High Performance Scientific Computing

Shivasubramanian Gopalakrishnan
sgopalak@iitb.ac.in

March 11, 2016

Libraries for Numerical Linear Algebra

- BLAS
- LAPACK
- ATLAS

Slides based on material from S. Boyd, Stanford

Numerical Linear Algebra

most memory usage and computation time is spent on numerical linear algebra, e.g.,

- Construction of linear algebra systems
- matrix–matrix products, matrix–vector products
- factoring, forward and backward substitution

How to write Numerical Linear Algebra based software

whenever possible, rely on existing, mature software libraries

- you can focus on the higher-level algorithm
- your code will be more portable, less buggy, and will run faster – sometimes much faster

NETLIB

http://www.netlib.org

- maintained by University of Tennessee, Oak Ridge National Laboratory, and colleagues worldwide
- most of the code is public domain or freely licensed
- much written in FORTRAN 77.

BLAS

written by people who had the foresight to understand the future benefits of a standard suite of kernel routines for linear algebra.
created and organized in three levels:

- Level 1, 1973-1977: $O(n)$ vector operations: addition, scaling, dot products, norms
- Level 2, 1984-1986: $O(n^2)$ matrix-vector operations: matrix–vector products, triangular matrix–vector solves, rank-1 and symmetric rank-2 updates
- Level 3, 1987-1990: $O(n^3)$ matrix-matrix operations: matrix-matrix products, triangular matrix solves, low-rank update

BLAS Operations

Level 1	addition/scaling	$\alpha x, \quad \alpha x + y$
	dot products, norms	$x^T y, \quad \|x\|_2, \quad \|x\|_1$
Level 2	matrix/vector products	$\alpha A x + \beta y, \quad \alpha A^T x + \beta y$
	rank 1 updates	$A + \alpha x y^T, \quad A + \alpha x x^T$
	rank 2 updates	$A + \alpha x y^T + \alpha y x^T$
	triangular solves	$\alpha T^{-1} x, \quad \alpha T^{-T} x$
Level 3	matrix/matrix products	$\alpha AB + \beta C, \quad \alpha AB^T + \beta C$
		$\alpha A^T B + \beta C, \quad \alpha A^T B^T + \beta C$
	rank-k updates	$\alpha AA^T + \beta C, \quad \alpha A^T A + \beta C$
	rank-$2k$ updates	$\alpha A^T B + \alpha B^T A + \beta C$
	triangular solves	$\alpha T^{-1} C, \quad \alpha T^{-T} C$

Level 1: BLAS naming convention

BLAS routines have a Fortran-inspired naming convention:

`cblas_`	`X`	`XXXX`
prefix	data type	operation

data types:

`s`	single precision real	`d`	double precision real
`c`	single precision complex	`z`	double precision complex

operations:

`axpy`	$y \leftarrow \alpha x + y$	`dot`	$r \leftarrow x^T y$
`nrm2`	$r \leftarrow \|x\|_2 = \sqrt{x^T x}$	`asum`	$r \leftarrow \|x\|_1 = \sum_i \|x_i\|$

example:

`cblas_ddot` double precision real dot product

Level 2/3: BLAS naming convention

`cblas_`	`X`	`XX`	`XXX`
prefix	data type	structure	operation

matrix structure:

`tr`	triangular	`tp`	packed triangular	`tb`	banded triangular
`sy`	symmetric	`sp`	packed symmetric	`sb`	banded symmetric
`hy`	Hermitian	`hp`	packed Hermitian	`hn`	banded Hermitian
`ge`	general			`gb`	banded general

operations:

`mv`	$y \leftarrow \alpha Ax + \beta y$	`sv`	$x \leftarrow A^{-1}x$ (triangular only)
`r`	$A \leftarrow A + xx^T$	`r2`	$A \leftarrow A + xy^T + yx^T$
`mm`	$C \leftarrow \alpha AB + \beta C$	`r2k`	$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$

examples:

`cblas_dtrmv`	double precision real triangular matrix-vector product
`cblas_dsyr2k`	double precision real symmetric rank-$2k$ update

Using BLAS Efficiently

always choose a higher-level BLAS routine over multiple calls to a lower-level BLAS routine

$$A \leftarrow A + \sum_{i=1}^{k} x_i y_i^T, \quad A \in \mathbf{R}^{m \times n},\ x_i \in \mathbf{R}^m,\ y_i \in \mathbf{R}^n$$

two choices: k separate calls to the Level 2 routine `cblas_dger`

$$A \leftarrow A + x_1 y_1^T, \quad \ldots \quad A \leftarrow A + x_k y_k^T$$

or a single call to the Level 3 routine `cblas_dgemm`

$$A \leftarrow A + XY^T, \quad X = [x_1 \cdots x_k], \quad Y = [y_1 \cdots y_k]$$

the Level 3 choice will perform much better

BLAS Operations

why use BLAS when writing your own routines is so easy?

