AMS 250: An Introduction to High Performance Computing

Parallel Math Libraries

Shawfeng Dong

shaw@ucsc.edu

(831) 502-7743

Applied Mathematics & Statistics

University of California, Santa Cruz

Outline

- Dense Linear Algebra
- Spare Linear Algebra
- FFT
- Intel MKL
- Cray Scientific and Math Library
- Frameworks
  - PETSc
  - Trilinos

What is Dense Linear Algebra?

- Not just matmul (Matrix Multiplication)!
- Linear Systems: $Ax = b$
- Least Squares: choose x to minimize $||Ax - b||_2$
  - Overdetermined or underdetermined; Unconstrained, constrained, or weighted
- Eigenvalues and Eigenvectors of Symmetric Matrices
  - Standard ($Ax = \lambda x$) or Generalized ($Ax = \lambda Bx$)
- Eigenvalues and Eigenvectors of Unsymmetric Matrices
- Singular Value Decomposition (SVD): $M = U\Sigma V^*$
- Different matrix structures
  - Real, complex; Symmetric, Hermitian, positive definite; dense, triangular, banded …
  - 27 types in LAPACK (and growing…)
- Level of detail
  - Simple Driver ("$x = A \setminus b$")
  - Expert Drivers with error bounds, extra-precision, other options
  - Lower level routines ("apply certain kind of orthogonal transformation", matmul, …)

UCB CS267:
http://www.cs.berkeley.edu/~demmel/cs267_Spr16/

A Brief History of Dense Linear Algebra Software (1)

- In the beginning there were do-loops...
  - libraries like **EISPACK** (for eigenvalue problems)
- Then **BLAS** (1) were invented (1973 - 1977)
  - Standard library of 15 operations mostly on *vectors*
    - "AXPY" ($y = \alpha \cdot x + y$), dot product, scale ($x = \alpha \cdot x$), etc.
    - Up to 4 versions of each (S/D/C/Z), 46 routines, 3300 lines of code (LOC)
  - Goals:
    - Common "pattern" to ease programming, readability
    - Robustness, via careful coding (avoiding over/underflow)
    - Portability + Efficiency via machine specific implementations
  - *O(n)* ops on *O(n)* data
  - Used in libraries like **LINPACK** (for linear systems)

A Brief History of Dense Linear Algebra Software (2)

- But BLAS-1 weren't enough
  - Consider AXPY ($y = \alpha \cdot x + y$): *2n* flops on *3n* read/writes
  - Computational intensity = (2n)/(3n) = 2/3, too low to run near peak speed (read/write dominates)!
  - Hard to vectorize on supercomputers of the day (1980s)
- So BLAS-2 were invented (1984-1986)
  - Standard library of 25 operations mostly on *matrix/vector* pairs
    - "GEMV": $y = \alpha \cdot A \cdot x + \beta \cdot x$, "GER": $A = A + \alpha \cdot x \cdot y^T$, $x = T^{-1} \cdot x$, etc.
    - Up to 4 versions of each (S/D/C/Z), 66 routines, 18K LOC
  - $O(n^2)$ ops on $O(n^2)$ data
  - Computational intensity still just ~ $(2n^2)/(n^2) = 2$
    - OK for vector machines, but not for machine with caches

A Brief History of Dense Linear Algebra Software (3)

- The next step: BLAS-3 (1987-1988)
  - Standard library of 9 operations mostly on matrix/matrix pairs
    - "GEMM": $C = \alpha \cdot A \cdot B + \beta \cdot C$, $C = \alpha \cdot A \cdot A^T + \beta \cdot C$, $B = T^{-1} \cdot B$
    - Up to 4 versions of each (S/D/C/Z), 30 routines, 10K LOC
  - $O(n^3)$ ops on $O(n^2)$ data
  - Computational intensity ~ $(2n^3)/(4n^2) = n/2$
    - Good for machines with caches, other memory hierarchy levels
- How much BLAS code so far
  - Reference implementation at http://www.netlib.org/blas/:
    - Latest version: 3.7.0, released in December 2016
    - 142 routines, 31K LOC for source & 28K LOC for testing
    - Unoptimized! E.g., 3 nested loops for GEMM
  - Lots more code for optimized implementations

A Brief History of Dense Linear Algebra Software (4)

- **LAPACK** (**L**inear **A**lgebra **PACK**age) (1989 – now)
  - http://www.netlib.org/lapack/
  - Successor to EISPACK and LINPACK
  - Contents of LAPACK
    - Algorithms that are (nearly) 100% BLAS-3
      Linear Systems: solve $Ax = b$ for x
      Least Squares: choose x to minimize $||Ax - b||_2$
    - Algorithms that are ~50% BLAS-3
      Eigenproblems: Find λ and x where $Ax = \lambda x$
      Singular Value Decomposition (SVD)
    - Generalized problems (e.g., $Ax = \lambda Bx$)
    - Error bounds for everything
    - Lots of variants depending on A's structure (banded, symmetric, etc.)
  - Latest version, 3.7.0, released in Dec. 2016
    - Source: 1750 routines, 721K LOC; Testing: 1094 routines, 472K LOC

A Brief History of Dense Linear Algebra Software (5)

- Is LAPACK parallel?
  - Only if the BLAS are parallel (possible in shared memory)
- **ScaLAPACK** (**Sca**lable **LAPACK**) (1995 – now)
  - http://www.netlib.org/scalapack/
  - For distributed memory – uses MPI
  - More complex data structures & algorithms than LAPACK
- Other active projects:
  - MAGMA (Matrix Algebra on GPU and Multicore Architectures): http://icl.eecs.utk.edu/magma/
  - PLASMA (Parallel Linear Algebra Software for Multicore Architectures): http://icl.eecs.utk.edu/plasma/

BLAS

- BLAS = Basic Linear Algebra Subprograms

  Level 1: vector-vector operations that are linear ($O(n)$) in data and linear ($O(n)$) in work, e.g., AXPY

  Level 2: matrix-vector operations that are quadratic ($O(n^2)$) in data and quadratic ($O(n^2)$) in work, e.g., GEMV

  Level 3: operations that are quadratic ($O(n^2)$) in data and cubic ($O(n^3)$) in work, e.g., GEMM

- The first letter of the subprogram name indicates the precision used:

  S: Real single precision, e.g., SGEMM

  D: Real double precision, e.g., DGEMM

  C: Complex single precision, e.g., CGEMM

  Z: Complex double precision, e.g., ZGEMM

Quick Reference Guide to the BLAS:
http://www.netlib.org/blas/blasqr.pdf

Level 1 BLAS

dim	scalar	vector	vector	scalars	5-element array		prefixes
SUBROUTINE xROTG (				A, B, C, S)		Generate plane rotation	S, D
SUBROUTINE xROTMG(				D1, D2, A, B,	PARAM)	Generate modified plane rotation	S, D
SUBROUTINE xROT (N,		X, INCX,	Y, INCY,	C, S)		Apply plane rotation	S, D
SUBROUTINE xROTM (N,		X, INCX,	Y, INCY,		PARAM)	Apply modified plane rotation	S, D
SUBROUTINE xSWAP (N,		X, INCX,	Y, INCY)			$x \leftrightarrow y$	S, D, C, Z
SUBROUTINE xSCAL (N,	ALPHA,	X, INCX)				$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
SUBROUTINE xCOPY (N,		X, INCX,	Y, INCY)			$y \leftarrow x$	S, D, C, Z
SUBROUTINE xAXPY (N,	ALPHA,	X, INCX,	Y, INCY)			$y \leftarrow \alpha x + y$	S, D, C, Z
FUNCTION xDOT (N,		X, INCX,	Y, INCY)			$dot \leftarrow x^T y$	S, D, DS
FUNCTION xDOTU (N,		X, INCX,	Y, INCY)			$dot \leftarrow x^T y$	C, Z
FUNCTION xDOTC (N,		X, INCX,	Y, INCY)			$dot \leftarrow x^H y$	C, Z
FUNCTION xxDOT (N,		X, INCX,	Y, INCY)			$dot \leftarrow \alpha + x^T y$	SDS
FUNCTION xNRM2 (N,		X, INCX)				$nrm2 \leftarrow \|x\|_2$	S, D, SC, DZ
FUNCTION xASUM (N,		X, INCX)				$asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$	S, D, SC, DZ
FUNCTION IxAMAX(N,		X, INCX)				$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level 2 BLAS

options		dim	b-width	scalar	matrix	vector	scalar	vector		
xGEMV (	TRANS,	M, N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xGBMV (	TRANS,	M, N,	KL, KU,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xHEMV (UPLO,		N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHBMV (UPLO,		N, K,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHPMV (UPLO,		N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xSYMV (UPLO,		N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSBMV (UPLO,		N, K,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSPMV (UPLO,		N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xTRMV (UPLO,	TRANS, DIAG,	N,			A, LDA,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTBMV (UPLO,	TRANS, DIAG,	N, K,			A, LDA,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTPMV (UPLO,	TRANS, DIAG,	N,			AP,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTRSV (UPLO,	TRANS, DIAG,	N,			A, LDA,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTBSV (UPLO,	TRANS, DIAG,	N, K,			A, LDA,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTPSV (UPLO,	TRANS, DIAG,	N,			AP,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z

options		dim	scalar	vector	vector	matrix			
xGER (		M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)		$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
xGERU (		M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)		$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
xGERC (		M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)		$A \leftarrow \alpha xy^H + A, A - m \times n$	C, Z
xHER (	UPLO,	N,	ALPHA,	X, INCX,		A, LDA)		$A \leftarrow \alpha xx^H + A$	C, Z
xHPR (	UPLO,	N,	ALPHA,	X, INCX,		AP)		$A \leftarrow \alpha xx^H + A$	C, Z
xHER2 (	UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)		$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xHPR2 (	UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	AP)		$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xSYR (	UPLO,	N,	ALPHA,	X, INCX,		A, LDA)		$A \leftarrow \alpha xx^T + A$	S, D
xSPR (	UPLO,	N,	ALPHA,	X, INCX,		AP)		$A \leftarrow \alpha xx^T + A$	S, D
xSYR2 (	UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)		$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D
xSPR2 (	UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	AP)		$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D

Level 3 BLAS

options			dim		scalar	matrix	matrix	scalar	matrix		
xGEMM (		TRANSA, TRANSB,	M, N,	K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha op(A)op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$	S, D, C, Z
xSYMM (	SIDE, UPLO,		M, N,		ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^T$	S, D, C, Z
xHEMM (	SIDE, UPLO,		M, N,		ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$	C, Z
xSYRK (	UPLO,	TRANS,	N,	K,	ALPHA,	A, LDA,		BETA,	C, LDC)	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$	S, D, C, Z
xHERK (	UPLO,	TRANS,	N,	K,	ALPHA,	A, LDA,		BETA,	C, LDC)	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$	C, Z
xSYR2K(	UPLO,	TRANS,	N,	K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB^T + \bar{\alpha} BA^T + \beta C, C \leftarrow \alpha A^T B + \bar{\alpha} B^T A + \beta C, C - n \times n$	S, D, C, Z
xHER2K(	UPLO,	TRANS,	N,	K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C, C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C, C - n \times n$	C, Z
xTRMM (	SIDE, UPLO, TRANSA,	DIAG,	M, N,		ALPHA,	A, LDA,	B, LDB)			$B \leftarrow \alpha op(A)B, B \leftarrow \alpha Bop(A), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z
xTRSM (	SIDE, UPLO, TRANSA,	DIAG,	M, N,		ALPHA,	A, LDA,	B, LDB)			$B \leftarrow \alpha op(A^{-1})B, B \leftarrow \alpha Bop(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z

BLAS Implementations

- CPU

  Netlib reference implementation

  ATLAS

  GotoBLAS / GotoBLAS2 / OpenBLAS

  AMD Core Math Library (ACML)

  Intel Math Kernel Library (MKL)

