HPC Linear Algebra

Chris Kauffman

Last Updated:
Tue Feb 24 02:00:19 PM EST 2026

Logistics

Assignments

- ▶ A1 grading complete
- ▶ Mini-Exam 1 grading in progress
- ▶ A2 posted

Today

- ▶ Complete Discussion of MPI Collective Communication
- ▶ Overview of some Linear Algebra Libraries

Hand-Coded Matrix Algebra

▶ Very common for students to
learn how to code up some
basic linear algebra routines

▶ In reality, prototype and
production code benefits from
use of mature libraries for these

▶ Existing libraries for Linear
Algebra already exist, are
reliable and fast, both
important in HPC / Parallel
Computing

```c
void matmult(
  int arows, int bcols, int midim,
  double A[][], // arows * midim
  double B[][], // midim * bcols
  double C[][]) // arows * bcols
{
  for (int i=0 ; i < arows ; ++i ){
    for (int j=0 ; j < bcols ; ++j ){
      C[i][j] = 0.0;
      for (int k=0 ; k < midim ; ++k ){
        C[i][j] += A[i][k] * B[k][j];
      }
    }
  }
}
// try dgemm() instead
```

BLAS: Basic Linear Algebra Subroutines

- ▶ Started in the 1970's and now WIDELY deployed
- ▶ Defines a set of numerical operations in 3 Levels
  1. Vector/Scalar operations (add constant onto all vector elements) and Vector/Vector operations (dot product)
  2. Matrix/Vector operations (mat-vec multiply)
  3. Matrix/Matrix operations (mat-mat multiply)
- ▶ Interestingly these are all mostly $O(N), O(N^2), O(N^3)$ operations respectively
- ▶ The names for the function suck and take significant study to understand and use effectively

  axpy()? ddot()? sgemm()? Are these rappers, hacker handles, or did someone just slap the keyboard randomly to name functions?
  According to legend, all the function names are 5 letters or less as this was the limit imposed by the Fortran compiler which originally compiled them.

4

BLAS Introductory Example

`dgemm()` : Multiply 2 double precision, general format matrices

```
dgemm(opa, opb,              // transpose A,B or not
      arows, bcols, midim,   // matrix dimensions
      alpha,                 // scaling factor for product
      A, lda, B, ldb,        // A and B matrix + cols
      beta, C, ldc)          // answer scaling + storage + dim
```

`C := alpha * opa( A )* opb( B ) + beta*C`

Super transparent, excellent software engineering...

▶ Targets Fortran77: different calling conventions than C
▶ Complex due to flexibility: 4 variants based on `opa,opb`

$$C \leftarrow \alpha AB + \beta C \qquad C \leftarrow \alpha A^T B + \beta C$$
$$C \leftarrow \alpha AB^T + \beta C \qquad C \leftarrow \alpha A^T B^T + \beta C$$

▶ Allows for scaling with `alpha,beta` but both often are 1
▶ Naming Convention: d ge mm ()
  ▶ d: double precision real
  ▶ ge: general matrix, not symmetric or banded
  ▶ mm: matrix multiply

C BLAS Example

- ▶ cblas are C language bindings to BLAS routines
- ▶ Slightly easier to understand, uses symbolic names for some (extra) arguments
- ▶ Accounts for C being Row-Major vs Fortran being Column-Major

```
// dgemm_example.c
// A : arows * midim matrix
// B : midim * bcols matrix
// B : arows * bcols matrix
// C <- A*B
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
            arows, bcols, midim,
            alpha,
            A, midim, B, bcols,
            beta, C, bcols);
```

LAPACK: Linear Algebra Package

- ▶ Basic Operations like Matrix Multiply are covered in BLAS
- ▶ Many Linear Algebra problems come up in HPC
  - ▶ Solve a Linear System: $Ax = b$, find x give A, b
  - ▶ Determine eigenvectors / eigenvalues for matrix A
  - ▶ Calculate Singular Value Decomposition on A
- ▶ LAPACK builds on BLAS to provide algorithms for all of these
- ▶ Has many of the same properties as BLAS
  - ▶ Netlib version Written in Fortran77 with bindings for C in LAPACKE
  - ▶ Naming conventions are difficult: `dgesv()`
    d: double precision real
    ge: general format matrix
    sv: "solve" linear system via a LUP decomposition
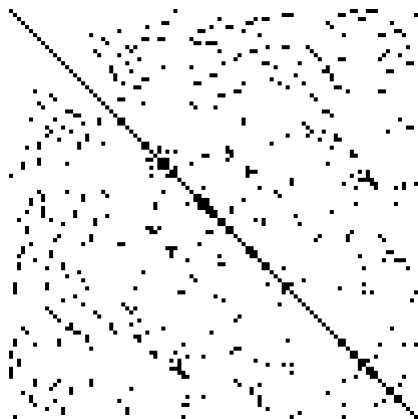- ▶ LAPACK / BLAS often packaged together in single libraries

Implementations of BLAS+LAPACK

Package	Notes
Netlib	The official LAPACK. (BSD/Open Source)
OpenBLAS	Open Source Implementation of BLAS/LAPACK routines
ATLAS	Automatically Tuned Linear Algebra Software (BSD/Open Source)
GSL	GNU Scientific Library (GNU / Open Source)
Intel MKL	Intel's Math routines for their x86 CPUs. (Freeware/Closed Source)
ARM PL	ARM Processor Performance Libraries (Freeware/Closed Source)
NVBLAS	NVIDIA BLAS, optimized for CUDA / GPU execution
cuSOLVER	NVIDIA LAPACK, optimized for CUDA / GPU execution
...	