$$A \leftarrow A + XY^T, \qquad A \in \mathbf{R}^{m \times n},\ X \in \mathbf{R}^{m \times p},\ Y \in \mathbf{R}^{n \times p}$$

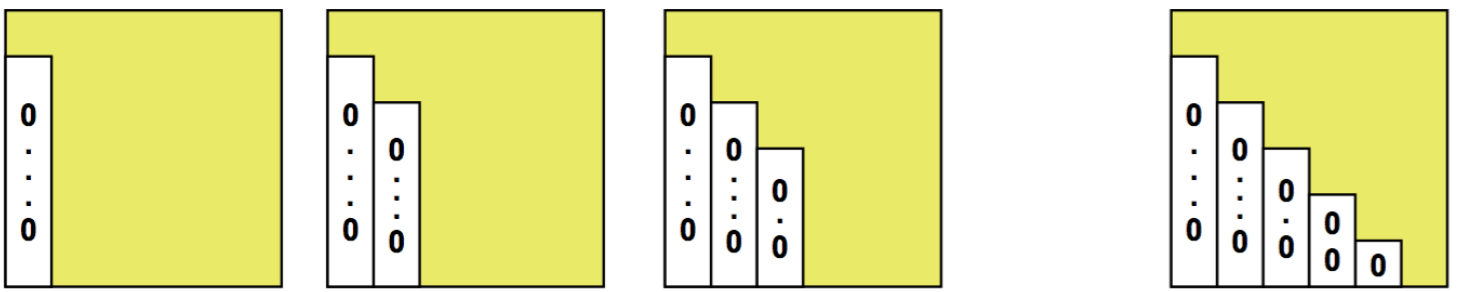
$$A_{ij} \leftarrow A_{ij} + \sum_{k=1}^{p} X_{ik} Y_{jk}$$

```
void matmultadd( int m, int n, int p, double* A,
                 const double* X, const double* Y ) {
  int i, j, k;
  for ( i = 0 ; i < m ; ++i )
    for ( j = 0 ; j < n ; ++j )
      for ( k = 0 ; k < p ; ++k )
        A[ i + j * n ] += X[ i + k * p ] * Y[ j + k * p ];
}
```

Gaussian Elimination (GE) for solving Ax=b

- Add multiples of each row to later rows to make A upper triangular
- Solve resulting triangular system Ux = c by substitution

```
    for each column i
    zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
        for each row j below row i
    for j = i+1 to n
            add a multiple of row i to row j
        tmp = A(j,i);
        for k = i to n
            A(j,k) = A(j,k) - (tmp/A(i,i)) * A(i,k)
```



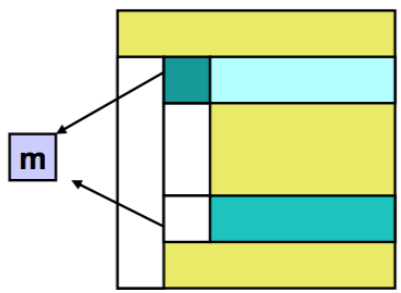
Refine GE Algorithm (1)

- **Initial Version**

```
    for each column i
    zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
        for each row j below row i
    for j = i+1 to n
            add a multiple of row i to row j
        tmp = A(j,i);
        for k = i to n
            A(j,k) = A(j,k) - (tmp/A(i,i)) * A(i,k)
```

- **Remove computation of constant tmp/A(i,i) from inner loop.**

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i to n
            A(j,k) = A(j,k) - m * A(i,k)
```



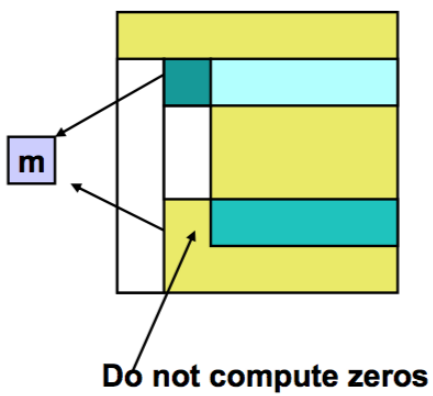
Refine GE Algorithm (2)

- **Last version**

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i to n
            A(j,k) = A(j,k) - m * A(i,k)
```