- GPU

  cuBLAS: *not* a drop-in replacement of standard BLAS; one must use the cuBLAS / cuBLAS-XT API

  http://docs.nvidia.com/cuda/cublas/index.html

  NVBLAS: *is* a drop-in replacement of standard BLAS; can accelerate most BLAS Level-3 routines

  http://docs.nvidia.com/cuda/nvblas/index.html

Netlib BLAS

- http://www.netlib.org/blas/
- A platform independent implementation of BLAS but *without any attempt at optimizing performance*; not suitable for production work!
- Written in Fortran 77

$$C = \alpha \cdot A \cdot B + \beta \cdot C$$

```
***** SGEMM *****
      DO 90 j = 1,n
              DO 60 i = 1,m
                  c(i,j) = beta*c(i,j)
60            CONTINUE

          DO 80 l = 1,k
              temp = alpha*b(l,j)
              DO 70 i = 1,m
                  c(i,j) = c(i,j) + temp*a(i,l)
70            CONTINUE
80        CONTINUE
90    CONTINUE
```

A Simple BLAS Example

```
      program blas3
      implicit none
      real*4 a(4,5), b(5,4), c(4,4)
      external sgemm

C     Note Fortran is column major
      data a/ 1,  6, 11, 16,
     *        2,  7, 12, 17,
     *        3,  8, 13, 18,
     *        4,  9, 14, 19,
     *        5, 10, 15, 20 /
      data b/1,0,0,0,0,
     *       0,0,1,0,0,
     *       0,1,0,0,0,
     *       0,0,0,1,0/

C                tfm  tfm  rowA colB K  alpha a lda  b  ldb beta c  ldc
      call sgemm('N', 'N', 4,   4,   5, 1.0,  a,  4, b, 5,  0.0, c, 4)

      end
```

```
subroutine sgemm(character TRANSA,
                 character TRANSB,
                 integer M,
                 integer N,
                 integer K,
                 real ALPHA,
                 real, dimension(lda,*) A,
                 integer LDA,
                 real, dimension(ldb,*) B,
                 integer LDB,
                 real BETA,
                 real, dimension(ldc,*) C,
                 integer LDC)
```

SGEMM

```
SGEMM performs one of the matrix-matrix operations
    C := alpha*op( A )*op( B ) + beta*C,
where  op( X ) is one of
    op( X ) = X   or   op( X ) = X**T,
alpha and beta are scalars, and A, B and C are matrices, with op( A )
an m by k matrix,  op( B ) a k by n matrix and C an m by n matrix.
```

```
TRANSA is CHARACTER*1
On entry, TRANSA specifies the form of op( A ) to be used in
the matrix multiplication as follows:
    TRANSA = 'N' or 'n',  op( A ) = A
    TRANSA = 'T' or 't',  op( A ) = A**T
    TRANSA = 'C' or 'c',  op( A ) = A**T
```

```
LDA is INTEGER
On entry, LDA specifies the first dimension of A as declared
in the calling (sub) program. When  TRANSA = 'N' or 'n' then
LDA must be at least  max( 1, m ), otherwise  LDA must be at
least  max( 1, k ).
```

BLAS Routines:
http://www.netlib.org/blas/#_blas_routines

Linking with Netlib BLAS

On Hyades, Netlib BLAS is installed at /pfs/sw/serial/gcc/lapack-3.5.0.

- To link with the Netlib BLAS library, using **gfortran**:

```
$ gfortran -o blas3.x blas3.f \
           -L/pfs/sw/serial/gcc/lapack-3.5.0/lib -lblas
```

- To link with the Netlib BLAS library, using the Intel Fortran Compiler:

```
$ ifort -o blas3.x blas3.f \
        -L/pfs/sw/serial/gcc/lapack-3.5.0/lib -lblas \
        -lgfortran
```

Calling Fortran BLAS from C Code

Fortran subroutines are the equivalent of C functions returning void. When compiling, most Fortran compilers append an underscore (_) to the subroutine name. For example:

```
$ nm /pfs/sw/serial/gcc/lapack-3.5.0/lib/libblas.a | grep sgemm
sgemm.o:
0000000000000000 T sgemm_
```

To call, e.g., the Fortran subroutine sgemm from C, first declare its prototype in the C code:

```
extern void sgemm_( char *, char *, int *, int *, int *, float *,
    float *, int *, float *, int *, float *, float *, int * );
```

To compile a C program and link with the Netlib Fortran BLAS library, use the following flags:

```
-L/pfs/sw/serial/gcc/lapack-3.5.0/lib -lblas -lgfortran
```

http://www.yolinux.com/TUTORIALS/LinuxTutorialMixingFortranAndC.html

Calling Fortran BLAS from C++ Code

To call, e.g., the Fortran subroutine sgemm from C++, first declare its prototype in the C++ code:

```
extern "C" void sgemm_( char *, char *, int *, int *, int *,
                        float *, float *, int *, float *, int *,
                        float *, float *, int * );
```

To compile a C++ program and link with the Netlib Fortran BLAS library, use the following flags:

```
-L/pfs/sw/serial/gcc/lapack-3.5.0/lib -lblas -lgfortran
```

A quick Note on iso_c_binding

- Fortran 2003 has defined a standard intrinsic module, iso_c_binding, for C interoperability
  http://fortranwiki.org/fortran/show/ISO_C_BINDING
- A good tutorial on how to use C from Fortran and vice versa, using the portable facilities defined in Fortran 2003
  http://people.ds.cam.ac.uk/nmm1/fortran/paper_14.pdf

Netlib CBLAS

- Netlib also provides a reference implementation of C interface to the BLAS
- Not to be confused with f2c'ed BLAS, the naming scheme of which is to take the Fortran BLAS routine name, make it lower case, and add the suffix _. For example, the Fortran routine DGEMM becomes dgemm_
- The CBLAS function names are of the form cblas_*, e.g., cblas_dgemm
- CBLAS header file:

```
#include <cblas.h>
```

- On Hyades, the Netlib CBLAS library (libcblas.a) is also installed at /pfs/sw/serial/gcc/lapack-3.5.0

LAPACK

- LAPACK (**L**inear **A**lgebra **PACK**age): http://www.netlib.org/lapack/
- Written in Fortran 90
- Provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems
- Dense and banded matrices are handled, but not general *sparse* matrices
- LAPACK routines are written so that as much as possible of the computation is performed by calls to the **BLAS**!
- The first letter of the subprogram name indicates the precision used:
  - S: Real single precision, e.g., SGESV
  - D: Real double precision, e.g., DGESV
  - C: Complex single precision, e.g., CGESV
  - Z: Complex double precision, e.g., ZGESV

LAPACK Users' Guide, 3rd Edition
http://www.netlib.org/lapack/lug

LAPACK Matrix Types

- BD – bidiagonal
- GB – general banded
- GE – general
- GG – general , pair
- GT – tridiagonal
- HB – Hermitian banded
- HE – Hermitian
- HG – upper Hessenberg, pair
- HP – Hermitian, packed
- HS – upper Hessenberg
- OR – (real) orthogonal
- OP – (real) orthogonal, packed
- PB – positive definite, banded
- PO – positive definite
- PP – positive definite, packed
- PT – positive definite, tridiagonal
- SB – symmetric, banded
- SP – symmetric, packed
- ST – symmetric, tridiagonal
- SY – symmetric
- TB – triangular, banded
- TG – triangular, pair
- TP – triangular, packed
- TR – triangular
- TZ – trapezoidal
- UN – unitary
- UP – unitary packed

LAPACK Driver Types

Two types of driver routines are provided for solving systems of linear equations:

1. a simple driver (name ending with **SV**), which solves the system $Ax = b$ by factorizing A and overwriting b with the solution x
2. an expert driver (name ending with **SVX**), which can also perform the following functions (some of them optionally):
   - solve $A^T x = b$ or $A^H x = b$ (unless A is symmetric or Hermitian);
   - estimate the condition number of A, check for near-singularity, and check for pivot growth;
   - refine the solution and compute forward and backward error bounds;
   - equilibrate the system if A is poorly scaled.

The expert driver requires roughly twice as much storage as the simple driver in order to perform these extra functions.

A Simple LAPACK Example

```
      PROGRAM SGESV_EX
      INTEGER    N, NRHS
      PARAMETER ( N = 5, NRHS = 3 )
      INTEGER    LDA, LDB
      PARAMETER ( LDA = N, LDB = N )
      INTEGER    INFO
      INTEGER    IPIV( N )
      REAL A( LDA, N ), B( LDB, NRHS )
      DATA A/ 6.80,-2.11, 5.66, 5.97, 8.23,
     $       -6.05,-3.30, 5.36,-4.44, 1.08,
     $       -0.45, 2.58,-2.70, 0.27, 9.04,
     $        8.32, 2.71, 4.35,-7.17, 2.14,
     $       -9.67,-5.14,-7.26, 6.08,-6.87 /
      DATA B/ 4.02, 6.19,-8.22,-7.57,-3.03,
     $       -1.56, 4.00,-8.67, 1.75, 2.86,
     $        9.81,-4.09,-4.57,-8.61, 8.99 /
      EXTERNAL SGESV
*     Solve the equations A*X = B
      CALL SGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
      END
```

```
subroutine sgesv ( integer N,
    integer NRHS,
    real, dimension( lda, * ) A,
    integer LDA,
    integer, dimension( * ) IPIV,
    real, dimension( ldb, * ) B,
    integer LDB,
    integer INFO
)
```

SGESV

```
SGESV computes the solution to a real system of linear equations
    A * X = B,
where A is an N-by-N matrix and X and B are N-by-NRHS matrices.
The LU decomposition with partial pivoting and row interchanges is
used to factor A as
    A = P * L * U,
where P is a permutation matrix, L is unit lower triangular, and U is
upper triangular.  The factored form of A is then used to solve the
system of equations A * X = B.
```

```
A is REAL array, dimension (LDA,N)
On entry, the N-by-N coefficient matrix A.
On exit, the factors L and U from the factorization
A = P*L*U; the unit diagonal elements of L are not stored.
```

```
B is REAL array, dimension (LDB,NRHS)
On entry, the N-by-NRHS matrix of right hand side matrix B.
On exit, if INFO = 0, the N-by-NRHS solution matrix X.is INTEGER
```

Explore LAPACK code:
http://www.netlib.org/lapack/explore-html/

Netlib BLAS & LAPACK

On Hyades, Netlib LAPACK 3.5.0, built with the Netlib BLAS implementation, is installed at /pfs/sw/serial/gcc/lapack-3.5.0.

- To compile a LAPACK program and link with Netlib BLAS & LAPACK, using **gfortran**:

```
$ module load lapack/s_gcc_netlib_3.5.0
$ gfortran -o sgesv_ex.x sgesv_ex.f -llapack -lblas
```

- If you prefer the Intel Fortran Compiler:

```
$ ifort -o sgesv_ex.x sgesv_ex.f -llapack -lblas \
        -lgfortran
```

LAPACKE

- **LAPACKE** is a C interface to LAPACK: http://www.netlib.org/lapack/lapacke.html
- Not to be confused with **CLAPACK**, which is f2c'ed from the Fortran LAPACK
- The LAPACKE interface is two-level:
  - The *high-level* interface handles all workspace memory allocation internally, while the *middle-level* interface requires the user to provide workspace arrays as in the original FORTRAN interface. Both interfaces provide support for both column-major and row-major matrices.
  - The naming scheme for the high-level interface is to take the Fortran LAPACK routine name, make it lower case, and add the prefix LAPACKE_. For example, the LAPACK subroutine DGETRF becomes LAPACKE_dgetrf.
  - The naming scheme for the middle-level interface is to take the Fortran LAPACK routine name, make it lower case, then add the prefix LAPACKE_ and the suffix _work. For example, the LAPACK subroutine DGETRF becomes LAPACKE_dgetrf_work.
- LAPACKE header file:

```
#include <lapacke.h>
```

Netlib LAPACKE

On Hyades, Netlib LAPACKE 3.5.0 is also installed at /pfs/sw/serial/gcc/lapack-3.5.0.

