High-Performance Libraries and Tools

HPC Fall 2012

Prof. Robert van Engelen

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

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{x} + \boldsymbol{y}$$

$$\boldsymbol{y} \leftarrow \alpha A \boldsymbol{x} + \beta \boldsymbol{y}$$

$$T\boldsymbol{x} = \boldsymbol{y} \qquad \text{(Triangular } T\text{)}$$

$$C \leftarrow \alpha AB + \beta C$$

$$B \leftarrow \alpha T^{-1} B \quad \text{(Triangular } T\text{)}$$

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

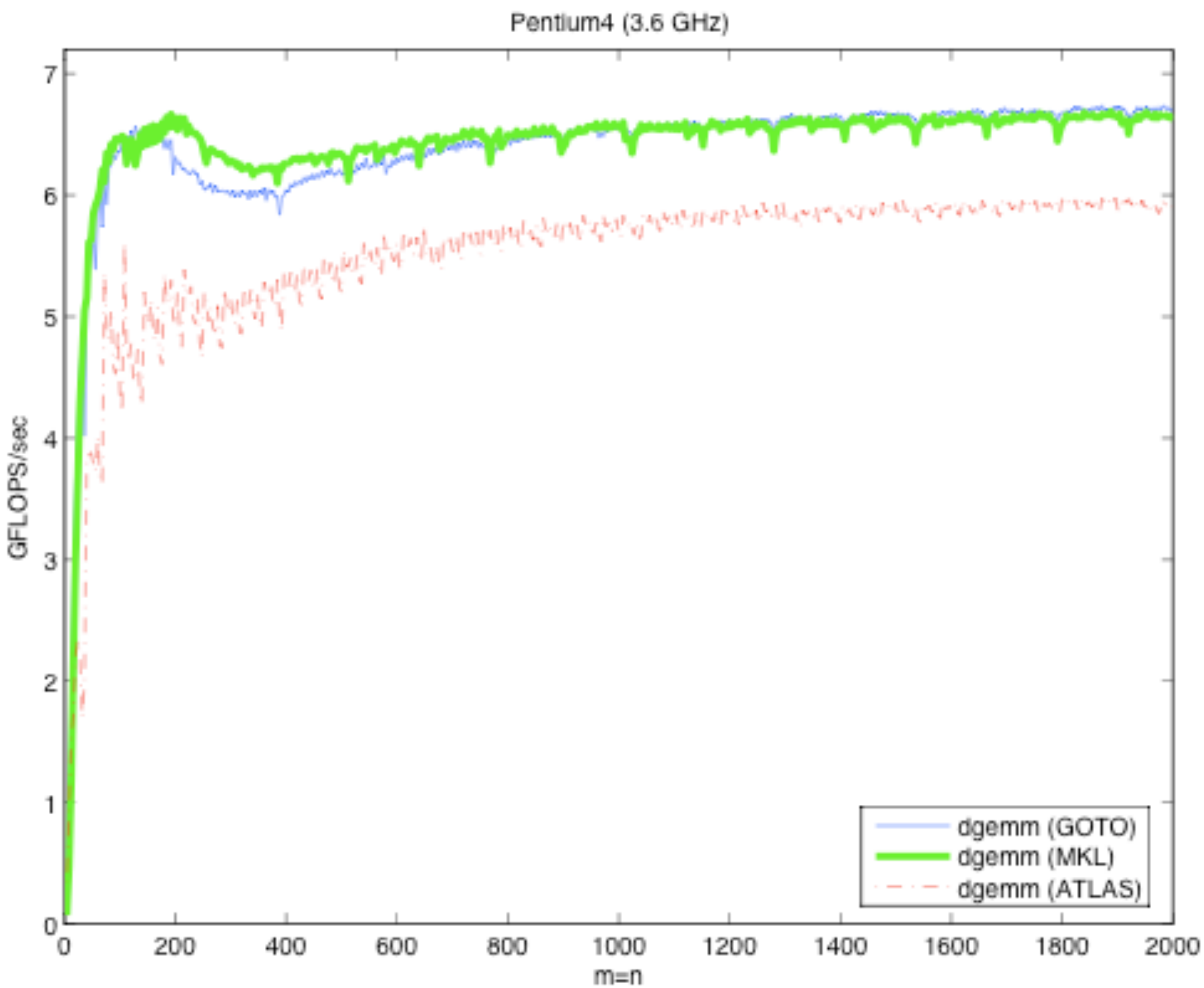


Image source: Robert van de Geijn (TACC)

