CS 5220: Dense Linear Algebra

David Bindel

2017-10-17

Where we are

- This week: *dense* linear algebra
- Next week: *sparse* linear algebra

Numerical linear algebra in a nutshell

- Basic problems
  - Linear systems: $Ax = b$
  - Least squares: minimize $\|Ax - b\|_2^2$
  - Eigenvalues: $Ax = \lambda x$
- Basic paradigm: matrix factorization
  - $A = LU$, $A = LL^T$
  - $A = QR$
  - $A = V\Lambda V^{-1}$, $A = QTQ^T$
  - $A = U\Sigma V^T$
- Factorization $\equiv$ switch to basis that makes problem easy

Numerical linear algebra in a nutshell

Two flavors: dense and sparse

- Dense == common structures, no complicated indexing
  - General dense (all entries nonzero)
  - Banded (zero below/above some diagonal)
  - Symmetric/Hermitian
  - Standard, robust algorithms (LAPACK)
- Sparse == stuff not stored in dense form!
  - Maybe few nonzeros (e.g. compressed sparse row formats)
  - May be implicit (e.g. via finite differencing)
  - May be "dense", but with compact repn (e.g. via FFT)
  - Most algorithms are iterative; wider variety, more subtle
  - Build on dense ideas

History

BLAS 1 (1973–1977)

- Standard library of 15 ops (mostly) on vectors
  - Up to four versions of each: S/D/C/Z
  - Example: DAXPY
    - Double precision (real)
    - Computes $Ax + y$
  - Goals
    - Raise level of programming abstraction
    - Robust implementation (e.g. avoid over/underflow)
    - Portable interface, efficient machine-specific implementation
  - BLAS 1 == $O(n^1)$ ops on $O(n^1)$ data
  - Used in LINPACK (and EISPACK?)

History

BLAS 2 (1984–1986)

- Standard library of 25 ops (mostly) on matrix/vector pairs
  - Different data types and matrix types
  - Example: DGEMV
    - Double precision
    - GEneral matrix
    - Matrix-Vector product
- Goals
  - BLAS1 insufficient
  - BLAS2 for better vectorization (when vector machines roamed)
- BLAS2 == $O(n^2)$ ops on $O(n^2)$ data

History

BLAS 3 (1987–1988)

- Standard library of 9 ops (mostly) on matrix/matrix
  - Different data types and matrix types
  - Example: DGEMM
    - Double precision
    - GEneral matrix
    - Matrix-Matrix product
  - BLAS3 == $O(n^3)$ ops on $O(n^2)$ data
- Goals
  - Efficient cache utilization!

BLAS goes on

- `http://www.netlib.org/blas`
- CBLAS interface standardized
- Lots of implementations (MKL, Veclib, ATLAS, Goto, ...)
- Still new developments (XBLAS, tuning for GPUs, ...)

Why BLAS?

Consider Gaussian elimination.

LU for 2×2:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ c/a & 1 \end{bmatrix} \begin{bmatrix} a & b \\ 0 & d - bc/a \end{bmatrix}$$

Block elimination

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & B \\ 0 & D - CA^{-1}B \end{bmatrix}$$

Block LU

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{12} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{12}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}$$

Why BLAS?

Block LU

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{12} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{12}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}$$

Think of A as $k \times k$, k moderate:

```
[L11,U11] = small_lu(A);   % Small block LU
U12 = L11\B;               % Triangular solve
L12 = C/U11;               % "
S   = D-L21*U12;           % Rank k update
[L22,U22] = lu(S);         % Finish factoring
```

Three level-3 BLAS calls!

- Two triangular solves
- One rank-k update

LAPACK

LAPACK (1989–present): `http://www.netlib.org/lapack`

- Supercedes earlier LINPACK and EISPACK
- High performance through BLAS
  - Parallel to the extent BLAS are parallel (on SMP)
  - Linear systems and least squares are nearly 100% BLAS 3
  - Eigenproblems, SVD — only about 50% BLAS 3
- Careful error bounds on everything
- Lots of variants for different structures

ScaLAPACK

ScaLAPACK (1995–present):
`http://www.netlib.org/scalapack`

- MPI implementations
- Only a small subset of LAPACK functionality

PLASMA and MAGMA

PLASMA and MAGMA (2008–present):

- Parallel LA Software for Multicore Architectures
    - Target: Shared memory multiprocessors
    - Stacks on LAPACK/BLAS interfaces
    - Tile algorithms, tile data layout, dynamic scheduling
    - Other algorithmic ideas, too (randomization, etc)
- Matrix Algebra for GPU and Multicore Architectures
    - Target: CUDA, OpenCL, Xeon Phi
    - Still stacks (e.g. on CUDA BLAS)
    - Again: tile algorithms + data, dynamic scheduling
    - Mixed precision algorithms (+ iterative refinement)
- Dist memory: PaRSEC / DPLASMA