PHY1610H - Scientific Computing: Linear Algebra – Applications & Libraries

Ramses van Zon and Marcelo Ponce

SciNet HPC Consortium/Physics Department
University of Toronto

March 2nd, 2021

Lecture 13: Numerical Linear Algebra

Applications

- Using packages for Linear Algebra
- BLAS
- LAPACK
- etc...

Theory // Review

- Review of Linear Algebra
- Solving Ax = b
- System Properties
- Direct Solvers
- Iterative Solvers
- Dense vs. Sparse matrices

How to write numerical linear algebra

As much as possible, rely on existing, mature software libraries for performing numerical linear algebra computations. By doing so...

- Focus on your code details
- Reduce the amout of code to produce/debug
- Libraries are tuned and optimized, ie. your code will run faster
- More options to switch methods if necessary

Software

Packages

- Netlib (`http://www.netlib.org`)
  - Maintained by UT and ORNL
  - Most of the code is public domain or freely licensed
  - Mostly written in FORTRAN 77 !
  - BLAS & LAPACK
- PETSc (`http://www.mcs.anl.gov/petsc/`)
  - Argonne National Labs
  - Open Source
  - C++
  - PDE & Iterative Linear Solvers
- Trilinos (`http://trilinos.sandia.gov/`)
  - Sandia National Labs
  - Collection of 50+ packages
  - Linear Solvers, Preconditioners, etc.
- Others

BLAS

Basic Linear Algebra Subroutines

- A well defined standard interface for these routines
- Many highly-tuned implementations exist for various platforms. (Atlas, Flame, Goto, PLASMA, cuBLAS, ...)
- (Interface vs. Implementation! Trick is designing a sufficiently general interface.)
- Higher-order operations (matrix factorizations, like as we'll see, gaussian elimiation) defined in LAPACK, on top of BLAS.

Typical BLAS routines

- Level 1:
  - `sdot` (dot product, single),
  - `zaxpy` ($\boldsymbol{ax} + \boldsymbol{y}$, dbl complex)
- Level 2:
  - `dgemv` (dbl matrix*vec),
  - `dsymv` (dbl symmetric matrix*vec)
- Level 3:
  - `sgemm` (general matrix-matrix),
  - `ctrmm` (triangular matrix-matrix)
- Somehow cryptic names, interfaces.

Prefixes

S: Single	**C**: Complex
D: Double	**Z**: Double Complex Matrix

Types

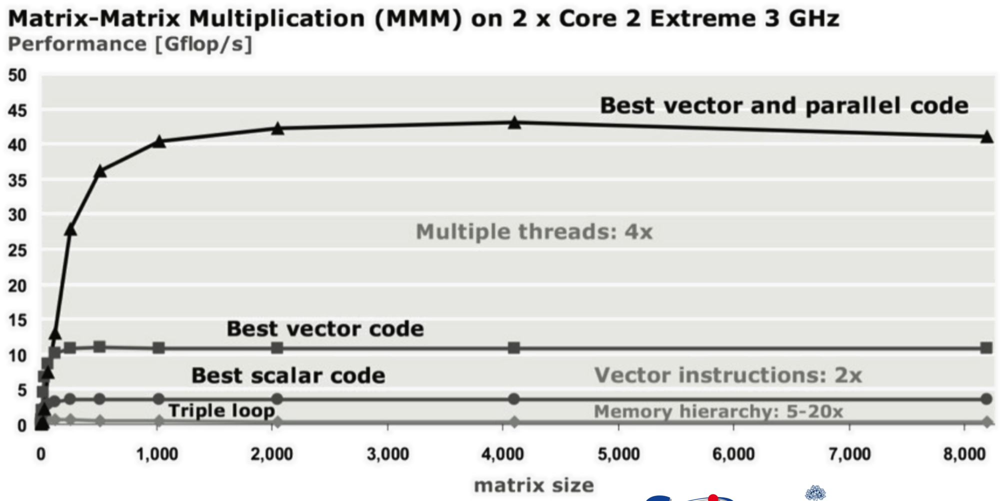
GE: General	**GB**: General Banded
HY: Hermetian	**HB**: Hermetian Banded
SY: Symmetric	**SB**: Symmetric Banded
TR: Triangular	**TB**: Triangular Banded
	TP: Triangular Packed

Why using Linear Algebra Packages?