DGEMM

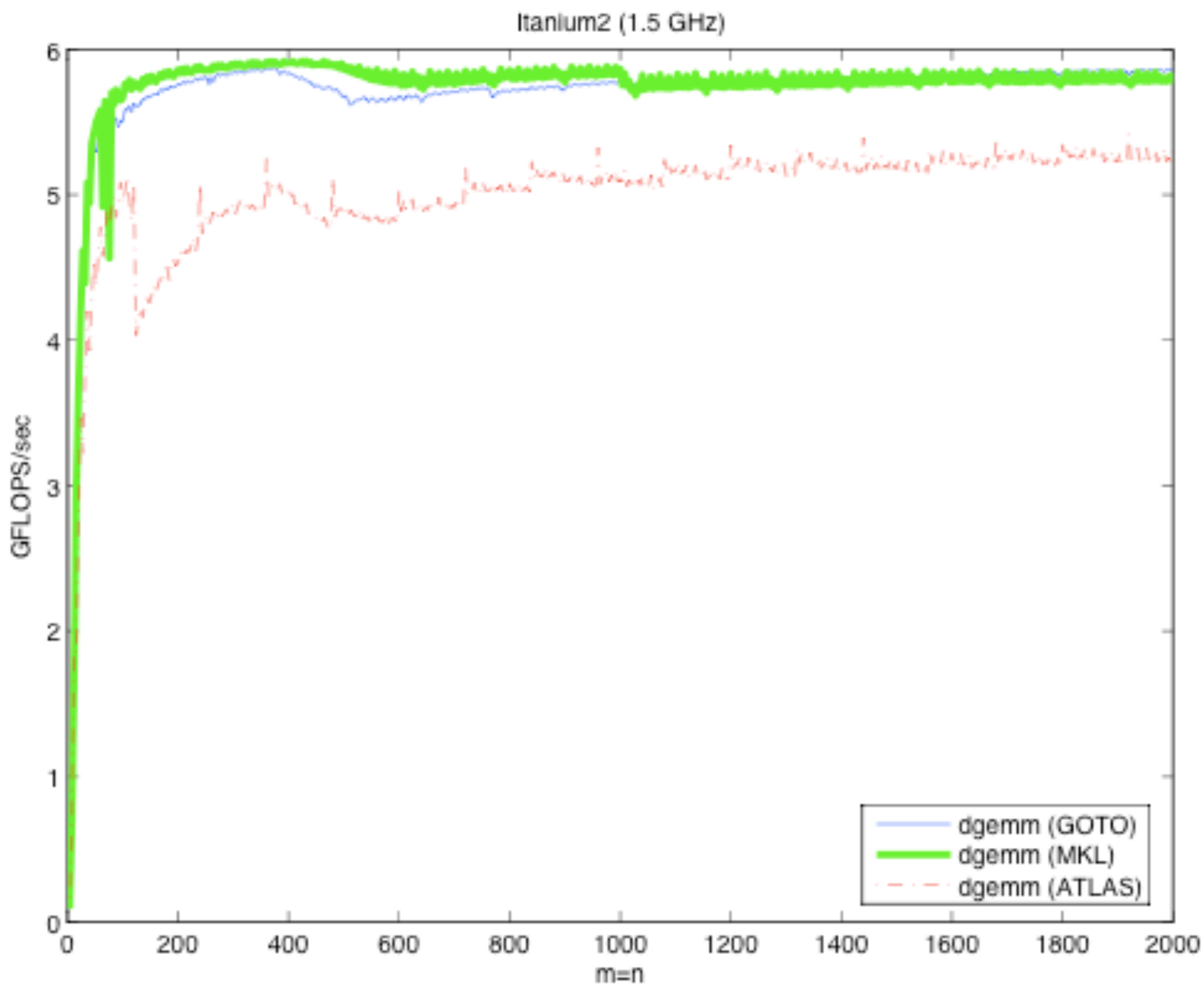


Image source: Robert van de Geijn (TACC)

DGEMM

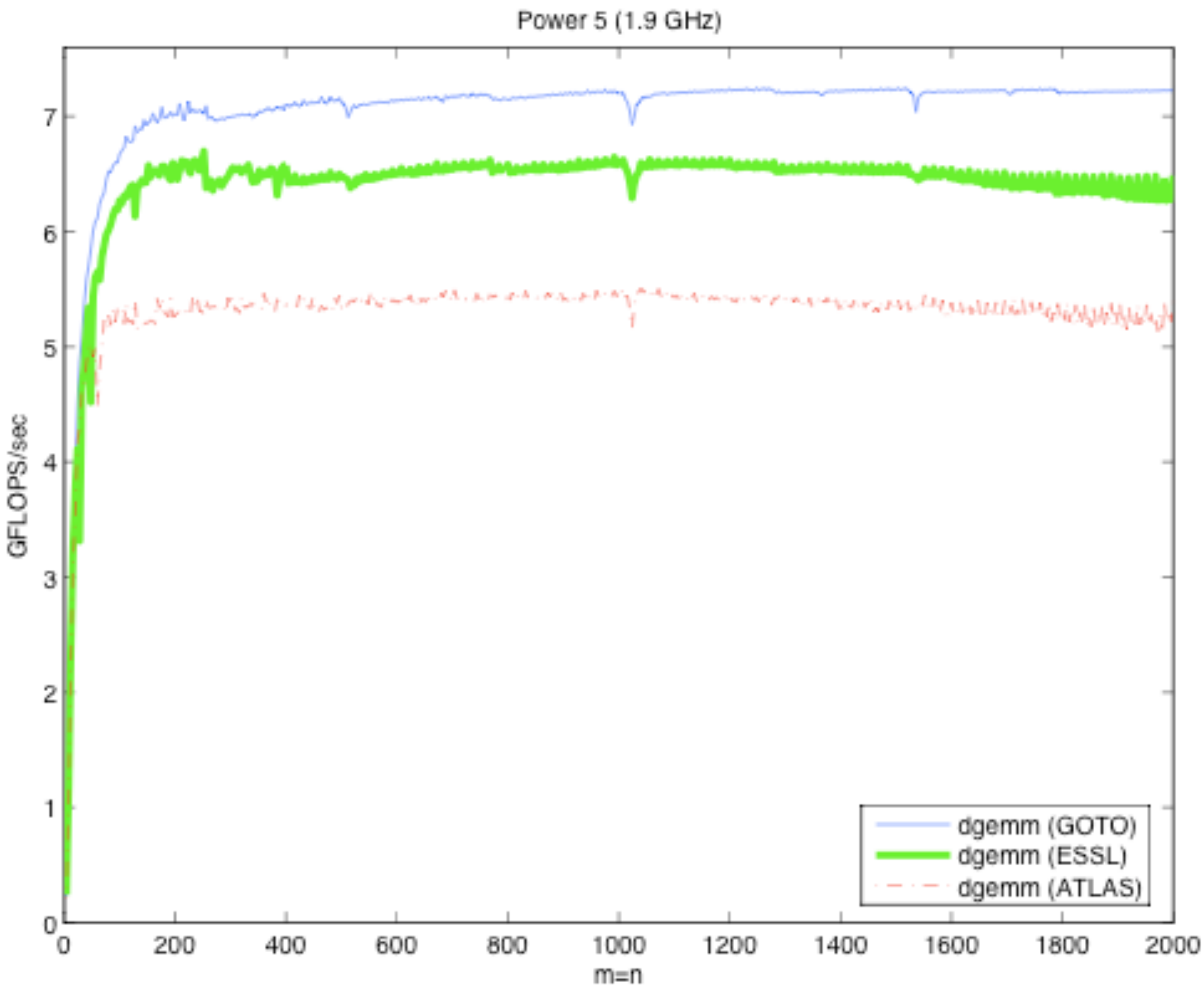


Image source: Robert van de Geijn (TACC)