- To compile a LAPACKE program and link with Netlib libraries, using **gcc**:

```
$ module load lapack/s_gcc_netlib_3.5.0
$ gcc -o example_DGESV_rowmajor.x \
      example_DGESV_rowmajor.c lapacke_example_aux.c \
      -llapacke -llapack -lblas -lgfortran
```

- If you prefer the Intel Fortran Compiler:

```
$ icc -o example_DGESV_rowmajor.x \
      example_DGESV_rowmajor.c lapacke_example_aux.c \
      -llapacke -llapack -lblas -lgfortran
```

ATLAS

- ATLAS (Automatically Tuned Linear Algebra Software):
  http://math-atlas.sourceforge.net/
- ATLAS paper: http://www.netlib.org/lapack/lawnspdf/lawn131.pdf
- An open source efficient and full implementation of **BLAS** APIs for C and Fortran 77, and a few routines from **LAPACK**
- Large improvement over the Netlib BLAS implementation
- But performance often trails that of specialized libraries written for one specific hardware platform, e.g., Intel MKL

ATLAS on Hyades

On Hyades, ATLAS 3.10.2 is installed at /pfs/sw/serial/gcc/atlas-3.10.2. It includes the following libraries:

liblapack.a:	The serial LAPACK routines provided by ATLAS, including both Fortran LAPACK and ATLAS's clapack (which is different from either LAPACKE or Netlib CLAPACK).
libcblas.a:	The ANSI C interface to the BLAS.
libf77blas.a:	The Fortran77 interface to the BLAS.
libptlapack.a:	The threaded LAPACK routines provided by ATLAS.
libptcblas.a:	The ANSI C interface to the threaded BLAS.
libptf77blas.a:	The Fortran77 interface to the threaded BLAS.
libatlas.a:	The main ATLAS library, providing low-level routines for all interface libs.

Linking with ATLAS

- On Hyades, to compile a Fortran program and link with the serial BLAS library provided by ATLAS, using **gfortran**:

```
$ module load lapack/s_gcc_ATLAS_3.10.2
$ gfortran -o blaspgm.x blaspgm.f -lf77blas –latlas
```

- To compile a C program and link with the serial CBLAS library provided by ATLAS, using **gcc**:

```
$ gcc -o cblaspgm.x cblaspgm.c -lcblas –latlas
```

- To compile a Fortran program and link with the serial LAPACK library provided by ATLAS, using **gfortran**:

```
$ gfortran -o lapackpgm.x lapackpgm.f \
           -llapack -lf77blas -lcblas –latlas
```

GotoBLAS

- Originally written by Kazushige Goto (後藤和茂) at TACC in 2002
  - immediately boost the performance of a supercomputer based on Pentium 4 from 1.5 TFLOPS to 2 TFLOPS!
- GotoBLAS paper:
  https://www.cs.utexas.edu/users/pingali/CS378/2008sp/papers/gotoPaper.pdf
- GEMM is highly tuned for the x86 and AMD86 processor architecture by means of *handcrafted assembly code*, using
  - L2 cache
  - GEBP (General block-times-panel multiply) kernel
- Development ceased
  - final version touting optimal performance on Intel's Nehalem architecture (contemporary in 2008)

OpenBLAS

- http://www.openblas.net/
- OpenBLAS is an optimized BLAS library based on GotoBLAS2
- Adds optimized implementation for several processor architecture, including Intel Sandy Bridge and Loongson
- Claims to achieve performance comparable to Intel MKL

A quick Aside: Why is NumPy slow?

```
$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
>>> import numpy as np
>>> np.show_config()
blas_info:
    libraries = ['blas', 'blas']
    library_dirs = ['/usr/lib']
    define_macros = [('HAVE_CBLAS', None)]
    language = c
lapack_info:
    libraries = ['lapack', 'lapack']
    library_dirs = ['/usr/lib']
    language = f77
mkl_info:
  NOT AVAILABLE
lapack_mkl_info:
  NOT AVAILABLE
atlas_info:
  NOT AVAILABLE
openblas_info:
  NOT AVAILABLE
```

Ubuntu 16.04

Netlib reference BLAS

Netlib reference LAPACK

Anaconda distribution of Python

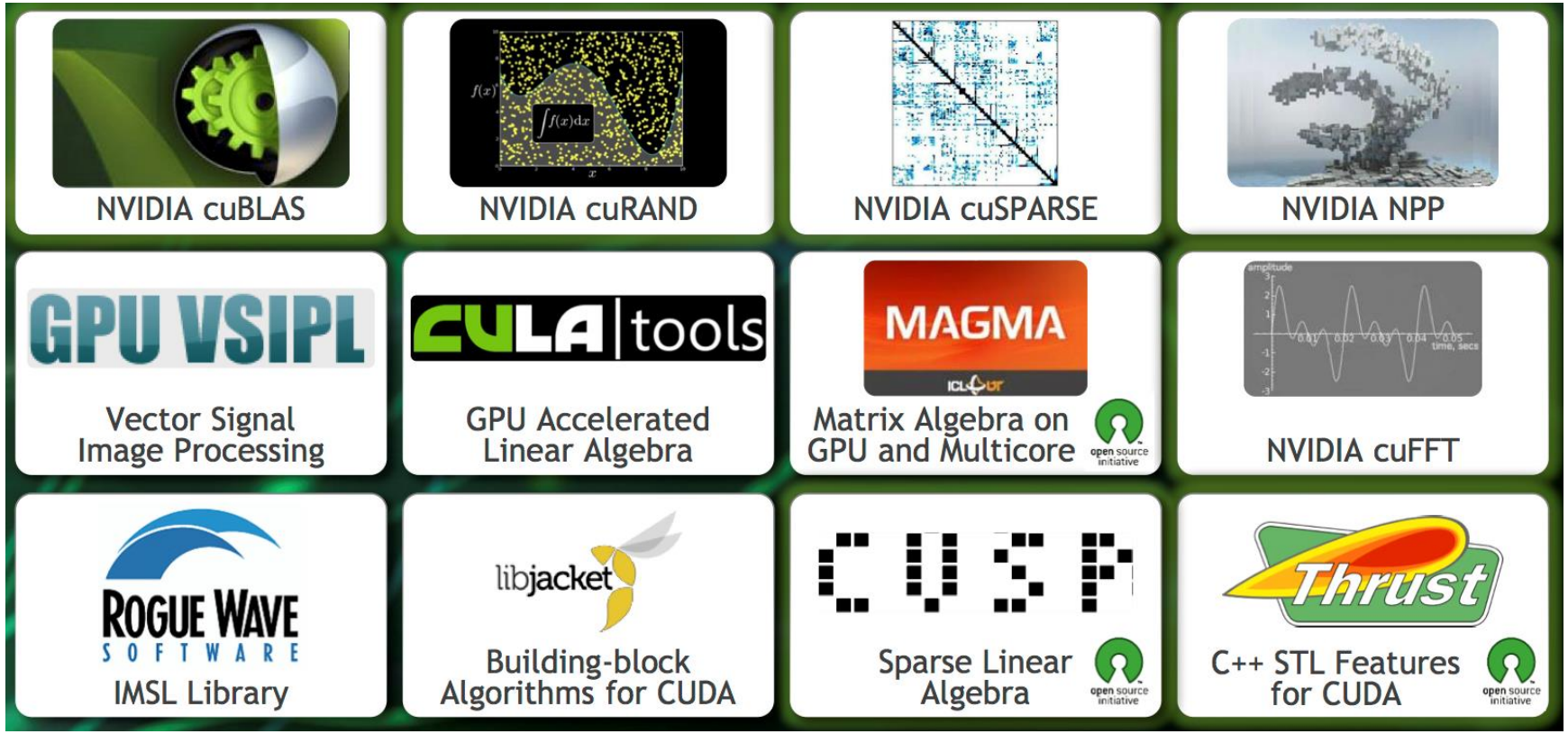
Anaconda: https://www.continuum.io/downloads

```
(C:\Python\Anaconda3) C:\Users\dong>python
Python 3.6.0 |Anaconda 4.3.1 (64-bit)| (default, Dec 23 2016, 11:57:41) [MSC
v.1900 64 bit (AMD64)] on win32
>>> import numpy as np
>>> np.show_config()
blas_mkl_info:
    libraries = ['mkl_core_dll', 'mkl_intel_lp64_dll', 'mkl_intel_thread_dll']
    library_dirs = ['C:/Python/Anaconda3\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:/Python/Anaconda3\\Library\\include']
lapack_mkl_info:
    libraries = ['mkl_core_dll', 'mkl_intel_lp64_dll', 'mkl_intel_thread_dll']
    library_dirs = ['C:/Python/Anaconda3\\Library\\lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['C:/Python/Anaconda3\\Library\\include']
```

Another high-performance Python distribution:

Intel Distribution for Python

GPU-Accelerated Libraries

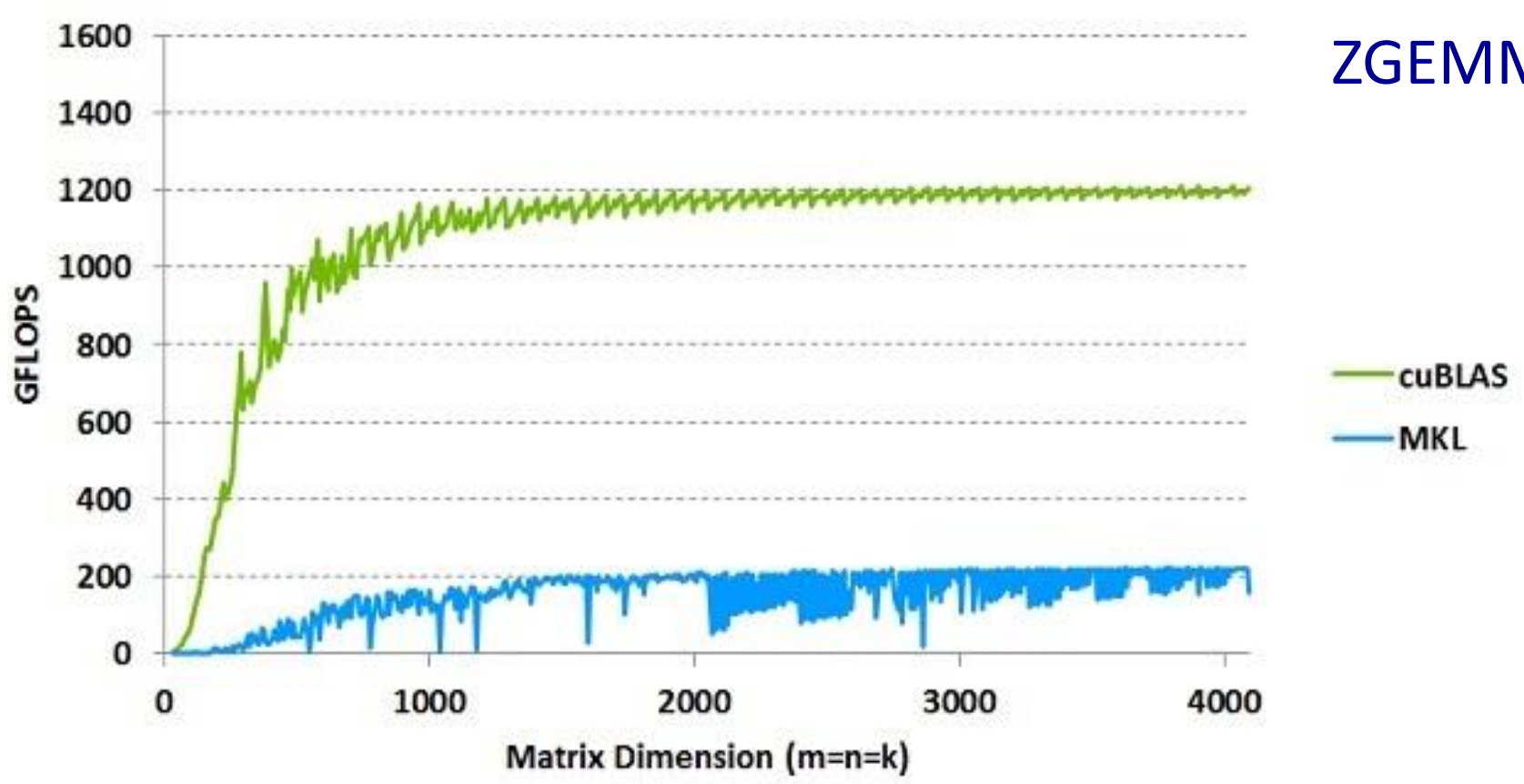


https://developer.nvidia.com/gpu-accelerated-libraries

cuBLAS

- The cuBLAS library is an implementation of BLAS on top of the CUDA runtime: http://docs.nvidia.com/cuda/cublas/index.html
- The cuBLAS library exposes 2 sets of API: cuBLAS & CUBLASXT
- To use the cuBLAS API, the application must allocate the required matrices and vectors in the GPU memory space, fill them with data, call the sequence of desired cuBLAS functions, and then upload the results from the GPU memory space back to the host
- To use the CUBLASXT API, the application must keep the data on the Host and the Library will take care of dispatching the operation to one or multiple GPUS present in the system, depending on the user request