- Why bother?
- Finding, downloading, installing the library
- Figuring out how to link
- C/Fortran issues
- Just write it - it's not rocket science ...

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        for (k=0; k<N; k++)
            c[i][j] = a[i][k]*b[k][j];
```

Never, ever, write your own...



Using BLAS

BLAS & LAPACK

- netlib provides “reference” implementation
- Most vendors provide optimized versions
- Commercial: Intel (MKL), AMD (ACML), IBM (ESSL)
- Open Source: ATLAS, GotoBLAS, OpenBLAS
- Fortran functions
- C interface using CBLAS and LAPACKE

Using BLAS

Install OpenBLAS (http://www.openblas.net/)

```
$ git clone git://github.com/xianyi/OpenBLAS.git
$ cd OpenBLAS
$ make FC=gfortran
$ make install PREFIX=$HOME/OpenBLAS/
```

Put the following in .bashrc

```
export BLAS_INC=${HOME}/OpenBLAS/include/
export BLAS_LIB=${HOME}/OpenBLAS/lib/
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${HOME}/OpenBLAS/lib/
```

However, if you are using SciNet, just use the corresponding modules,

eg. @Niagara cluster:

```
$ module spider openblas
$ module load openblas/0.2.20
```

BLAS Example: DSCAL ($\mathbf{x} \leftarrow \alpha\mathbf{x}$)

```
#include <iostream>
#include <cblas.h>
int main(int argc, char **argv) {
    double x[] = { 1.0, 2.0, 3.0 };
    double coeff = 4.323;
    int one = 1;
    int n = 3;
    //Direct Fortran call
    dscal_(&n, &coeff, &x[0], &one);
    for (int i = 0; i < n; i++)
    std::cout<<" "<<x[i];
    return 0;
}
```

```
#include <iostream>
#include <cblas.h>
int main(int argc, char **argv) {
    double x[] = { 1.0, 2.0, 3.0 };
    double coeff = 4.323;
    int one = 1;
    int n = 3;
    //Using CBLAS interface
    cblas_dscal(n,coeff,x,one);
    for (int i = 0; i < n; i++)
    std::cout<<" "<<x[i];
    return 0;
}
```

BLAS Example: DSCAL ($\mathbf{x} \leftarrow \alpha\mathbf{x}$)

compiling on your local machine,

```
$ g++ -std=c++14 dscal.cc -o dscal -I${BLAS_INC} -L${BLAS_LIB} -lopenblas
```

compiling on SciNet using the openblas-module,

```
$ g++ -std=c++14 dscal.cc -o dscal -I${SCINET_OPENBLAS_INC} -L${SCINET_OPENBLAS_LIB} -lopenblas
```

running...

```
$ ./dscal
$ 4.323 8.646 12.96
```

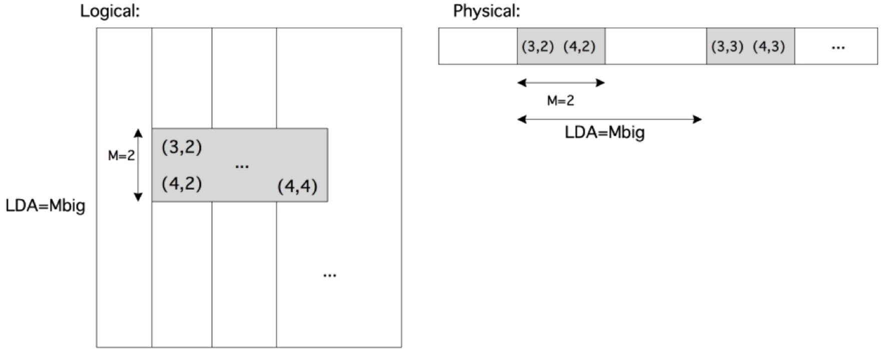
BLAS Example: DGEMM ($\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}$)

Documentation

- `http://www.netlib.org/blas/blast-forum/`
- `$ man dgemm`