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

FLAME-Derived Blocked LU

Algorithm: $[A] := \text{LU\_BLK\_VAR5}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

$A_{11} = \text{LU}(A_{11})$

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

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

$A_{22} = A_{22} - A_{21} A_{12}$

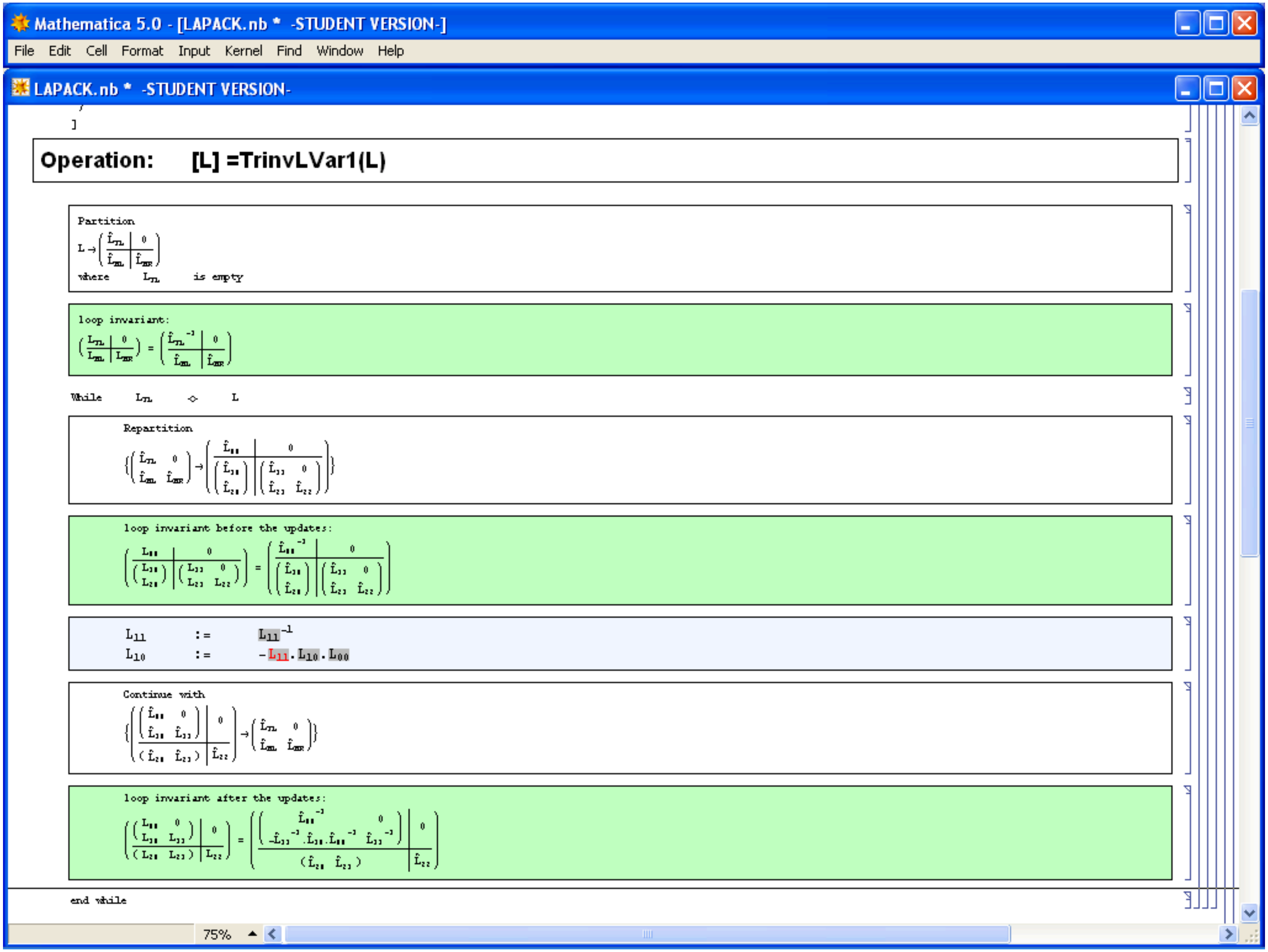
Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