cuBLAS vs MKL



- cuBLAS 6.5 on K40m, ECC ON, input and output data on device
- MKL 11.0.4 on Intel IvyBridge 12-core E5-2697 v2 @2.7GHZ

https://developer.nvidia.com/cublas

GEMM

- GEMM = **Ge**neral **M**atrix-**M**atrix multiplication
- C = α * op(A) * op(B) + β * C
- BLAS level 3

Fortran BLAS:

```
SUBROUTINE SGEMM ( TRANSA, TRANSB, M, N, K, ALPHA,
                   A, LDA, B, LDB, BETA, C, LDC)
    CHARACTER * TRANSA, TRANSB
    INTEGER     M, N, K, LDA, LDB, LDC
    REAL        ALPHA, BETA
    REAL        A(LDA, *), B(LDB, *), C(LDC, *)
```

cuBLAS:

```
cublasStatus_t cublasSgemm(cublasHandle_t handle,
                           cublasOperation_t transa,
                           cublasOperation_t transb,
                           int m, int n, int k,
                           const float *alpha,
                           const float *A, int lda,
                           const float *B, int ldb,
                           const float *beta,
                           float *C, int ldc)
```

Matrix Multiplication: using cuBLAS

```
#include <cuda_runtime.h>
#include <cublas_v2.h>

    const float alpha = 1.0f;
    const float beta  = 0.0f;
    // allocate arrays on GPU (skipped)
    // create the handle
    cublasHandle_t handle;
    cublasCreate(&handle);

    cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N,
                m, n, k, &alpha,
                d_B, WB, d_A, WA,
                &beta, d_C, WA);

    // destroy the handle
    cublasDestroy(handle);
```

```
cublasStatus_t cublasSgemm(
                    cublasHandle_t handle,
                    cublasOperation_t transa,
                    cublasOperation_t transb,
                    int m, int n, int k,
                    const float *alpha,
                    const float *A, int lda,
                    const float *B, int ldb,
                    const float *beta,
                    float *C, int ldc)
```

Typos?

No! Why?

Linking with cuBLAS

```
$ g++ matrixMulCUBLAS.cpp -l cudart -lcublas \
      -o matrixMulCUBLAS
```

For complete code examples, see

- CUDA Code Samples: 0_Simple/matrixMulCUBLAS/
- https://solarianprogrammer.com/2012/05/31/matrix-multiplication-cuda-cublas-curand-thrust/
- Matrix computations on the GPU, cuBLAS and MAGNA by example: https://developer.nvidia.com/sites/default/files/akamai/cuda/files/Misc/mygpu.pdf

PBLAS

- PBLAS (Parallel Basic Linear Algebra Subprograms):
  http://www.netlib.org/scalapack/pblas_qref.html
- An implementation of Level 2 and 3 BLAS intended for *distributed memory* architectures
- PBLAS depends on
  - Level 1 sequential BLAS operations for local computation
  - BLACS (Basic Linear Algebra Communication Subprograms) for communication between nodes, using, e.g., MPI
- Matrices are laid out in a two-dimensional block cyclic decomposition
- PBLAS example: http://www.netlib.org/scalapack/examples/pblas.tgz

Block Cyclic Data Distribution

The table lists the data on each process. A = global array, B = local array, array indices start at 1.

Process *(coordinates)*	Array Values
0 *(0,0)*	B[1,1]=A[1,1] B[1,2]=A[1,2] B[2,1]=A[2,1] B[2,2]=A[2,2] B[1,3]=A[1,7] B[1,4]=A[1,8] B[2,3]=A[2,7] B[2,4]=A[2,8] B[3,1]=A[5,1] B[3,2]=A[5,2] B[4,1]=A[6,1] B[4,2]=A[6,2] B[3,3]=A[5,7] B[3,4]=A[5,8] B[4,3]=A[6,7] B[4,4]=A[6,8] B[5,1]=A[9,1] B[5,2]=A[9,2] B[5,3]=A[9,7] B[5,4]=A[9,8]
1 *(0,1)*	B[1,1]=A[1,3] B[1,2]=A[1,4] B[2,1]=A[2,3] B[2,2]=A[2,4] B[1,3]=A[1,9] B[2,3]=A[2,9] B[3,1]=A[5,3] B[3,2]=A[5,4] B[4,1]=A[6,3] B[4,2]=A[6,4] B[3,3]=A[5,9] B[4,3]=A[6,9] B[5,1]=A[9,3] B[5,2]=A[9,4] B[5,3]=A[9,9]
2 *(0,2)*	B[1,1]=A[1,5] B[1,2]=A[1,6] B[2,1]=A[2,5] B[2,2]=A[2,6] B[3,1]=A[5,5] B[3,2]=A[5,6] B[4,1]=A[6,5] B[4,2]=A[6,6] B[5,1]=A[9,5] B[5,2]=A[9,6]
3 *(1,0)*	B[1,1]=A[3,1] B[1,2]=A[3,2] B[2,1]=A[4,1] B[2,2]=A[4,2] B[1,3]=A[3,7] B[1,4]=A[3,8] B[2,3]=A[4,7] B[2,4]=A[4,8] B[3,1]=A[7,1] B[3,2]=A[7,2] B[4,1]=A[8,1] B[4,2]=A[8,2] B[3,3]=A[7,7] B[3,4]=A[7,8] B[4,3]=A[8,7] B[4,4]=A[8,8]
4 *(1,1)*	B[1,1]=A[3,3] B[1,2]=A[3,4] B[2,1]=A[4,3] B[2,2]=A[4,4] B[1,3]=A[3,9] B[2,3]=A[4,9] B[3,1]=A[7,3] B[3,2]=A[7,4] B[4,1]=A[8,3] B[4,2]=A[8,4] B[3,3]=A[7,9] B[4,3]=A[8,9]
5 *(1,2)*	B[1,1]=A[3,5] B[1,2]=A[3,6] B[2,1]=A[4,5] B[2,2]=A[4,6] B[3,1]=A[7,5] B[3,2]=A[7,6] B[4,1]=A[8,5] B[4,2]=A[8,6]

The color scheme shows the distribution of the global array on the computational grid. For the sake of resolution, colors are only displayed for less than 16 processes.

0	0	1	1	2	2	0	0	1
0	0	1	1	2	2	0	0	1
3	3	4	4	5	5	3	3	4
3	3	4	4	5	5	3	3	4
0	0	1	1	2	2	0	0	1
0	0	1	1	2	2	0	0	1
3	3	4	4	5	5	3	3	4
3	3	4	4	5	5	3	3	4
0	0	1	1	2	2	0	0	1

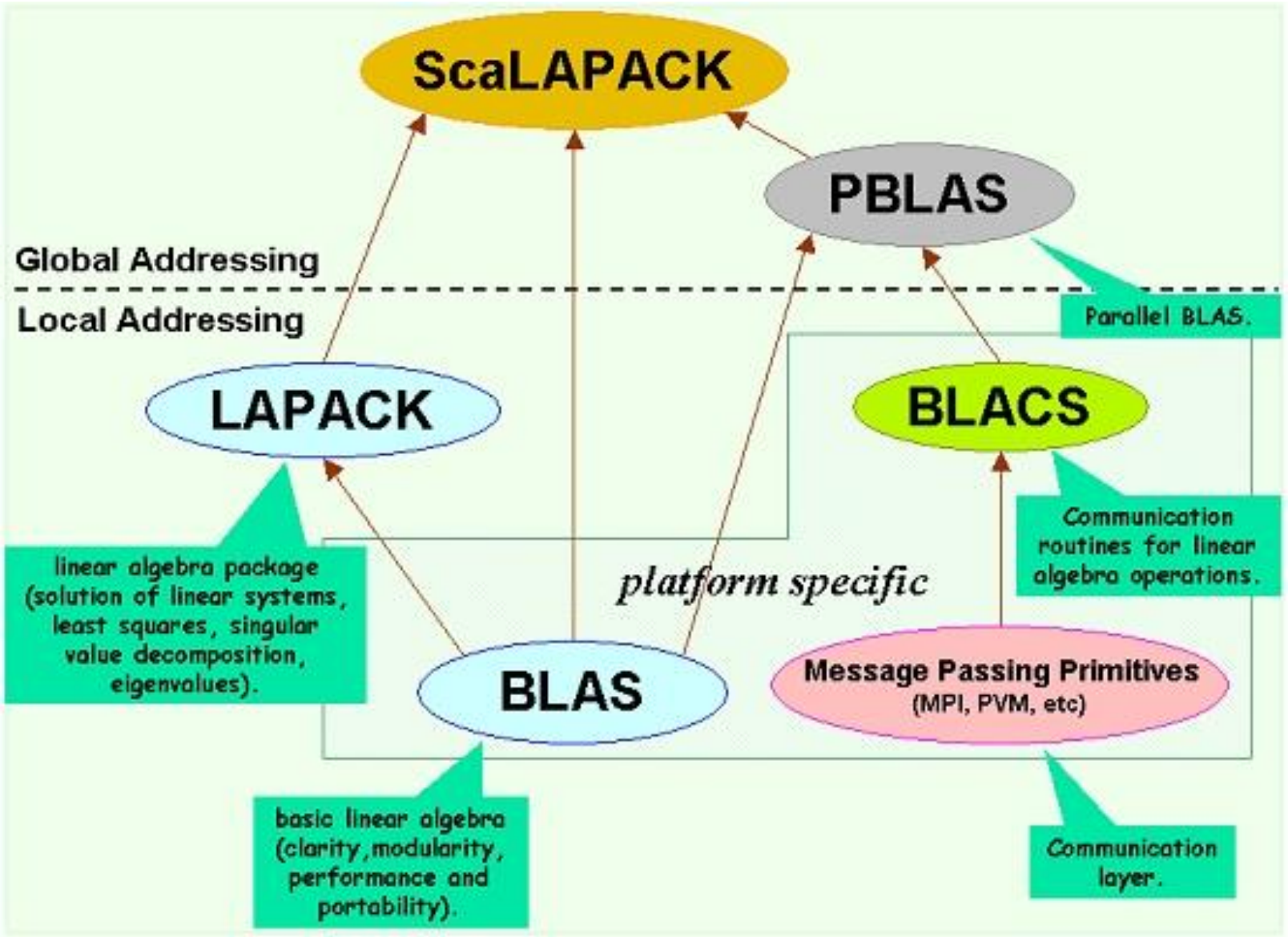
ScaLAPACK

- ScaLAPACK (Scalable LAPACK): http://www.netlib.org/scalapack/
- Includes a subset of LAPACK routines redesigned for *distributed memory* architectures
- Depends on PBLAS
- Key ideas:
  - a **block cyclic data distribution** for dense matrices and a block data distribution for banded matrices, parametrizable at runtime;
  - **block-partitioned algorithms** to ensure high levels of data reuse;
  - well-designed **low-level modular components** that simplify the task of parallelizing the high level routines by making their source code the same as in the sequential case.
- ScaLAPACK Example Programs: http://www.netlib.org/scalapack/examples/