```
NAME
       DGEMM - performs one of the matrix-matrix operations   C := alpha*op( A )*op( B ) + beta*C,

SYNOPSIS
       SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)

           DOUBLE                                                              PRECISION ALPHA,BETA
           INTEGER                                                             K,LDA,LDB,LDC,M,N
           CHARACTER                                                           TRANSA,TRANSB
           DOUBLE                                                              PRECISION
                                                                               A(LDA,*),B(LDB,*),C(LDC,*)
PURPOSE
       DGEMM  performs one of the matrix-matrix operations

       where  op( X ) is one of

          op( X ) = X   or   op( X ) = X',

       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.
```

BLAS



Miscellaneous Details

- LDA - Leading Dimension of “A” used to access subblocks of
- CBLAS additions `CblasRowMajor`, `CblasColMajor`

BLAS Example: DGEMM ($\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}$)

```
#include <iostream>
#include <cblas.h>
int main(int argc, char **argv) {
   int m = 5, k = 5, n = 5;
   double alpha = 1.0, beta = 0.0;
   double *A = new double[m*k];
   double *B = new double[k*n];
   double *C = new double[m*n];
   for (int i=0; i<(m*k); i++) A[i] = (double)(i+1);
   for (int i=0; i<(k*n); i++) B[i] = (double)(-i-1);
   for (int i=0; i<(m*n); i++) C[i] = 0.0;
   cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
   m, n, k, alpha, A, k, B, n, beta, C, n);
   ...
}
```

BLAS Example: DGEMM ($\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}$)

```
Matrix A : 5 by 5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25

Matrix B: 5 by 5
-1 -2 -3 -4 -5
-6 -7 -8 -9 -10
-11 -12 -13 -14 -15
-16 -17 -18 -19 -20
-21 -22 -23 -24 -25

Matrix C: 5 by 5
-215 -230 -245 -260 -275
-490 -530 -570 -610 -650
-765 -830 -895 -960 -1025
-1040 -1130 -1220 -1310 -1400
-1315 -1430 -1545 -1660 -1775
```

LAPACK

The Linear Algebra PACKage (LAPACK)

LAPACK contains a variety of subroutines for solving linear systems, matrix decompositions, and factorizations.

- Intermally uses BLAS calls
- Supports the same data types (single/double precision, real/complex and matrix structure types (symmetric, banded, etc.) as BLAS
- Three categories: auxiliary routines, computational routines, and driver routines
- C interface with prefix `LAPACKE_`
  - `http://www.netlib.org/lapack/lapacke.html`

The Linear Algebra PACKage (LAPACK)

Computational routines are designed to perform single, specific computational tasks:

- factorizations:

  $\boldsymbol{LU}$, $\boldsymbol{LL^T}$ /$\boldsymbol{LL^H}$, $\boldsymbol{LDL^T}$ /$\boldsymbol{LDL^H}$,

  $\boldsymbol{QR}$, $\boldsymbol{LQ}$, $\boldsymbol{QRZ}$ generalized $\boldsymbol{QR}$ and $\boldsymbol{RQ}$

- symmetric/Hermitian and nonsymmetric eigenvalue decompositions
- singular value decompositions
- generalized eigenvalue and singular value decompositions

LAPACK Example: DGESV (Solve $\mathbf{Ax} = \mathbf{b}$)

```
NAME
       DGESV - computes the solution to a real system of linear equations  A * X = B,

SYNOPSIS
       SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )

           INTEGER       INFO, LDA, LDB, N, NRHS

           INTEGER       IPIV( * )

           DOUBLE        PRECISION A( LDA, * ), B( LDB, * )

PURPOSE
       DGESV 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.
```

LAPACK Example: DGESV (Solve $\mathbf{Ax} = \mathbf{b}$)

