BLAS: Basic Linear Algebra Subroutines I

- Most numerical programs do similar operations
- 90% time is at 10% of the code
- If these 10% of the code is optimized, programs will be fast
- Frequently used subroutines should be available
- For numerical computations, common operations can be easily identified
- Example:

BLAS: Basic Linear Algebra Subroutines II

```
daxpy(n, alpha, p, inc, w, inc);
daxpy(n, malpha, q, inc, r, inc);

rtr = ddot(n, r, inc, r, inc);
rnorm = sqrt(rtr);
tnorm = sqrt(ddot(n, t, inc, t, inc));
```

- ddot: inner product
- daxpy: $ax + y$, x, b are vectors and a is a scalar
- If they are subroutines $\Rightarrow$ several for loops
- The first BLAS paper:

BLAS: Basic Linear Algebra Subroutines III

C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, Basic Linear Algebra Subprograms for FORTRAN usage, ACM Trans. Math. Soft., 5 (1979), pp. 308–323.

ACM Trans. Math. Soft.: a major journal on numerical software

- Become de facto standard for the elementary vector operations
  (`http://www.netlib.org/blas/`)
  Faster code than what you write

BLAS: Basic Linear Algebra Subroutines IV

- Netlib (`http://www.netlib.org`) is the largest site which contains freely available software, documents, and databases of interest to the numerical, scientific computing, and other communities (starting even before 1980).

  Interfaces from e-mail, ftp, gopher, x_based tool, to WWW
- Level 1 BLAS includes:

  dot product

  constant times a vector plus a vector

BLAS: Basic Linear Algebra Subroutines V

rotation (don't need to know what this is now)

copy vector x to vector y

swap vector x and vector y

length of a vector ($\sqrt{x_1^2 + \cdots + x_n^2}$)

sum of absolute values ($|x_1| + \cdots |x_n|$)

constant times a vector

index of element having maximum absolute value

- Programming convention

```
dw = ddot(n, dx, incx, dy, incy)
```

BLAS: Basic Linear Algebra Subroutines VI

$$w = \sum_{i=1}^{n} dx_{1+(i-1)incx} dy_{1+(i-1)incy}$$

Example:

```
dw = ddot(n, dx, 2, dy, 2)
```

$$w = dx_1 dy_1 + dx_3 dy_3 + \cdots$$

- `sdot` is single precision, `ddot` is for double precision
- $y = ax + y$

```
call daxpy(n, da, dx, incx, dy, incy)
```

BLAS: Basic Linear Algebra Subroutines VII

- To include which subroutines: difficult to make decisions
- C and Fortran interface

  C calls Fortran:

  ```
  rtr = ddot_(&n, r, &inc, r, &inc);
  ```

  C calls C:

  ```
  rtr = ddot(n, r, inc, r, inc);
  ```
- Traditionally they are written in Fortran

BLAS: Basic Linear Algebra Subroutines VIII

ddot_: calling Fortran subroutines (machine dependent)

&n: call by reference for Fortran subroutines

- Arrays: both call by reference

  C: start with 0, Fortran: with 1

  Should not cause problems here

- There is CBLAS interface
- For example, on a linux computer

BLAS: Basic Linear Algebra Subroutines IX

```
$ ls /usr/lib| grep blas
libblas
libblas.a
libblas.so
libblas.so.3
libblas.so.3gf
libcblas.so.3
libcblas.so.3gf
libf77blas.so.3
libf77blas.so.3gf
```

.a: static library, .so: dynamic library, .3 versus .3gf: g77 versus gfortran

Level 2 BLAS I

- The original BLAS contains only $O(n)$ operations

  That is, vector operations

  Matrix-vector product takes more time
- Level 2 BLAS involves $O(n^2)$ operations, n size of matrices

  J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, An extended set of FORTRAN Basic Linear Algebra Subprograms, ACM Trans. Math. Soft., 14 (1988), pp. 1–17.