LAPACK and ScaLAPACK

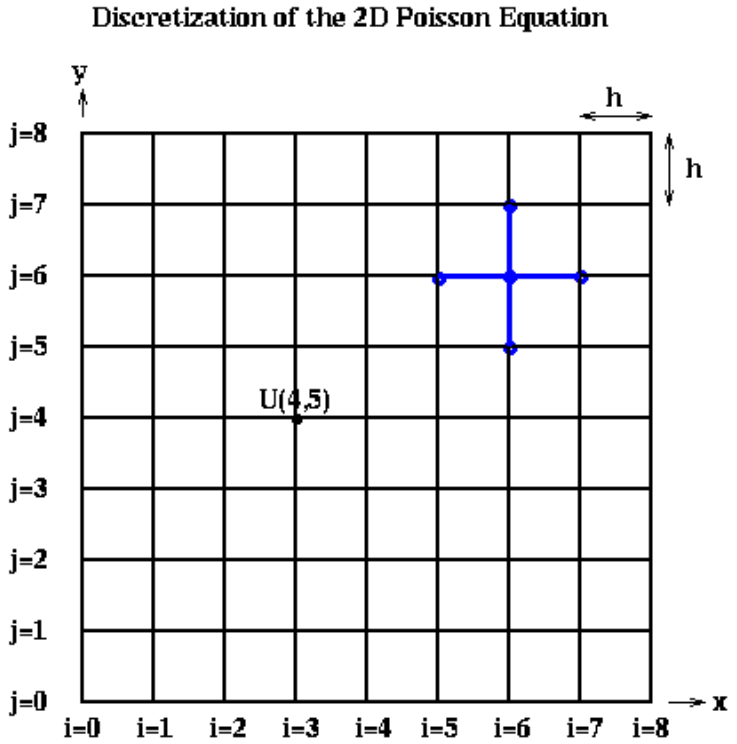
	LAPACK	ScaLAPACK
Machines	Workstations, Vector, SMP	Distributed Memory, DSM
Based on	BLAS	BLAS, BLACS
Functionality	Linear Systems Least Squares Eigenproblems	Linear Systems Least Squares Eigenproblems (less than LAPACK)
Matrix types	Dense, band	Dense, band, out-of-core
Error Bounds	Complete	A few
Languages	F77 or C	F77 and C
Interfaces to	C++, F90	HPF
Manual?	Yes	Yes
Where?	www.netlib.org/ lapack	www.netlib.org/ scalapack

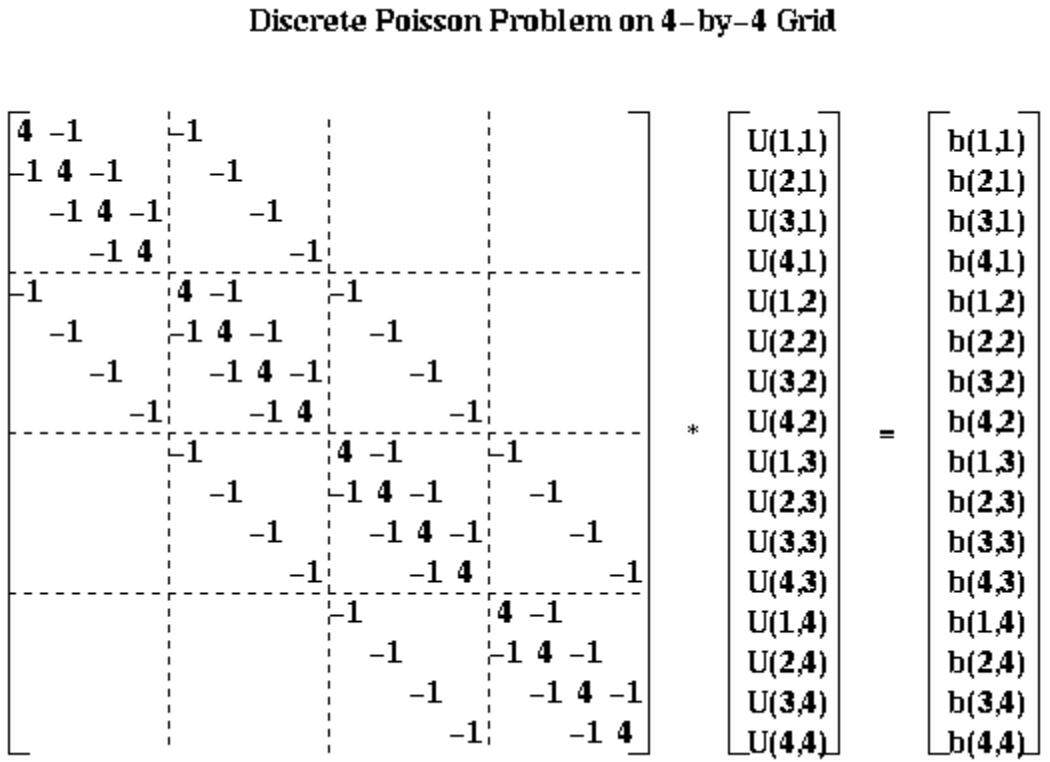
ScaLAPACK Software Hierarchy



Sparse Linear Systems

- A matrix is sparse if it has relatively few *nonzeros* in its entries
- Sparsity can be exploited to use *much less* than $O(n^2)$ storage and $O(n^3)$ work required in standard approach to solving system with dense matrix
- For example, consider the discrete Poisson equation with a 5-point stencil:





Sparse Linear Algebra

- Dense methods
  - Direct representation of matrices with simple data structures (no need for indexing data structure)
  - Mostly $O(n^3)$ factorization algorithms
- Sparse direct methods
  - Direct representation, keep only the nonzeros
  - Factorization costs depend on problem structure (1D cheap; 2D reasonable; 3D gets expensive; not easy to give a general rule, and NP hard to order for optimal sparsity)
  - Robust, but hard to scale to large 3D problems
- Iterative methods
  - Only need $y = Ax$ (maybe $y = A^T x$)
  - Produce successively better (?) approximations
  - Good convergence depends on preconditioning
  - Best preconditioners are often hard to parallelize

Linear Algebra Software

- Dense: LAPACK, ScaLAPACK, ...
- Sparse direct: SuperLU, UMFPACK, MUMPS, PARDISO, SPOOLES, ...
- Sparse iterative: Hypre and many others
- Sparse mega-libraries (frameworks)
  - PETSc (Argonne, object-oriented C)
  - Trilinos (Sandia, C++)
- Good references:
  - Survey on “Parallel Linear Algebra Software”: http://www.netlib.org/utk/people/JackDongarra/PAPERS/siam-la-sw-survey-chapter13-2006.pdf
  - ACTS collection at NERSC: http://acts.nersc.gov/tools.html (dead!)

Storage Formats of Sparse Matrices

- Sparse matrices are typically stored in special formats that store only nonzero entries, along with indices to identify their locations in matrix, such as
  - Compressed Row Storage (CRS)
  - Compressed Column storage (CCS)
  - Block Compressed Row Storage (BCRS)
- Survey of Sparse Matrix Storage Formats: http://netlib.org/linalg/html_templates/node90.html
- Explicitly storing indices incurs additional storage overhead and makes arithmetic operations on nonzeros less efficient due to indirect addressing to access operands, so they are beneficial only for very sparse matrices
- Storage format can have big impact on the effectiveness of different versions of same algorithm (with different ordering of loops)
- Besides direct methods, these storage formats are also important in implementing iterative and mutigrid solvers

Example of Compressed Row Storage (CRS)

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}.$$

n = dimension = 6

nnz = # of nonzeros = 19

1 nnz=19

val	10	-2	3	9	3	7	8	7	3 ··· 9	13	4	2	-1
col_ind	1	5	1	2	6	2	3	4	1 ··· 5	6	2	5	6

row_ptr	1	3	6	9	13	17	20

row_ptr(n+1) = nnz + 1

row_ptr(7) = 20

Sparse Direct Methods

- LU factorization of matrix A (L is a lower triangular matrix and U upper triangular matrix):

  $$A = L\ U$$

- Solve $A\ x = b$:
  1. $L\ y = b$
  2. $U\ x = y$
- Symmetric versions: $A = L\ L^T$, $A = L\ D\ L^T$
- When are direct methods effective?
  - 1D: Always, even on many, many processors
  - 2D: Almost always, except on many, many processors
  - 2.5D: Most of the time
  - 3D: Only for “small/medium” problems on “small/medium” processor counts
- Bottom line: Direct sparse solvers should always be in your toolbox

Sparse Direct Solver Packages

- HSL: http://www.hsl.rl.ac.uk
- MUMPS: http://mumps.enseeiht.fr
- Pardiso: http://www.pardiso-project.org
- PaStiX: http://pastix.gforge.inria.fr
- SuiteSparse: http://faculty.cse.tamu.edu/davis/suitesparse.html
- SuperLU: http://crd-legacy.lbl.gov/~xiaoye/SuperLU/index.html
- WSMP: http://researcher.watson.ibm.com/researcher/view_project.php?id=1426
- Trilinos/Amesos/Amesos2: http://trilinos.org

Notes:

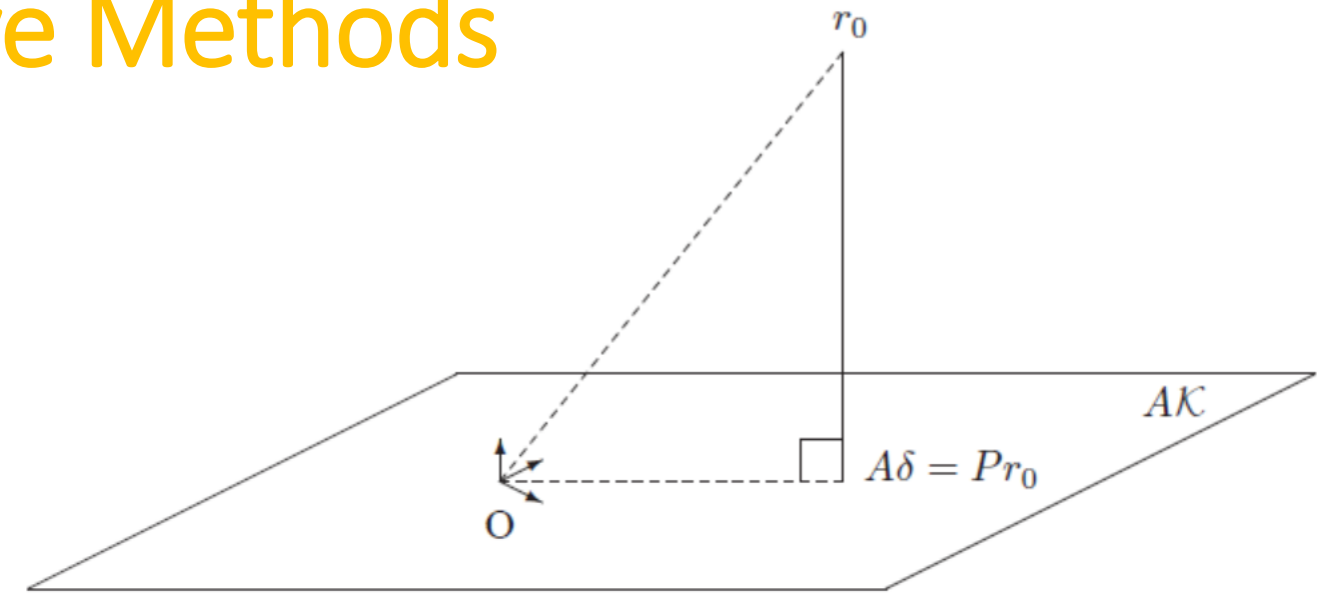
- All have threaded parallelism
- All but SuiteSparse have distributed memory (MPI) parallelism

SuperLU

- SuperLU (Supernodal LU) is a general purpose library for the *direct* solution of *large*, *sparse*, *nonsymmetric* systems of linear equations
  http://crd-legacy.lbl.gov/~xiaoye/SuperLU/
- Written in C; callable from either C or Fortran
- SuperLU package comes in three different flavors:
  - **SuperLU** for sequential machines
  - **SuperLU_MT** for shared memory parallel machines
  - **SuperLU_DIST** for distributed memory
- Users' guide: http://crd-legacy.lbl.gov/~xiaoye/SuperLU/ug.pdf
- Examples: http://acts.nersc.gov/superlu/index.html
- SuperLU and STRUMPACK Sparse Direct Solver and Preconditioner:
  http://press3.mcs.anl.gov/atpesc/files/2015/08/Li_fastmath-superlu-atpesc_120.pdf

Iterative Methods

- Basic iterative methods
  - Jacobi
  - Gauss-Seidel
  - SOR
- Krylov subspace methods
  - Ritz-Galerkin approach: FOM and CG
  - Minimum Residual approach: GMRES and MINRES
  - Petrov-Galerkin approach: Bi-CG and QMR
  - Minimum Error approach: SYMMLQ and GMERR
  - etc.
- Preconditioning is often the vital component in the development of efficient solvers



What is Preconditioning?