▶ Note vendor implementations for specific processors / architectures: target efficient operation on these chips

▶ ATLAS is notable as on install, runs a series of benchmarks to set parameters in BLAS that give the best performance (Automatically Tuned)

Sparse Matrices

- Various scientific problems in HPC involve matrices with **many** Zero elements
- Referred to as **sparse matrices** especially when stored in data structures that reduce their size
- Contrast with **dense matrices** which we have assumed so far
- Example: LINKS matrix in Page Rank could benefit a lot from sparse storage
- BLAS / LAPACK deal with dense matrices, Sparse Matrices are their own beast

Example of a sparsity pattern in a large matrix: **black** indicates non-zero element, white is zero

Data Structures to Store Sparse Matrices

- Dense Matrices
    - Use $NROW \times NCOL$ space
    - Provide $O(1)$ lookup for element (i,j)
    - Easily provides $O(N)$ iteration through matrix elements
- Sparse Matrix formats
    - Use $O(NNZ)$ storage: NNZ is the **Number of NonZeros**
    - Provide worse than $O(1)$ lookup for element (i,j)
    - Store only elements that are nonzero, assume if an index is not present that it is zero
    - Try to provide $O(NNZ)$ iteration through matrix elements
- Storage savings for sparse formats can be significant when matrix is mostly zeros ($NNZ \ll NROW \times NCOL$)

Octave Example of Sparse Matrix Storage

- ▶ Octave is an open-source scientific computing environment, mostly compatible with Matlab
- ▶ Has built-in support for sparse matrices
- ▶ Uses the Compressed Sparse Column format (CSC) internally
- ▶ Makes it easy to show space savings

```
octave:2> A=[
[10   20    0    0    0    0]
[ 0   30    0   40    0    0]
[ 0    0   50   60   70    0]
[ 0    0    0    0    0   80]];

octave:3> As = sparse(A);

octave:4> B = [ A zeros(4,94);
                zeros(96, 100)];

octave:5> Bs=sparse(B);

octave:6> whos
Variables visible from the current scope:

variables in scope: top scope

  Attr Name       Size        Bytes  Class
  ==== ====       ====        =====  =====
       A          4x6           192  double
       As         4x6           184  double
       B          100x100     80000  double
       Bs         100x100       936  double
       ans        1x1             8  double

Total is 20049 elements using 81320 bytes
```

11

Coordinate Format (COO)

- ▶ Store (`row,col,val`) for all non-zero elements
- ▶ Values/Indices stored in separate arrays
- ▶ Justify operational complexities:
    1. Space requirement is 3*NNZ
    2. Finding element (`i,j`) is $O(\log(NNZ))$
    3. Transpose is $O(NNZ)$

```
     0    1    2    3    4    5
0  [10   20    0    0    0    0]    SAMPLE DENSE MATRIX
1  [ 0   30    0   40    0    0]
2  [ 0    0   50   60   70    0]
3  [ 0    0    0    0    0   80]

NNZ = 8, NROW=4, NCOL=6
                0  1  2  3  4  5  6  7
VALUES    = [ 10 20 30 40 50 60 70 80 ]    COO DATA ARRAYS
ROW_INDEX = [  0  0  1  1  2  2  2  3 ]
COL_INDEX = [  0  1  1  3  2  3  4  5 ]
```

Compressed Sparse Row Format (CSR)

▶ Save space by "compressing" rows : store only row start positions

▶ Length of Row I is `ROW_START[I+1]`-`ROW_START[I]`

▶ Justify operational complexities:
    1. Space requirement is `2*NNZ + NROW+1`
    2. Finding element (i,j) is close to $O(1)$
    3. Transpose is $O(NNZ)$

```
      0    1    2    3    4    5
0 [10   20    0    0    0    0]        SAMPLE DENSE MATRIX
1 [ 0   30    0   40    0    0]
2 [ 0    0   50   60   70    0]
3 [ 0    0    0    0    0   80]

NNZ = 8, NROW=4, NCOL=6
                0  1  2  3  4  5  6  7
VALUES    = [ 10 20 30 40 50 60 70 80 ]   CSR DATA ARRAYS
COL_INDEX = [  0  1  1  3  2  3  4  5 ]
ROW_START = [  0  2  4  7  8 ]
                            ^ Extra element = NNZ
```

Common Types of Sparse Matrices

- ▶ COO: Coordinate Sparse Matrix
- ▶ CSR: Compressed Sparse Row Matrix
- ▶ CSC: Compressed Sparse Column Matrix

Further reading

- ▶ Matlab Sparse Matrix Docs: the OG for high-level numerical computing, sparse matrices since 1992, supports intermingling dense/sparse matrices, many built-in algorithms to operate on sparse matrices

- ▶ SciPy Sparse Module: sparse matrices for use in python numerical computing, also features *Block Sparse Row (BSR)* format, algorithms to operate on sparse matrices

Algorithms for Sparse Matrices

- ▶ BLAS / LAPACK do NOT work for sparse matrices
- ▶ Sparse BLAS spec exists but fewer implementations than dense version
- ▶ Less standardization around sparse matrices but many libraries exist tailored to various tasks with various assumptions
- ▶ Factorization like in LAPACK require significant algorithm changes to work for sparse matrices

Related Materials

Stephen Boyd's EE364 Linear Algebra Oveview

https://stanford.edu/class/ee364b/lectures/
num-lin-alg-software.pdf

- ▶ Discusses many of the same items we talked about here
- ▶ Focus is on optimization problems like linear programming
- ▶ Very similar considerations

CSCI 5304 - Computational Aspects of Matrix Theory

- ▶ Great course on doing linear algebra for scientific problems
- ▶ Some coverage of BLAS/LAPACK and sparse matrices
- ▶ Often taught by Prof Saad who is a resident expert on all things Matrix/Sparse