- **Don't compute what we already know: zeros below diagonal in column i**

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - m * A(i,k)
```



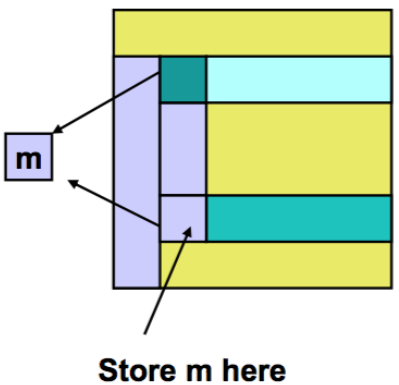
Refine GE Algorithm (3)

- **Last version**

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - m * A(i,k)
```

- **Store multipliers m below diagonal in zeroed entries for later use**

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```



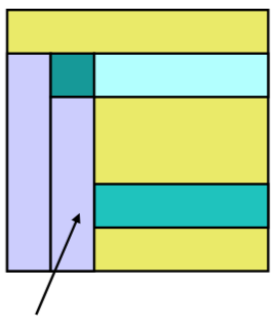
Refine GE Algorithm (4)

- **Last version**

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

- **Split Loop**

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```



Store all m's here before updating

Refine GE Algorithm (5)

- **Last version**

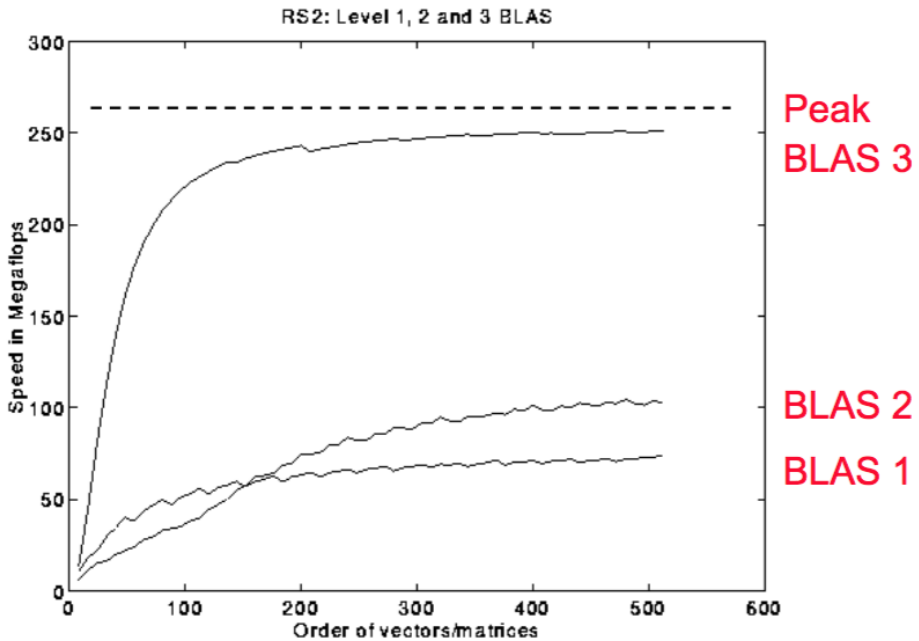
```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

- **Express using matrix operations (BLAS)**

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) * ( 1 / A(i,i) )
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )
            - A(i+1:n , i) * A(i , i+1:n)
```