Original Linear System: $Ax = b$

Preconditioning: $M = M_1 M_2$

$A = M_1 \tilde{A} M_2$

We instead solve the preconditioned system: $\tilde{A}\tilde{x} = \tilde{b}$

where: $\tilde{A} = M_1^{-1} A M_2^{-1}$

$\tilde{b} = M_1^{-1} b$

$\tilde{x} = M_2 x$

The art of preconditioning:

1. $\tilde{A}$ is close to identity, or has nice properties
2. M can be efficiently inverted

http://people.maths.ox.ac.uk/wathen/preconditioning.pdf

- **Hypre** is a library for solving large, sparse linear systems of equations on massively parallel computers
  http://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods/software
  https://github.com/LLNL/hypre
- User's Manual:
  http://computation.llnl.gov/sites/default/files/public/hypre-2.11.2_usr_manual.pdf
- Reference Manual:
  http://computation.llnl.gov/sites/default/files/public/hypre-2.11.2_ref_manual.pdf
- Tutorial:
  https://redmine.scorec.rpi.edu/anonsvn/fastmath/docs/ATPESC_2015/TutorialPresentations/fastmath-hypre-atpesc-08-2015.pdf
  http://press3.mcs.anl.gov/atpesc/files/2016/02/Falgout_fastmath-hypre-atpesc_1250.pdf
- Hands-on Examples:
  https://redmine.scorec.rpi.edu/anonsvn/fastmath/docs/ATPESC_2016/Exercises/hypre/examples/README.html

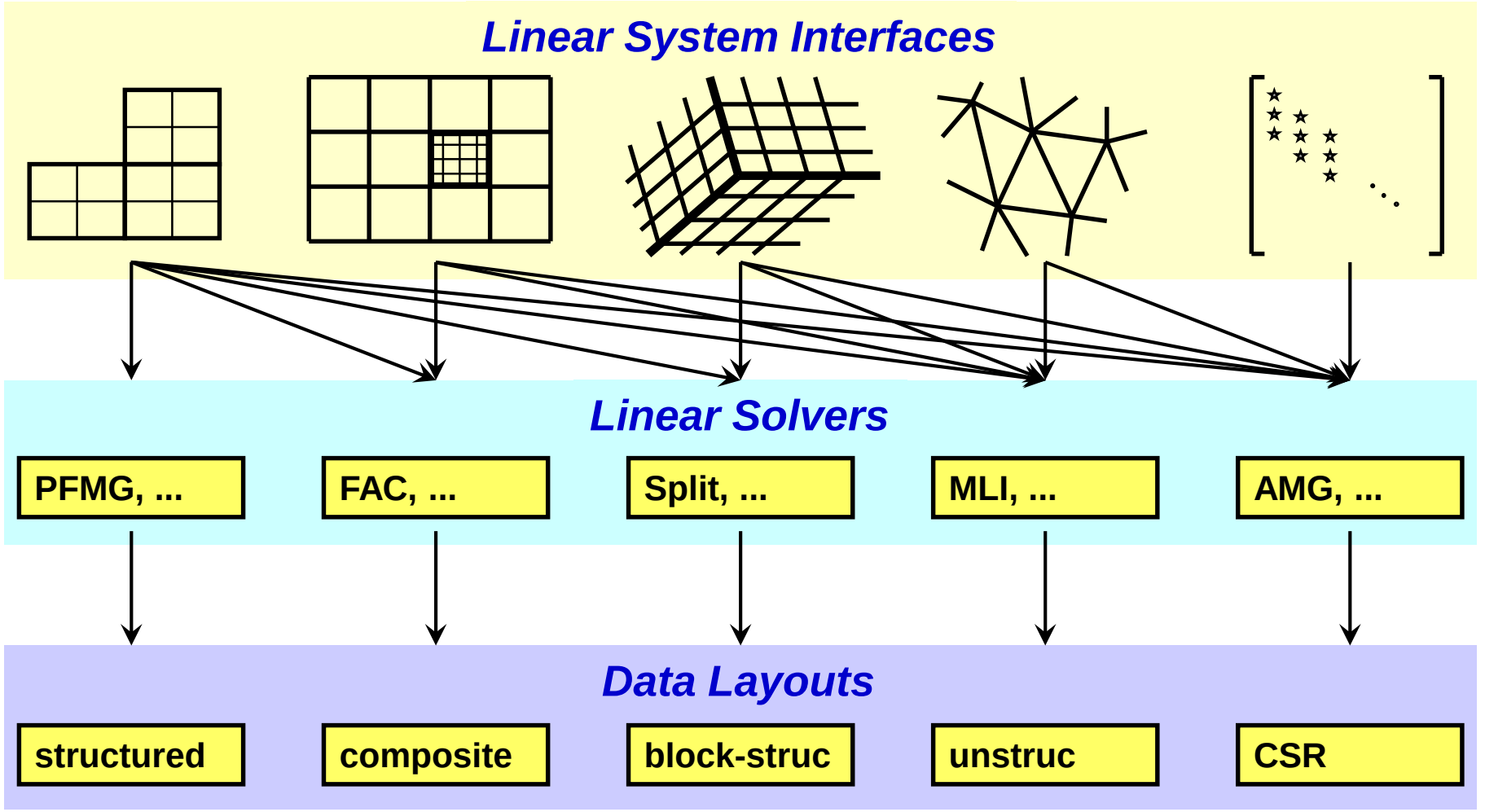
Getting Started with Hypre

- Before writing your code:
  - choose a linear system interface
  - choose a solver / preconditioner
  - choose a matrix type that is compatible with your solver / preconditioner and system interface

- Now write your code:
  - build auxiliary structures (e.g., grids, stencils)
  - build matrix/vector through system interface
  - build solver/preconditioner
  - solve the system
  - get desired information from the solver

Linear System Interfaces

Linear system interfaces are necessary to provide “best” solvers and data layouts

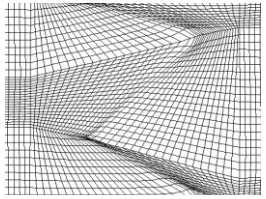


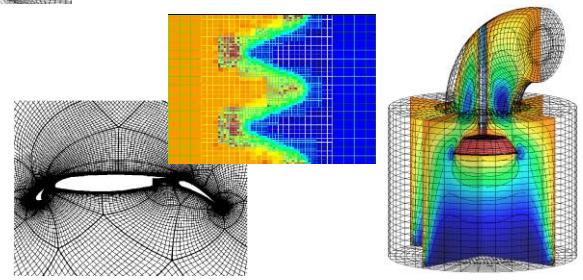
Why multiple interfaces?

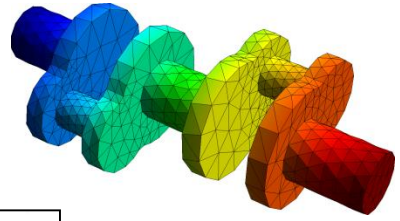
- Provides natural “views” of the linear system
- Eases some of the coding burden for users by eliminating the need to map to rows/columns
- Provides for more efficient (scalable) linear solvers
- Provides for more effective data storage schemes and more efficient computational kernels

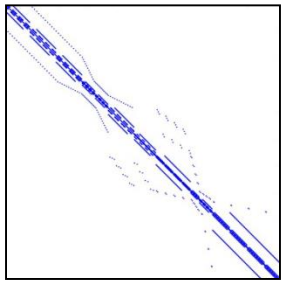
Interfaces supported by Hypre

- Structured-Grid (`Struct`)
  - *logically rectangular grids*
- Semi-Structured-Grid (`SStruct`)
  - *grids that are mostly structured*
- Finite Element (`FEI`)
  - *unstructured grids with finite elements*
- Linear-Algebraic (`IJ`)
  - *general sparse linear systems*









Solvers and Preconditioners

Data Layouts	Solvers	System Interfaces			
		Struct	SStruct	FEI	IJ
Structured	Jacobi	✓	✓		
	SMG	✓	✓		
	PFMG	✓	✓		
	Split		✓		
Semi-structured	SysPFMG		✓		
	FAC		✓		
	Maxwell		✓		
Sparse matrix	AMS, ADS		✓	✓	✓
	BoomerAMG		✓	✓	✓
	MLI		✓	✓	✓
	ParaSails		✓	✓	✓
	Euclid		✓	✓	✓
	PILUT		✓	✓	✓
Matrix free	PCG	✓	✓	✓	✓
	GMRES	✓	✓	✓	✓
	BiCGSTAB	✓	✓	✓	✓
	Hybrid	✓	✓	✓	✓

Setup and Use of solvers

- Create the solver

```
HYPRE_SolverCreate(MPI_COMM_WORLD, &solver);
```

- Set parameters

```
HYPRE_SolverSetTol(solver, 1.0e-06);
```

- Prepare to solve the system

```
HYPRE_SolverSetup(solver, A, b, x);
```

- Solve the system

```
HYPRE_SolverSolve(solver, A, b, x);
```

- Get solution info out via system interface

```
HYPRE_StructVectorGetValues(struct_x, index, values);
```

- Destroy the solver

```
HYPRE_SolverDestroy(solver);
```

Fast Fourier Transform (FFT)

- A Fast Fourier Transform (FFT) algorithm compute the Discrete Fourier Transform (DFT) of a sequence, or its inverse
- Fourier analysis converts a signal from its original domain (often *time* or *space*) to a representation in the *frequency* domain, and vice versa
- FFT reduces the complexity of computing the DFT from $O(n^2)$ to $O(n \log n)$, where *n* is the data size
- FFT is widely used for many applications in engineering, science, and mathematics, e.g.:
  - Signal processing
  - Image processing
  - Solving Poisson's Equation nearly optimally
  - Fast multiplication of large integers
  - …

Discrete Fourier Transform (DFT)

- Let *i* = sqrt(-1) and index matrices and vectors from 0
- Let *f* be a 1D function defined on a grid and labeled with index *m = 0 … N-1*, i.e., *f* is a vector of length *N*
- The **Discrete Fourier Transform** *F(f)* is another vector of length *N*:

  $F(f) = \Omega f$

  where Ω is a *N*N* matrix with matrix elements: $\Omega_{km} = \varpi^{-k*m}$

  and $\varpi = e^{2\pi i/N} = \cos\left(\frac{2\pi}{N}\right) + i * \sin(\frac{2\pi}{N})$

  ϖ is called N^{th} root of unity, because $\varpi^N = 1$.

