppers (TS)
- Euler
- Backward Euler
- Pseudo Time Stepping
- Other

Krylov Subspace Methods (KSP)
- GMRES
- CG
- CGS
- Bi-CG-STAB
- TFQMR
- Richardson
- Chebychev
- Other

Preconditioners (PC)
- Additive Schwartz
- Block Jacobi
- Jacobi
- ILU
- ICC
- LU (Sequential only)
- Others

Matrices (Mat)
- Compressed Sparse Row (AIJ)
- Blocked Compressed Sparse Row (BAIJ)
- Block Diagonal (BDIAG)
- Dense
- Matrix-free
- Other

Distributed Arrays (DA)

Vectors (Vec)

Index Sets (IS)
- Indices
- Block Indices
- Stride
- Other

Image source: PETSc project
PETSc Flow of Control for PDEs
PETSc Linear Solver Example

Ax = b

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);
/* (user-defined code to assemble matrix A not shown) */
VecCreate(PETSC_COMM_WORLD, &x);
VecSetSizes(x, PETSC_DECIDE, n);
VecSetFromOptions(x);
VecDuplicate(x, &b);
/* (user-defined code to assemble RHS vector b 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

```c
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);

SNESolve(snes, x, &its);

SNESDestroy(snes);
```
PETSc Meshes

Image source: PETSc project

11/26/12
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

11/26/12

HPC Fall 2012
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

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