Gauss Elimination with BLAS

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)                BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )              BLAS 2 (rank-1 update)
          - A(i+1:n , i) * A(i , i+1:n)
```



Improving performance through blocking

blocking is used to improve the performance of matrix/vector and matrix/matrix multiplications, Cholesky factorizations, etc.

$$A + XY^T \leftarrow \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} + \begin{bmatrix} X_{11} \\ X_{21} \end{bmatrix} + \begin{bmatrix} Y_{11}^T & Y_{21}^T \end{bmatrix}$$

$$\begin{aligned} A_{11} &\leftarrow A_{11} + X_{11}Y_{11}^T, & A_{12} &\leftarrow A_{12} + X_{11}Y_{21}^T, \\ A_{21} &\leftarrow A_{21} + X_{21}Y_{11}^T, & A_{22} &\leftarrow A_{22} + X_{21}Y_{21}^T \end{aligned}$$

optimal block size, and order of computations, depends on details of processor architecture, cache, memory

LAPACK computational routines

- factorizations: *LU, LLT /LLH, LDLT /LDLH, QR, LQ, QRZ, generalized QR and RQ*
- symmetric/Hermitian and nonsymmetric eigenvalue decompositions
- singular value decompositions
- generalized eigenvalue and singular value decompositions

Linear Algebra PACKage (LAPACK)

LAPACK contains subroutines for solving linear systems and performing common matrix decompositions and factorizations

- first release: February 1992; latest version (3.0): May 2000
- supercedes predecessors EISPACK and LINPACK
- supports same data types (single/double precision, real/complex) and matrix structure types (symmetric, banded, . . .) as BLAS
- uses BLAS for internal computations
- routines divided into three categories: auxiliary routines, computational routines, and driver routines

LAPACK driver routines

driver routines call a sequence of computational routines to solve standard linear algebra problems, such as

- linear equations: $AX = B$
- linear least squares: minimize$_x$ $\|b - Ax\|_2$
- linear least-norm:

$$\begin{array}{ll} \text{minimize}_y & \|y\|_2 \\ \text{subject to} & d = By \end{array}$$

- generalized linear least squares problems:

$$\begin{array}{ll} \text{minimize}_x & \|c - Ax\|_2 \\ \text{subject to} & Bx = d \end{array} \qquad \begin{array}{ll} \text{minimize}_y & \|y\|_2 \\ \text{subject to} & d = Ax + By \end{array}$$

Intel MKL©

- Optimized for Intel© chips: Pentium, Core (e.g., i7 and Xeon), and Itanium CPUs, multi-threaded
- Benchmark for high performance computing on Intel architecture
- Come with Intel compiler suite
- Platforms: Linux (free), Mac OS (paid) and Windows (paid)
- BLAS, Sparse BLAS, LAPACK, ScaLAPACK, VML, VSL, FFT, PDE, nonlinear optimization, ...

Availability of BLAS and LAPACK on GPU's

- NVIDIA© CUDA© math libraries
  - Optimized for NVIDIA GPUs
  - cuBLAS, cuSPARSE, cuRAND, cuFFT, CUDA Math Library, Thrust (data structures and algorithms)
  - Platforms: Linux (free), MacOS (free) and Windows (free)
- AMD© APPML (Accelerated Parallel Processing Math Library)
  - For AMD GPUs
  - BLAS, FFT