  For example, if *N*=4:

  $\varpi = i, \varpi^2 = -1, \varpi^3 = -i, \varpi^4 = 1$

Inverse and other Fourier Transforms

- f can be reconstructed from its discrete Fourier transformation *F(f)* by
  $$f = \Omega^* F(f)/N$$
  where * denotes complex conjugation
  Ω^* is an *N* by *N* matrix and *F(f)* is a vector of length *N*
- Most applications require both calculating Fourier transforms and reconstructing functions from their Fourier transforms
- However these are essentially the same algorithm as seen above and so we only need to illustrate one case
- Issues with parallelism and optimal performance are identical
- For solving the Poisson equation and various other applications, we use variations on the Fourier transform
  - The *sin* transform – use imaginary part of *F*
  - The *cos* transform – use real part of *F*

1D FFT

- Basic 1D Discrete Fourier Transform (DFT):

$$F_N(k,f) = \sum_{m=0}^{N-1} f(m)\exp(-\frac{2\pi ikm}{N})$$

- Inverse:

$$f_N(m,F) = \frac{1}{N}\sum_{k=0}^{N-1} F(k)\exp(\frac{2\pi ikm}{N})$$

- In a naïve analysis it would take $O(N^2)$ complex floating point additions and multiplications. However, we can group terms together to reduce this to $O(N\log N)$.
- This can be done in many ways and starts with a factorization $N=N_1N_2$ (product of integers) and a recursive iteration of factorization to values N_1, N_2 where special algorithms exist.

- FFTW (Fastest Fourier Transform in the West): http://fftw.org/
- FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data
- Both C and Fortran interfaces
- Computational *kernels* (80% of code) automatically generated, similar in spirit to ATLAS
- Self-optimized for your hardware = *portability* + *performance*
  - FFTW's performance is typically superior to that of other publicly available FFT software, and is even competitive with vendor-tuned codes
  - FFTW's performance is *portable*

FFTW on NERSC systems: http://www.nersc.gov/users/software/programming-libraries/math-libraries/fftw/

Parallel Versions of FFTW

- **Cilk** version: for shared memory architectures; parallelizes both one and multi-dimensional transforms
- **Threaded** version: for shared memory architectures; parallelizes both one and multi-dimensional transforms
- **MPI** version: for both distributed memory machine and shared memory architectures; parallelizes multi-dimensional transforms

FFTW manual: http://fftw.org/fftw3_doc/

Why is FFTW fast?

- FFTW implements many FFT algorithms. A *planner* picks the best composition by measuring the speed of different combinations.
- The resulting *plan* is executed with explicit recursion, which enhances *locality*
- The base cases of the recursion are *codelets*: highly-optimized dense code that is automatically generated by a special-purpose “compiler”

FFTW is easy to use

```
#include <fftw3.h>
...
{
    fftw_complex *in, *out;
    fftw_plan p;
    ...
    in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
    ...
    fftw_execute(p); /* repeat as needed */
    ...
    fftw_destroy_plan(p);
    fftw_free(in);
    fftw_free(out);
}
```

FFTW Fortran Example

```
...
include 'fftw3.f'
...
integer*8 :: plan
double complex, allocatable :: in(:), out(:)
...
allocate(in(n))
allocate(out(n))

call dfftw_plan_dft_1d(plan,n,in,out,FFTW_FORWARD,FFTW_ESTIMATE)
call dfftw_execute(plan)
call dfftw_destroy_plan(plan)

deallocate(in)
deallocate(out)
...
```

http://www.fftw.org/doc/Fortran-Examples.html

FFTW3 MPI Example

```
#include <fftw3-mpi.h>
int main(int argc, char **argv) {
  const ptrdiff_t N0 = 10000, N1 = 10000;

  fftw_plan plan;
  fftw_complex *data; //local data of course
  ptrdiff_t alloc_local, local_n0, local_0_start, i, j;

  MPI_Init(&argc, &argv);
  fftw_mpi_init();

  /* get local data size and allocate */
  alloc_local = fftw_mpi_local_size_2d(N0, N1, MPI_COMM_WORLD,
                    &local_n0, &local_0_start);
  data = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) * alloc_local);
  printf("%i\n", local_n0);

  /* create plan for forward DFT */
  plan = fftw_mpi_plan_dft_2d(N0, N1, data, data, MPI_COMM_WORLD,
              FFTW_FORWARD, FFTW_ESTIMATE);
```

FFTW3 MPI Example (cont'd)

```
  /* initialize data to some function my_function(x,y) */
  for (i = 0; i < local_n0; ++i) {
    for (j = 0; j < N1; ++j) {
      data[i*N1 + j][0] = local_0_start;
      data[i*N1 + j][1] = i;
    }
  }

  /* compute transforms, in-place, as many times as desired */
  fftw_execute(plan);
  fftw_destroy_plan(plan);
  fftw_free(data);
  MPI_Finalize();
  printf("finalize\n");
  return 0;
}
```

To compile the example code on Cori:

module load fftw

cc FFTW3MPI2DExample.c -l fftw3_mpi -lfftw3

Intel MKL

- Intel **MKL** (Math Kernel Library) is a library of optimized math routines
  - highly *vectorized* and *threaded* routines for Linear Algebra, FFT, Vector Math and Statistics
  - MPI versions of LAPACK, FFT and sparse solvers for distributed memory architectures
- *Hand-optimized* specifically for Intel processors
- Intel MKL has the following functional categories:
  - **Linear algebra**: including optimized BLAS, BLACS, LAPACK, ScaLAPACK, PBLAS, sparse BLAS, sparse solvers such as PARDISO, iterative sparse solvers, and Parallel Direct Sparse Solver for Clusters.
  - **Fast Fourier Transforms**: Multidimensional (up to 7D) FFTs, FFTW interfaces, Cluster FFT
  - **Vector Mathematical Functions (VML)**
  - **Statistical Functions (VSL)**
  - **Data fitting**
  - others: including Vector Random Number Generators, Poisson Solvers, Optimization Solvers

Intel MKL Documentation

- Documentation: https://software.intel.com/en-us/articles/intel-math-kernel-library-documentation
- Intel MKL Developer Reference – C: http://software.intel.com/en-us/mkl-reference-manual-for-c
- Intel MKL Developer Reference – Fortran: http://software.intel.com/en-us/mkl-reference-manual-for-fortran
- Intel MKL for Linux OS Developer Guide: http://software.intel.com/en-us/mkl-for-linux-userguide
- Intel MKL Cookbook: http://software.intel.com/en-us/mkl_cookbook
- Intel MKL LAPACK examples: http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/mkl_lapack_examples/index.htm

On Hyades, source codes and data files for Intel MKL examples are located at $MKLROOT/examples

NERSC: http://www.nersc.gov/users/software/programming-libraries/math-libraries/mkl/

FFT in MKL

- Chapter 10 of *Intel MKL Developer Reference*
- FFT implementations in Intel MKL:
  - FFT functions for single-processor or shared-memory systems
  - Cluster FFT functions for distributed-memory architectures
- Intel MKL also offers FFTW2 and FFTW3 interfaces
  - Appendix D of *Intel MKL Developer Reference*
  - enable applications using FFTW to gain performance with Intel MKL without changing the program source code

FFT and Cluster FFT functions

Both FFT and Cluster FFT functions compute an FFT in five steps:

1. Allocate a fresh descriptor for the problem with a call to the **DftiCreateDescriptor** or **DftiCreateDescriptorDM** function. The descriptor captures the configuration of the transform, such as the dimensionality, sizes, number of transforms, memory layout of the input/output data, and scaling factors.
2. Optionally adjust the descriptor configuration with a call to the **DftiSetValue** or **DftiSetValueDM** function.
3. Commit the descriptor with a call to the **DftiCommitDescriptor** or **DftiCommitDescriptorDM** function.
4. Compute the transform with a call to the **DftiComputeForward** / **DftiComputeBackward** or **DftiComputeForwardDM** / **DftiComputeBackwardDM** functions as many times as needed.
5. Deallocate the descriptor with a call to the **DftiFreeDescriptor** or **DftiFreeDescriptorDM** function.

Intel MKL FFT example

```
/* C example, float _Complex is defined in C9X */
#include "mkl_dfti.h"
float _Complex x[32];
float y[34];
DFTI_DESCRIPTOR_HANDLE desc1;
DFTI_DESCRIPTOR_HANDLE desc2;
MKL_LONG status;
//...put input data into x[0],...,x[31]; y[0],...,y[31]
status = DftiCreateDescriptor( &desc1, DFTI_SINGLE, DFTI_COMPLEX, 1, 32);
status = DftiCommitDescriptor( desc1 );
status = DftiComputeForward( desc1, x);
status = DftiFreeDescriptor( &desc1 );
/* result is x[0], ..., x[31] */
status = DftiCreateDescriptor( &desc2, DFTI_SINGLE, DFTI_REAL, 1, 32);
status = DftiCommitDescriptor( desc2);
status = DftiComputeForward( desc2, y);
status = DftiFreeDescriptor( &desc2 );
/* result is given in CCS format */
```

Compared to FFTW

```
#include <fftw3.h>
...
{
    fftw_complex *in, *out;
    fftw_plan p;
    ...
    in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
    ...
    fftw_execute(p); /* repeat as needed */
    ...
    fftw_destroy_plan(p);
    fftw_free(in);
    fftw_free(out);
}
```

Linking with Intel MKL

- Intel MKL is a vast and complex library, with a bewildering array of components, interfaces and options; so linking with it can be tricky
- Chapter 3 of *Intel MKL Developer Reference* is devoted to this topic
- **Intel MKL Link Line Advisor**: an online tool, which requests information about your system and on how you intend to use Intel MKL, then generates the appropriate link line for your application

  https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor
- **Command-line Link Tool**, which provides the options, libraries, and environment variables to use, and can also performs compilation and building of your application. To learn more about it, run:

  ```
  ${MKLROOT}/tools/mkl_link_tool --help
  ```

Linking with sequential MKL

The most common usage scenario is to link with sequential (serial) version of Intel MKL. For starters, you can use the following linking options:

```
-lmkl_intel_lp64 -lmkl_core -lmkl_sequential
```

which will use the **LP64** interface (32-bit integer) and link with the *dynamic* libraries.

For example, to compile a LAPACK program and link with sequential version of Intel MKL:

```
ifort -o lapackpgm.x lapackpgm.f \
      -lmkl_intel_lp64 -lmkl_core -lmkl_sequential
```

A shortcut is to use the -mkl=sequential option for Intel compilers:

```
ifort -o lapackpgm.x lapackpgm.f -mkl=sequential
```

If you want to link with the *static* libraries, use:

```
-Wl,--start-group ${MKLROOT}/lib/intel64/libmkl_intel_lp64.a \
${MKLROOT}/lib/intel64/libmkl_core.a \
${MKLROOT}/lib/intel64/libmkl_sequential.a -Wl,--end-group
```

or

```
-static -Wl,--start-group -lmkl_intel_lp64 \
        -lmkl_core -lmkl_sequential -Wl,--end-group
```

or

```
-static -mkl=sequential
```

Linking with multi-threaded MKL

To link *dynamically* with multi-threaded version of Intel MKL, use the following linking options:

```
-lmkl_intel_lp64 -lmkl_core -lmkl_intel_thread -openmp
```

or use Intel compilers option `-mkl=parallel` or `-mkl`.

To link *statically*, use:

```
-Wl,--start-group \
${MKLROOT}/lib/intel64/libmkl_intel_lp64.a \
${MKLROOT}/lib/intel64/libmkl_core.a \
${MKLROOT}/lib/intel64/libmkl_intel_thread.a \
-Wl,--end-group
-openmp
```

or use Intel compilers option `-static -mkl=parallel` or `-static -mkl`.