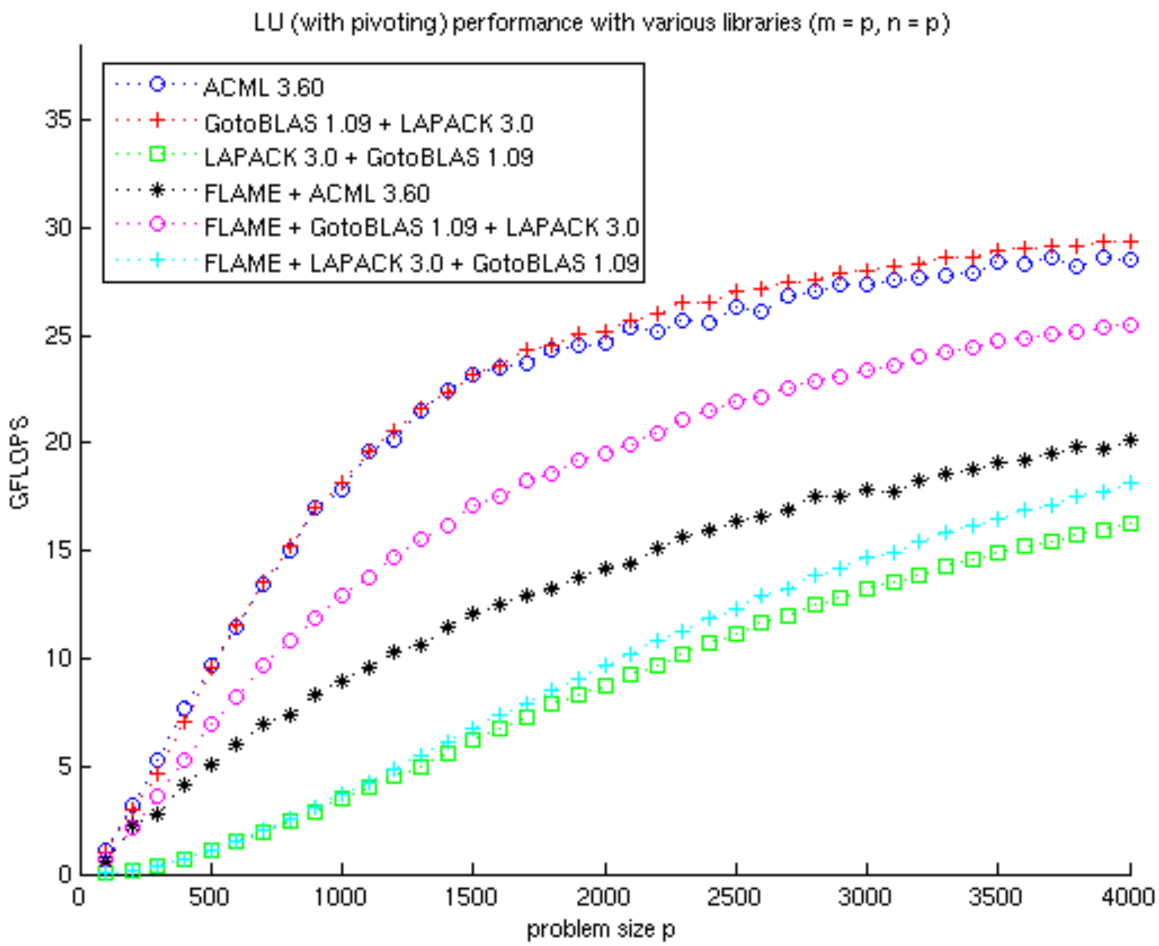
```
FLA_Part_2x2( A,    &ATL, &ATR,
                     &ABL, &ABR,     0, 0, FLA_TL );

 while (FLA_Obj_length( ATL ) < FLA_Obj_length( A )){

   b = min( FLA_Obj_length( ABR ), nb_alg );

   FLA_Repart_2x2_to_3x3
      ( ATL, /**/ ATR,        &A00, /**/ &A01, &A02,
     /* ************* */   /* ******************** */
                              &A10, /**/ &A11, &A12,
        ABL, /**/ ABR,        &A20, /**/ &A21, &A22,
        b, b, FLA_BR );
   /*------------------------------------------------*/
   LU_unb_var5( A11 );

   FLA_Trsm( FLA_LEFT, FLA_LOWER_TRIANGULAR,
             FLA_NO_TRANSPOSE, FLA_UNIT_DIAG,
             FLA_ONE, A11, A12 );

   FLA_Trsm( FLA_RIGHT, FLA_UPPER_TRIANGULAR,
             FLA_NO_TRANSPOSE, FLA_NONUNIT_DIAG,
             FLA_ONE, A11, A21 );

   FLA_Gemm( FLA_NO_TRANSPOSE, FLA_NO_TRANSPOSE,
             FLA_MINUS_ONE, A21, A12, FLA_ONE, A22 );
   /*------------------------------------------------*/
   FLA_Cont_with_3x3_to_2x2
      ( &ATL, /**/ &ATR,       A00, A01, /**/ A02,
                               A10, A11, /**/ A12,
      /* ************** */  /* ****************** */
        &ABL, /**/ &ABR,       A20, A21, /**/ A22,
        FLA_TL );
 }
```

AutoFLAME


Mathematica 5.0 - [LAPACK.nb * -STUDENT VERSION-]
File Edit Cell Format Input Kernel Find Window Help
LAPACK.nb * -STUDENT VERSION-
Operation: [L] =TrinvLVar1(L)
Partition
where is empty
loop invariant:
While
Repartition
loop invariant before the updates:
Continue with
loop invariant after the updates:
end while
75%

LU w/ Pivoting on 8 Cores

4 x AMD 2.4GHz dual-core Opteron 880



GotoBLAS

FLAME

LAPACK

Image source: Robert van de Geijn (TACC)