  - Platforms: Linux (free) and Windows (free)
- Third-party libraries
  - CULA (paid): CUDA LAPACK
  - MAGMA (free): OpenCL LAPACK

PETSc ("S" is silent)

Portable Extendable Toolkit for Scientific Computations

- Scientific Computations: parallel linear algebra, in particular linear and nonlinear solvers
- Toolkit: Contains high level solvers, but also the low level tools to roll your own.
- Portable: Available on many platforms, basically anything that has MPI

Why use it? Its big, powerful, well supported.

Slide based on material from V. Eijkhout, TACC

What does PETSc target?

- Serial and Parallel
- Linear and nonlinear
- Finite difference and finite element
- Structured and unstructured

Slide based on material from V. Eijkhout, TACC

What is in PETSc?

- Linear system solvers (sparse/dense, iterative/direct)
- Nonlinear system solvers
- Tools for distributed matrices
- Support for profiling, debugging, graphical output

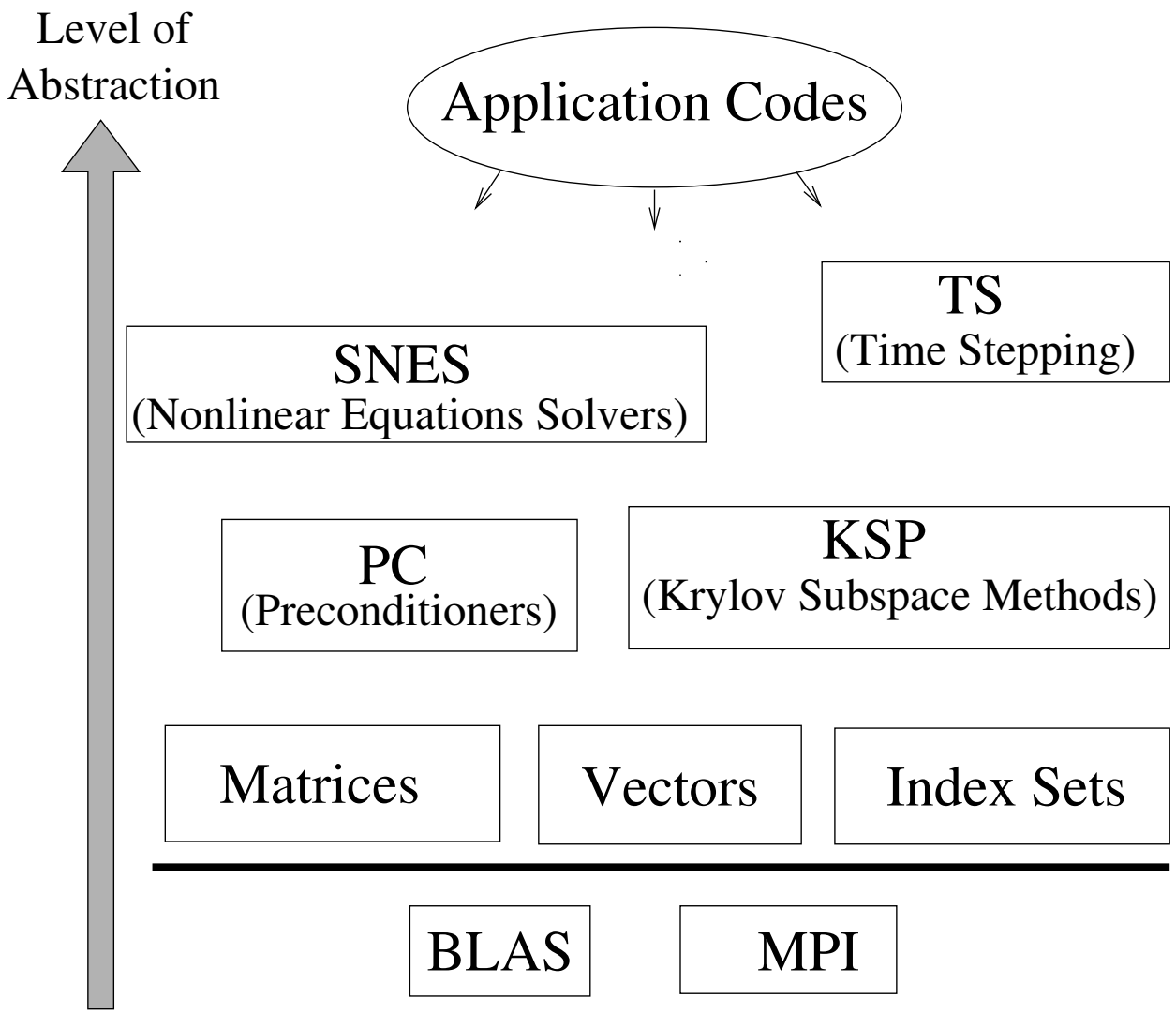
Slide based on material from V. Eijkhout, TACC

PETSc documentation

- Web page: http://tinyurl.com/PETSc-man-page
- PDF manual: http://tinyurl.com/PETSc-pdf-manual
- General questions about PETSc: petsc-maint@mcs.anl.gov

Slide based on material from V. Eijkhout, TACC

PETSc Structure



PETSc: Parallel numerical components

Nonlinear Solvers		
Newton–based Methods		Other
Line Search	Trust Region	

Time Steppers			
Euler	Backward Euler	Pseudo–Time Stepping	Other

Krylov Subspace Methods							
GMRES	CG	CGS	Bi–CG–Stab	TFQMR	Richardson	Chebychev	Other

Preconditioners						
Additive Schwarz	Block Jacobi	Jacobi	ILU	ICC	LU (sequential only)	Other

Matrices				
Compressed Sparse Row (AIJ)	Block Compressed Sparse Row (BAIJ)	Block Diagonal (BDiag)	Dense	Other

Vectors

Index Sets			
Indices	Block Indices	Stride	Other

PETSc: External packages

PETSc does not do everything, but it interfaces to other software:

- Dense linear algebra: Scalapack, Plapack
- Grid partitioning software: ParMetis, Jostle, Chaco, Party
- ODE solvers: PVODE
- Eigenvalue solvers (including SVD): SLEPc
- Optimization: TAO

Slide based on material from V. Eijkhout, TACC

PETSc and Parallelism

PETSc is layered on top of MPI MPI has basic tools: send elementary datatypes between processors PETSc has intermediate tools: insert matrix element in arbitrary location, do parallel matrix-vector product $\rightarrow$ you do not need to know much MPI when you use PETSc

Slide based on material from V. Eijkhout, TACC

PETSc and Parallelism

- All objects in Petsc are defined on a communicator; can only interact if on the same communicator
- Parallelism through MPI
- No OpenMP used; user can use shared memory programming
- Transparent: same code works sequential and parallel

Slide based on material from V. Eijkhout, TACC

PETSc: Object Oriented Design

Petsc uses objects: vector, matrix, linear solver, nonlinear solver

Overloading:

MATMult(A,x,y); // y $\leftarrow$ A x

same for sequential, parallel, dense, sparse

Slide based on material from V. Eijkhout, TACC