Linking with MKL Cluster Components

A quick way to link with the Intel MKL cluster components, like BLACS, CDFT (cluster FFT), ScaLAPACK, is to use Intel compilers option `-mkl=cluster` for Intel MPI. For example:

```
mpiifort -o mpipgm.x mpipgm.f90 -mkl=cluster
```

Otherwise, use **Intel MKL Link Line Advisor**.

Single Dynamic Library (SDL)

Intel MKL also provides another option for quick linking of your application: Single Dynamic Library (SDL). To use SDL, link your application with *libmkl_rt.so* (-lmkl_rt). For example:

```
ifort -o lapackpgm.x lapackpgm.f -lmkl_rt
```

SDL enables you to select the interface and threading library for Intel MKL at run time. By default, linking with SDL provides:

- Intel LP64 interface on systems based on the Intel 64 architecture
- Intel threading

Cray Scientific and Math Libraries

- The Cray Scientific and Math Libraries package, **LibSci**, is a collection of numerical routines optimized for best performance on Cray systems http://www.nersc.gov/users/software/programming-libraries/math-libraries/libsci/
- The Cray LibSci collection contains the following libraries:
  - **BLAS** (Basic Linear Algebra Subroutines)
  - **BLACS** (Basic Linear Algebra Communication Subprograms)
  - **LAPACK** (Linear Algebra Routines)
  - **LAPACKE** (C interfaces to LAPACK Routines)
  - **ScaLAPACK** (Scalable LAPACK)
  - **IRT** (Iterative Refinement Toolkit)
  - **FFTW2** (the Fastest Fourier Transforms in the West, release 2)
  - **FFTW3** (the Fastest Fourier Transforms in the West, release 3)
- The modulefile is loaded by default. No user action is required. This is true for all programming environments (Intel, Cray, GNU) as long as you use the Cray compiler wrappers (ftn, cc, and CC).

LibSci Documentation

- Chapter 7 of Cray XC Series Programming Environment User Guide
- Man pages, including: intro_libsci(3s), intro_blas1(3s), intro_blas2(3s), intro_blas3(3s), intro_blacs(3s), intro_lapack(3s), intro_scalapack(3s), intro_irt(3), intro_fftw2(3), intro_fftw3(3)

A quick Aside on Cray Compiler

Pattern Matching optimization capability of *Cray compiler*: It can recognize source code patterns that correspond to highly optimized routines in its **libsci** math library and uses the library code when it creates the executable program.
http://www.nersc.gov/users/getting-started/

```
do i=1,idim
  do j=1,idim
    do k=1,idim
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
    enddo
  enddo
enddo
```

```
scale=1.0
call dgemm("N","N",&
       idim,idim,idim,scale,&
       a,idim,b,idim,scale,&
       c,idim)
```

System	Compiler	Optimization	dgemm	Fortran	C	dgemm/F	dgemm/C	matmul
Edison	Intel	-fast -no-ipo	24.24 GF	13.53 GF	15.15 GF	1.79	1.60	9.94 GF
Edison	Cray	default	23.68 GF	23.66 GF	23.62 GF	1.00	1.00	23.61 GF
Edison	gnu	-Ofast	23.52 GF	.15 GF	.65 GF	155.91	36.08	2.17 GF

Frameworks

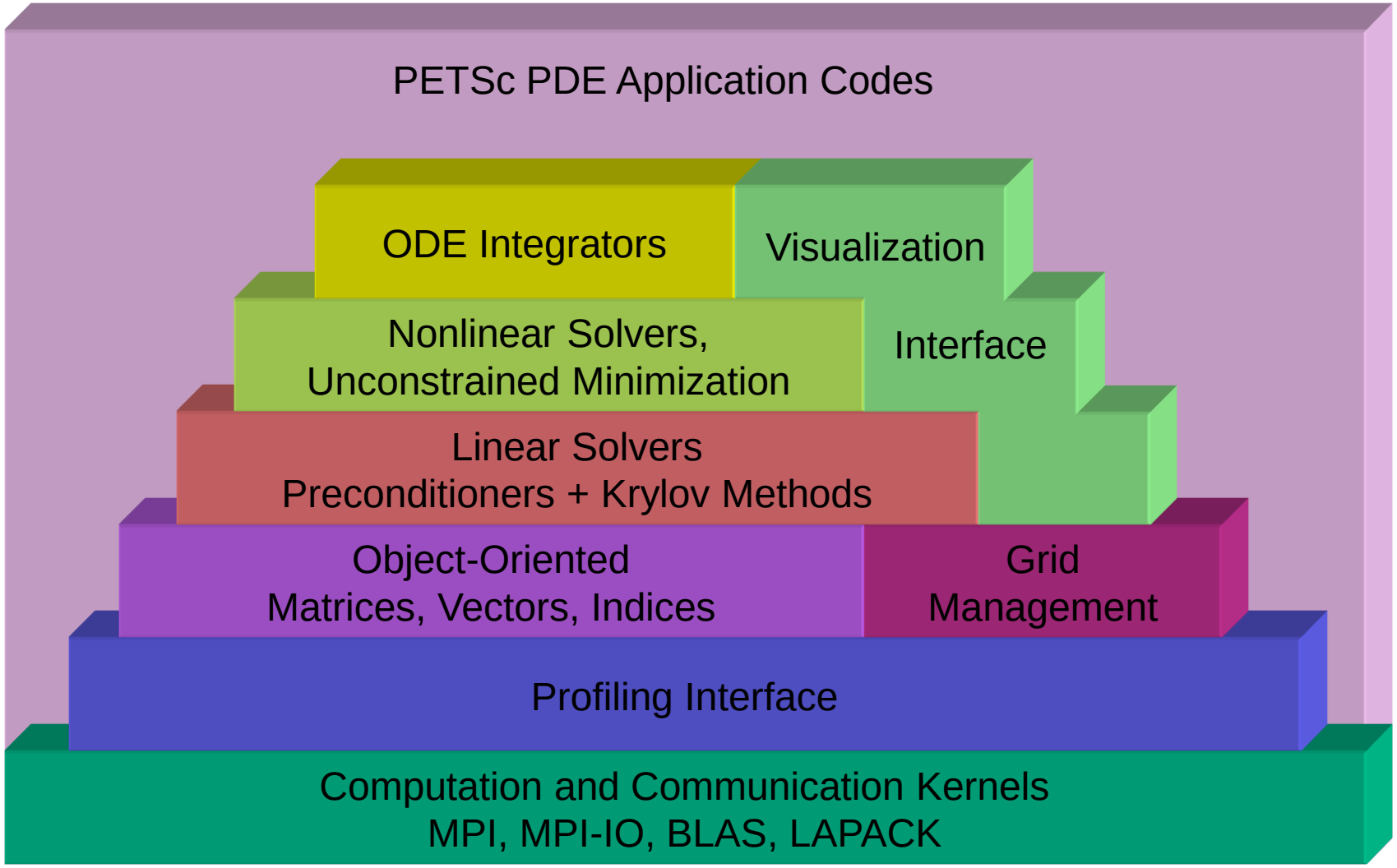
- Motivations:
  - Writing hand-parallelized application codes from scratch is extremely difficult and time consuming
  - We can ease the development of parallel application codes by utilizing general-purpose, parallel numerical frameworks
- The FASTMath SciDAC Institute has a good list of scalable frameworks for HPC applications:

  http://www.fastmath-scidac.org/software-catalog.html
  - PETSc
  - Trilinos
  - BoxLib
  - Chombo
  - etc.

PETSc

- PETSc = Portable, Extensible Toolkit for Scientific Computation http://www.mcs.anl.gov/petsc/
- A suite of *composable* data structures and algorithms for the scalable (parallel) solution of scientific applications modeled by PDEs
- Supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism
- Usable from C, C++, Fortran 77/90 and Python
- Documentation: http://www.mcs.anl.gov/petsc/documentation/index.html
- Tutorials: http://www.mcs.anl.gov/petsc/documentation/tutorials/index.html

PETSc Structure



PETSc 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 Schwartz	Block Jacobi	Jacobi	ILU	ICC	LU (Sequential only)	Others

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

Vectors

Index Sets			
Indices	Block Indices	Stride	Other

- https://trilinos.org/
- An object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems
  - A collection of over 50 packages written in C++
  - Composable and extensible
- Core capabilities include linear solvers and preconditioners (e.g. ML, ShyLU), nonlinear solvers and other analysis algorithms (e.g. NOX), and partitioning, load balancing, and data ordering algorithms (e.g. Zoltan)
- Tutorial: https://github.com/trilinos/Trilinos_tutorial/wiki/TrilinosHandsOnTutorial

Trilinos Package Summary

	Objective	Package(s)
Discretizations	Meshing & Discretizations	STK, Intrepid, Pamgen, Sundance, ITAPS, Mesquite
	Time Integration	Rythmos
Methods	Automatic Differentiation	Sacado
	Mortar Methods	Moertel
Services	Linear algebra objects	Epetra, Tpetra, Kokkos, Xpetra
	Interfaces	Thyra, Stratimikos, RTOp, FEI, Shards
	Load Balancing	Zoltan, Isorropia, Zoltan2
	“Skins”	PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos, Optika
	C++ utilities, I/O, thread API	Teuchos, EpetraExt, Kokkos, Triutils, ThreadPool, Phalanx, Trios
Solvers	Iterative linear solvers	AztecOO, Belos, Komplex
	Direct sparse linear solvers	Amesos, Amesos2, ShyLU
	Direct dense linear solvers	Epetra, Teuchos, Pliris
	Iterative eigenvalue solvers	Anasazi, Rbgen
	ILU-type preconditioners	AztecOO, IFPACK, Ifpack2, ShyLU
	Multilevel preconditioners	ML, CLAPS, Muelu
	Block preconditioners	Meros, Teko
	Nonlinear system solvers	NOX, LOCA, Piro
	Optimization (SAND)	MOOCHO, Aristos, TriKota, Globipack, Optipack
	Stochastic PDEs	Stokhos

Further Readings

- **Numerical Linear Algebra for High-Performance Computers**, 2nd edition, *by* Dongarra, Duff, Sorensen, & van der Vorst, SIAM, 1998
  http://epubs.siam.org/doi/book/10.1137/1.9780898719611
- **Applied Numerical Linear Algebra**, by James Demmel, SIAM, 1997
  http://epubs.siam.org/doi/book/10.1137/1.9781611971446
- **Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods**, *by* Barrett et al., SIAM, 1994
  http://www.netlib.org/linalg/html_templates/Templates.html
  http://epubs.siam.org/doi/book/10.1137/1.9781611971538
- **Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide**: *by* Bai et al., SIAM 2000
  http://www.netlib.org/utk/people/JackDongarra/etemplates/book.html
  http://epubs.siam.org/doi/book/10.1137/1.9780898719581
- **LAPACK Users' Guide**, 3rd edition: http://www.netlib.org/lapack/lug/
- **ScaLAPACK Users' Guide**: http://www.netlib.org/scalapack/slug/

Further Readings (cont'd)

- **Matrix Computation**, 4th edition, *by* Golub & Van Loan, Johns Hopkins University Press, 2012
- **Numerical Linear Algebra**, *by* Lloyd N. Trefethen & David Bau, III, SIAM 1997
- **Direct Methods for Sparse Linear Systems**, *by* Timothy Davis, SIAM, 2006
  http://epubs.siam.org/doi/book/10.1137/1.9780898718881
- **Iterative Methods for Sparse Linear Systems**, *by* Yousef Saad, SIAM, 2003
  http://www-users.cs.umn.edu/~saad/IterMethBook_2ndEd.pdf
- **SUMMA** (Scalable Universal Matrix Multiplication Algorithm) paper:
  http://www.netlib.org/lapack/lawns/lawn96.ps
- **Communication-Avoiding Algorithms**:
  http://www.cs.berkeley.edu/~demmel/AustMS2015_v2_noanimation.pdf
- Coppersmith–Winograd algorithm, which can multiply two n x n matrices in $O(n^{2.375477})$ time