QR Factorization on 8 Cores

4 x AMD 2.4GHz dual-core Opteron 880

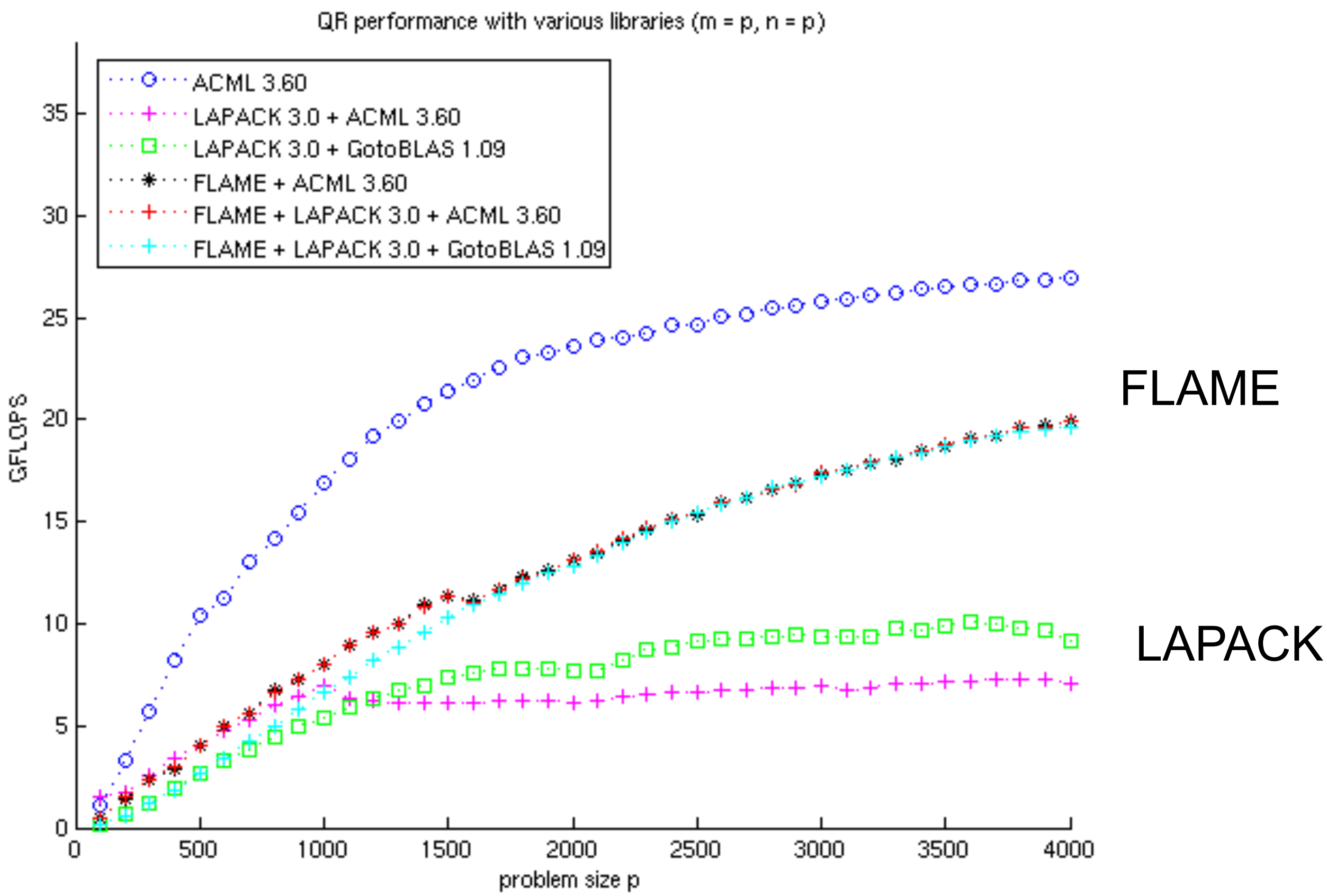


Image source: Robert van de Geijn (TACC)