```
#include <iostream>
#include <lapacke.h>
const int N=3, NRHS=2, LDA=N, LDB=N;
int main(int argc, char **argv) {
   int ipiv[N], info;
   double a[LDA*N] = {
      6.80, -2.11, 5.66,
      -6.05, -3.30, 5.36,
      -0.45, 2.58, -2.70
   };
   double b[LDB*NRHS] = {
      4.02, 6.19, -8.22,
      -1.56, 4.00, -8.67
   };
   info = LAPACKE_dgesv( LAPACK_COL_MAJOR, N, NRHS,
   a, LDA, ipiv, b, LDB );
   ...
}
```

LAPACK Example: DGESV (Solve $\mathbf{Ax} = \mathbf{b}$)

```
$ g++ -std=c++14 dgesv.cc -o dgesv -I${BLAS_INC} -L${BLAS_LIB} -lopenblas
$ ./dgesv
```

```
Solution ``x''
-0.0517981 -0.892398
 -0.819976 -0.736171
 1.30806   -0.121056

Details of LU factorization
6.8        -6.05      -0.45
0.832353   10.3957    -2.32544
-0.310294  -0.49802   1.28225

Pivot indices
1 3 3
```

What about non-dense matrices?

$$\begin{pmatrix}
-2 & 1 & & & & & & \\
1 & -2 & 4 & & & & & \\
 & 4 & -2 & 4 & & & & \\
 & & 4 & -2 & 4 & & & \\
 & & & 4 & -2 & 4 & & \\
 & & & & & \ddots & & \\
 & & & & & 4 & -2 & 1 \\
 & & & & & & 1 & -2
\end{pmatrix}$$

Types

- Banded: `DGBSV`
- Tri-Diagonal: `DGTSV`
- Symmetric Positive Definite: `DPOSV`

LAPACK Example: DGTSV (Solve $\mathbf{Ax} = \mathbf{b}$)

```
#include <iostream>
#include <lapacke.h>
const int N=5, NRHS=2;
int main(int argc, char **argv) {
   int ldb=N, info;
   double dl[N-1] = { 1, 4, 4, 1 };
   double d[N] = { -2, -2, -2, -2, -2 };
   double du[N-1] = {1, 4, 4, 1 };
   double b[N*NRHS] = {
      3, 5, 5, 5, 3,
      -1.56, 4.00, -8.67, 1.75, 2.86,
      9.81, -4.09, -4.57, -8.61, 8.99
   };
   info = LAPACKE_dgtsv(LAPACK_COL_MAJOR, N, NRHS,
   dl, d, du, b, ldb );
   ...
}
```

LAPACK Example: DGTSV (Solve $\mathbf{Ax} = \mathbf{b}$)

```
$ g++ -std=c++14 dgesv.cc -o dgtsv -I${BLAS_INC} -L${BLAS_LIB} -lopenblas
$ ./dgtsv
```

```
Solutions ''x''
 -0.93   0.29  -6.10
  1.14  -0.99  -2.39
  2.05   0.43  -0.69
  1.14  -0.96   0.90
 -0.93  -1.91  -4.05
```

Sparse Matrices

Sparse BLAS ?

Unfortunately there is not a mature, de facto standard sparse matrix BLAS library. Three potential options:

- "Official" Sparse BLAS: a reference implementation is not yet available `http://www.netlib.org/blas/blast-forum`
- NIST Sparse BLAS: An alternative BLAS system; a reference implementation is available `http://math.nist.gov/spblas`
- The Matrix Template Library: The C++ library mentioned above also provides support for sparse matrices `http://www.osl.iu.edu/research/mtl/intro.php3`

Some useful links

- `http://www.cs.colorado.edu/~jessup/lapack/`
- `https://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/mkl_lapack_examples/hh_goto.htm#dsyev_ex.c.htm`
- `http://web.cs.ucdavis.edu/~bai/publications/baidemmeletal06.pdf`

Conclusions

Conclusions

- Linear algebra pops up everywhere
- Statistics, data fitting, graph problems, PDE/coupled ODE solves, signal processing...
- Exploit structure in your matrices
- Chose best method based on system properties (condition number, sparsity, etc..)
- Many very highly tuned packages for any sort of problem that can be cast into matrices
- LAPACK, BLAS, etc..