Cholesky on 8 Cores

4 x AMD 2.4GHz dual-core Opteron 880

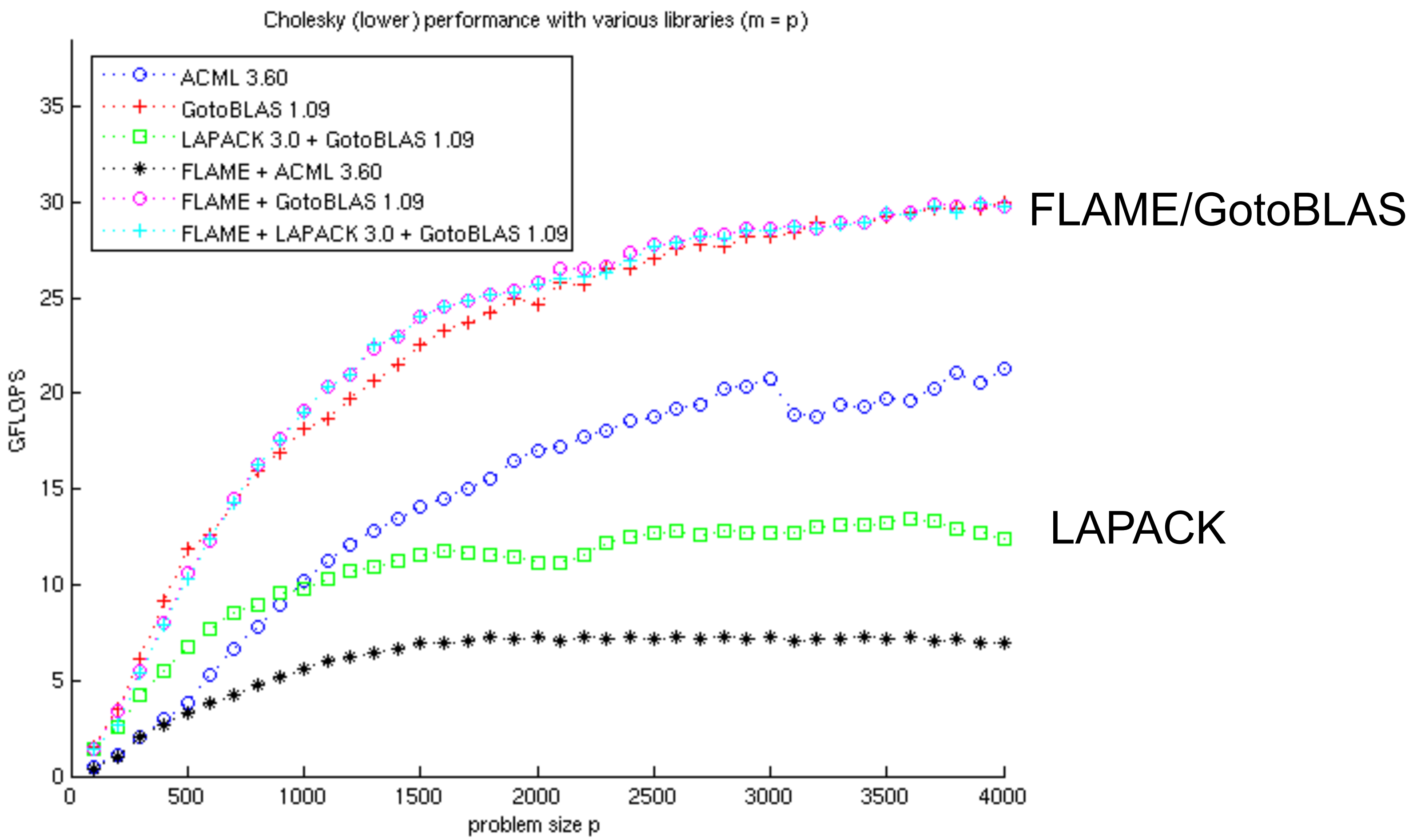


Image source: Robert van de Geijn (TACC)

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

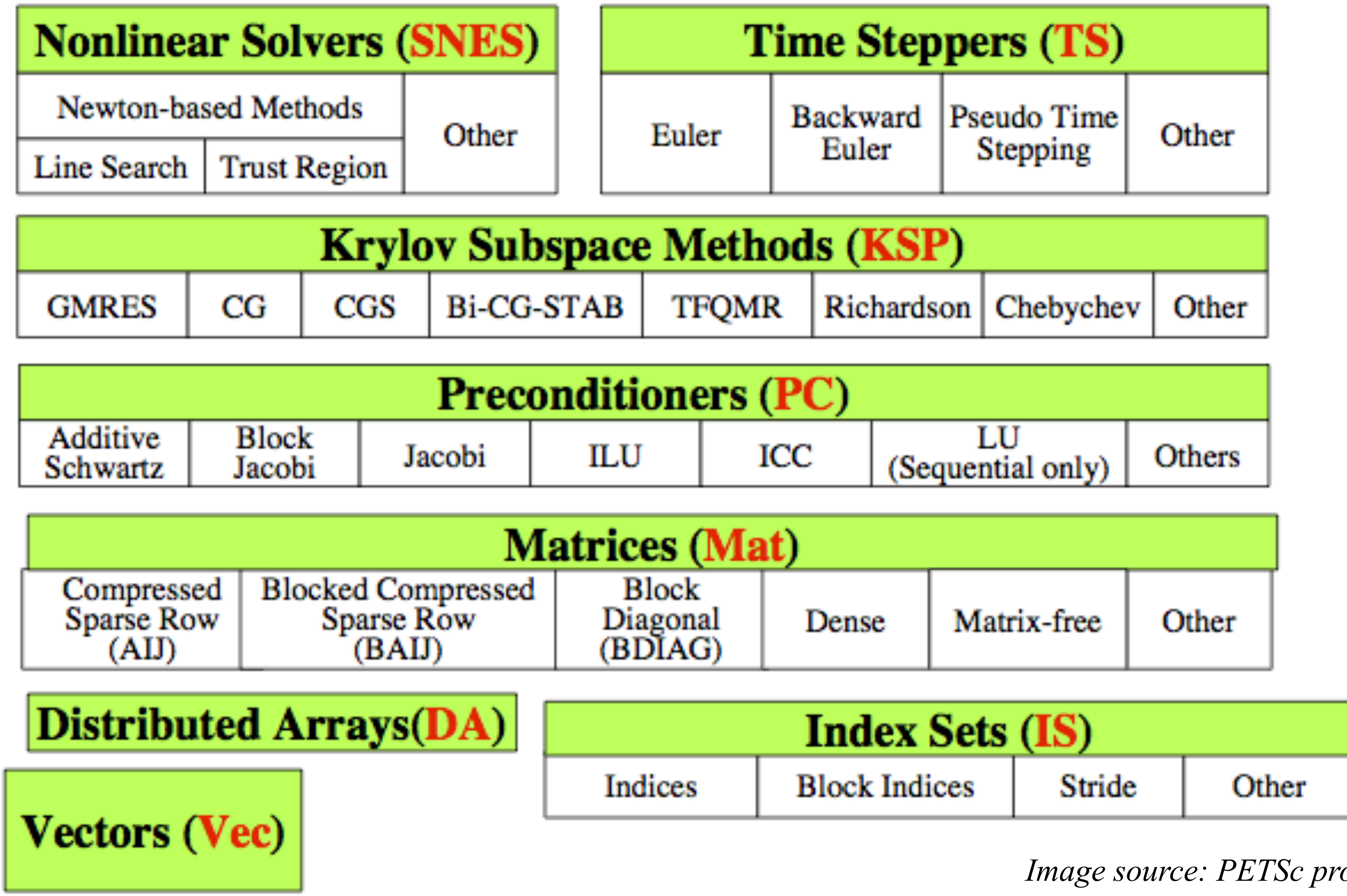


Image source: PETSc project

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 Flow of Control for PDEs

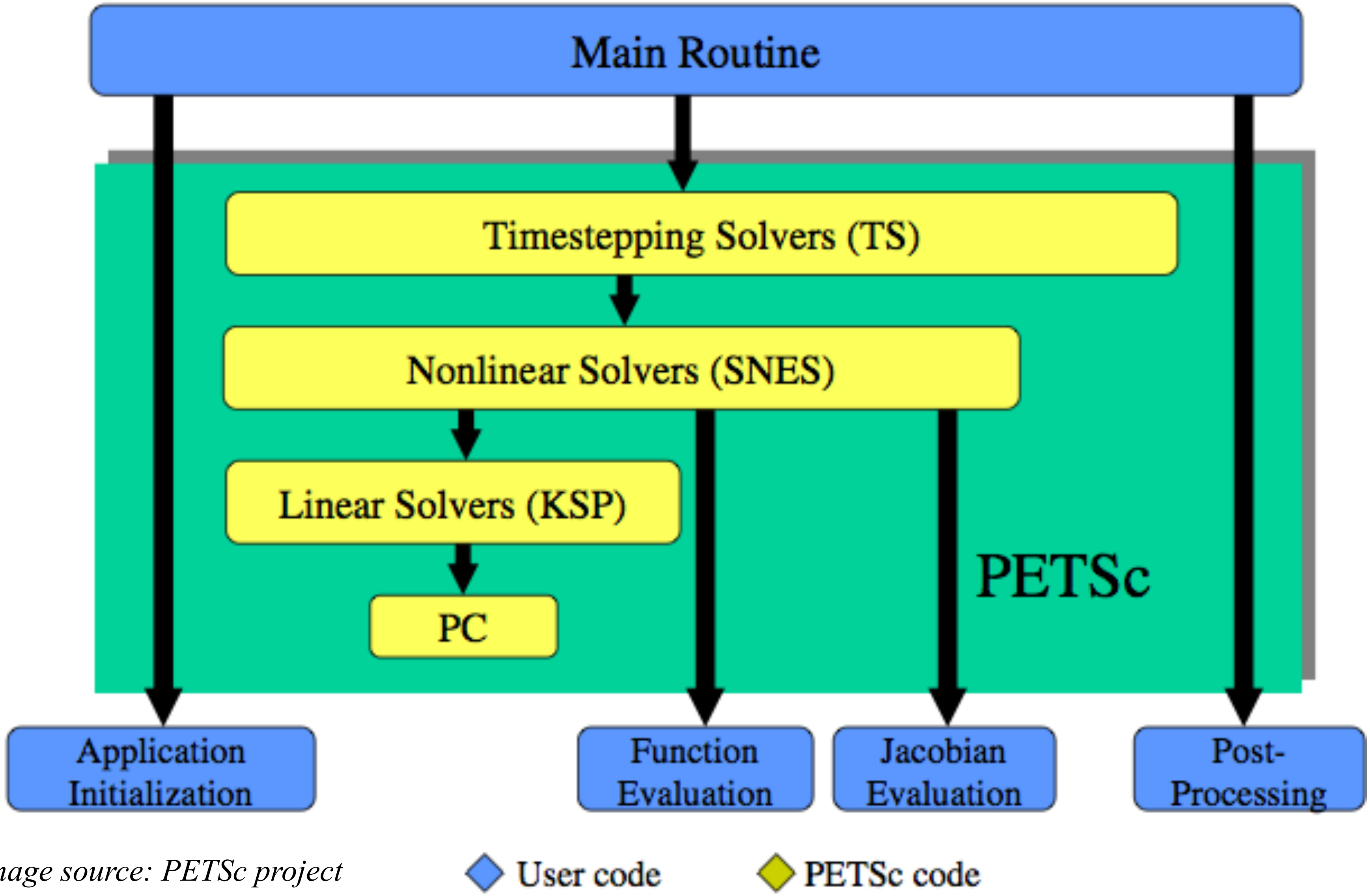


Image source: PETSc project

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

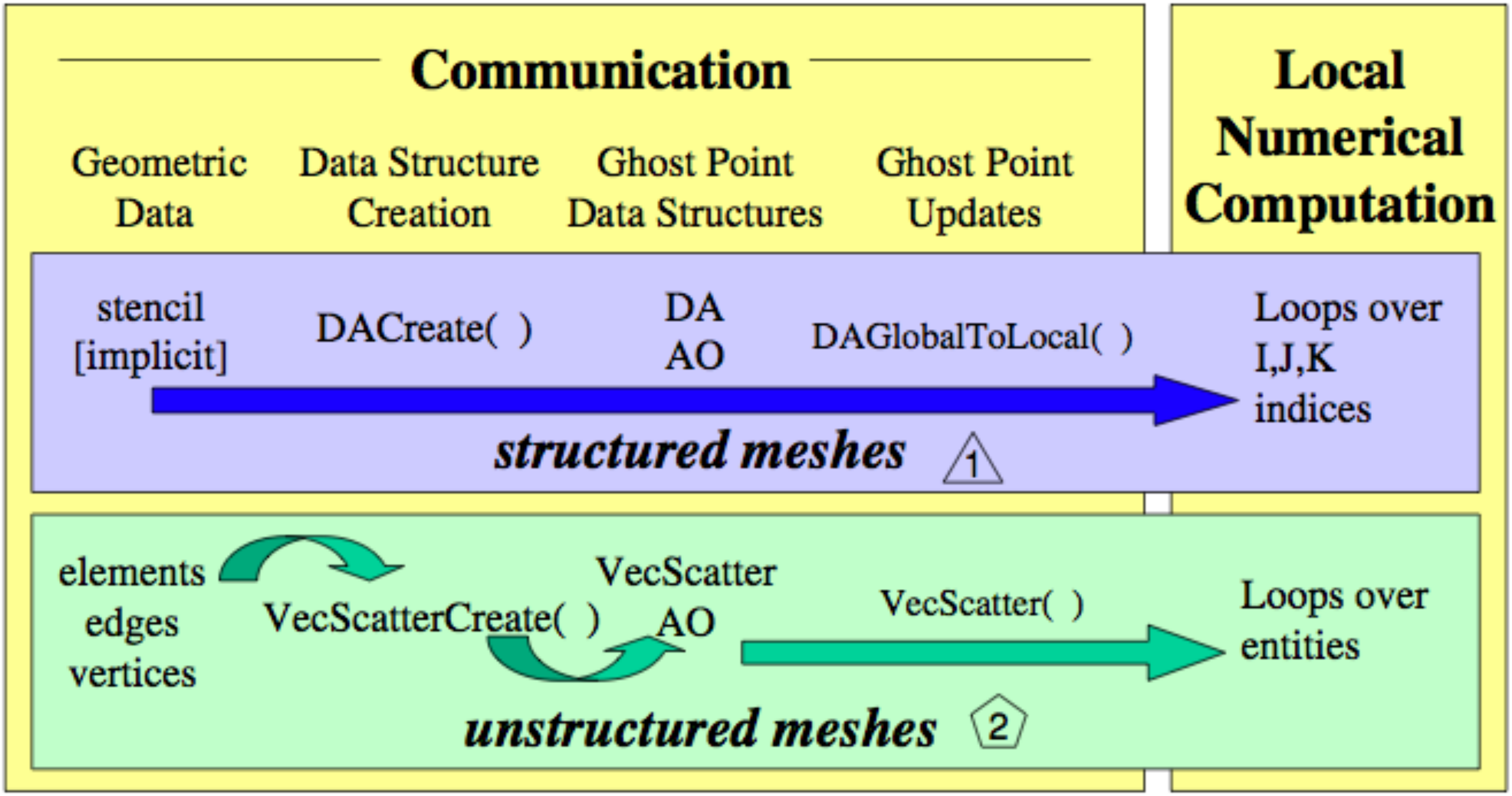
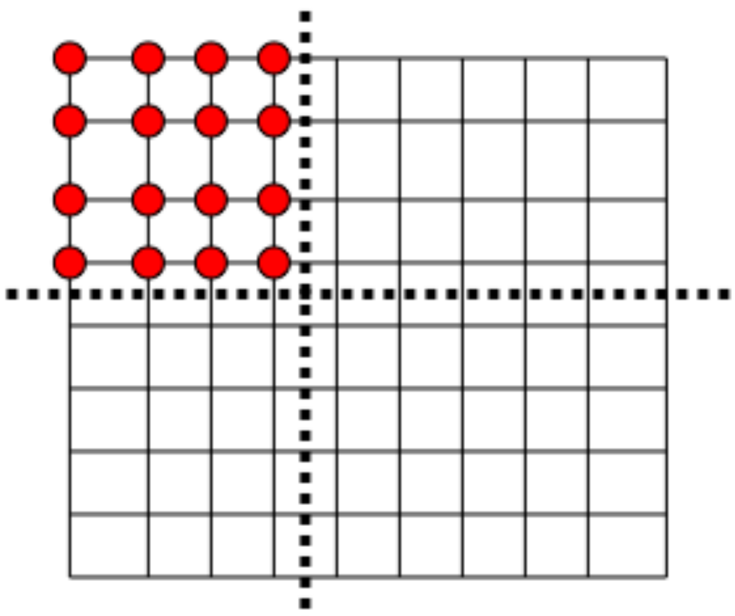
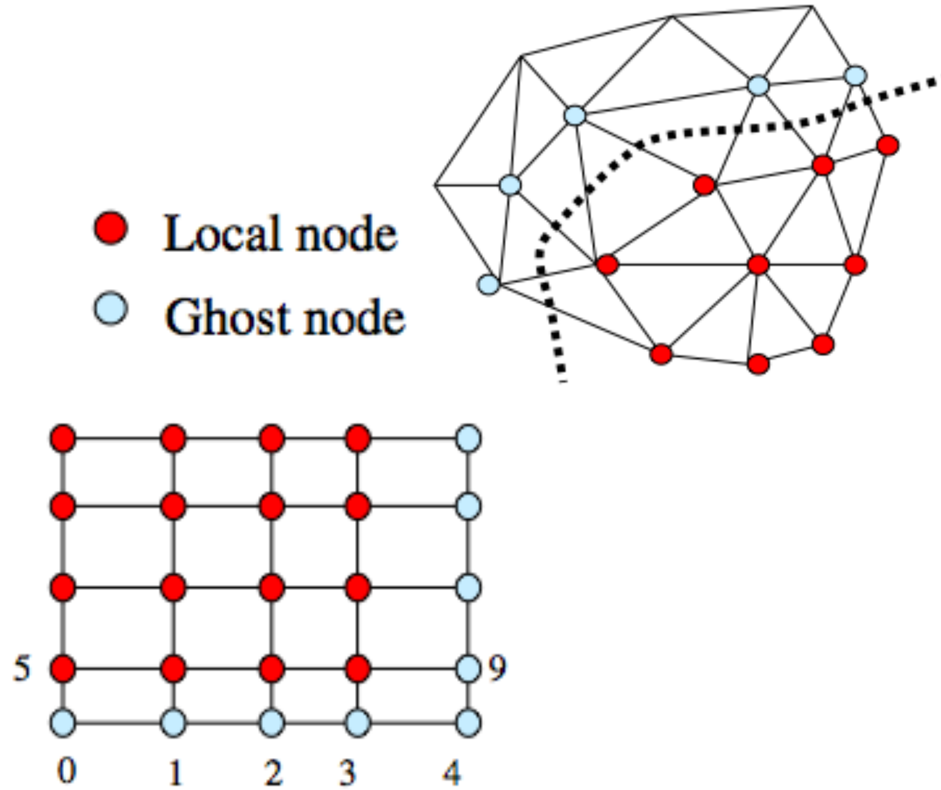


Image source: PETSc project

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

- [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