Level 2 BLAS II

- Matrix-vector product

$$Ax : (Ax)_i = \sum_{j=1}^{n} A_{ij} x_j, i = 1, \ldots, m$$

  Like m inner products. However, inefficient if you use level 1 BLAS to implement this
- Scope of level 2 BLAS :
- Matrix-vector product

$$y = \alpha A x + \beta y, y = \alpha A^T x + \beta y, y = \alpha \bar{A}^T x + \beta y$$

Level 2 BLAS III

α, β are scalars, x, y vectors, A matrix

$x = Tx, x = T^T x, x = \bar{T}^T x$

x vector, T lower or upper triangular matrix

If $A = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$, the lower is $\begin{bmatrix} 2 & 0 \\ 4 & 5 \end{bmatrix}$

- Rank-one and rank-two updates

  $A = \alpha x y^T + A, H = \alpha x \bar{y}^T + \bar{\alpha} y \bar{x}^T + H$

  H is a Hermitian matrix ($H = \bar{H}^T$, symmetric for real numbers)

  rank: # of independent rows (columns) of a matrix

  column rank = row rank

Level 2 BLAS IV

$xy^T(\left[\begin{smallmatrix}1\\2\end{smallmatrix}\right]\left[\begin{smallmatrix}3 & 4\end{smallmatrix}\right] = \left[\begin{smallmatrix}3 & 4\\6 & 8\end{smallmatrix}\right])$ is a rank one matrix
n^2 operations
$xy^T + yx^T$ is a rank-two matrix

$$\begin{bmatrix}1\\2\end{bmatrix}[3\ 4] = \begin{bmatrix}3 & 4\\6 & 8\end{bmatrix}, \begin{bmatrix}3\\4\end{bmatrix}[1\ 2] = \begin{bmatrix}3 & 6\\4 & 8\end{bmatrix}$$

- Solution of triangular equations

Level 2 BLAS V

$$x = T^{-1}y$$

$$\begin{bmatrix} T_{11} & & & \\ T_{21} & T_{22} & & \\ \vdots & \vdots & \ddots & \\ T_{n1} & T_{n2} & \cdots & T_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

- The solution

$$\begin{aligned} x_1 &= y_1/T_{11} \\ x_2 &= (y_2 - T_{21}x_1)/T_{22} \\ x_3 &= (y_3 - T_{31}x_1 - T_{32}x_2)/T_{33} \end{aligned}$$

Level 2 BLAS VI

- Number of multiplications/divisions:

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

Level 3 BLAS I

- Things involve $O(n^3)$ operations
- Reference:
  J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling, A set of Level 3 Basic Linear Algebra Subprograms, ACM Trans. Math. Soft., 16 (1990), pp. 1–17.
- Matrix-matrix products

  $C = \alpha AB + \beta C$, $C = \alpha A^T B + \beta C$,

  $C = \alpha AB^T + \beta C$, $C = \alpha A^T B^T + \beta C$
- Rank-k and rank-2k updates

Level 3 BLAS II

- Multiplying a matrix by a triangular matrix

  $B = \alpha TB, \ldots$

- Solving triangular systems of equations with multiple right-hand side:

  $B = \alpha T^{-1}B, \ldots$

- Naming conversions: follows the conventions of the level 2

- No subroutines for solving general linear systems or eigenvalues

  In the package LAPACK described later

Block Algorithms I

- Let's test the matrix multiplication
- A C program:

```
#define n 2000
double a[n][n], b[n][n], c[n][n];

int main()
{
    int i, j, k;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++) {
            a[i][j]=1; b[i][j]=1;
```

Block Algorithms II

```
        }

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

- A Matlab program

Block Algorithms III

```
n = 2000;
A = randn(n,n); B = randn(n,n);
t = cputime; C = A*B; t = cputime -t
```

- Matlab is much faster than a code written by ourselves. Why ?
- Optimized BLAS: the use of memory hierarchies
- Data locality is exploited
- Use the highest level of memory as possible
- Block algorithms: transferring sub-matrices between different levels of storage

  localize operations to achieve good performance

Memory Hierarchy I

CPU

$\downarrow$

Registers

$\downarrow$

Cache

$\downarrow$

Main Memory

$\downarrow$

Secondary storage (Disk)

- $\uparrow$: increasing in speed
- $\downarrow$: increasing in capacity

Memory Management I

- Page fault: operand not available in main memory
  transported from secondary memory
  (usually) overwrites page least recently used
- I/O increases the total time
- An example: $C = AB + C$, $n = 1,024$
- Assumption: a page 65,536 doubles $=$ 64 columns
- 16 pages for each matrix
  48 pages for three matrices

Memory Management II

- Assumption: available memory 16 pages, matrices access: column oriented

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

column oriented: 1 3 2 4

row oriented: 1 2 3 4

- access each row of A: 16 page faults, $1024/64 = 16$
- Assumption: each time a continuous segment of data into one page
- Approach 1: inner product

Memory Management III

```
for i =1:n
  for j=1:n
    for k=1:n
      c(i,j) = a(i,k)*b(k,j)+c(i,j);
    end
  end
end
```

We use a matlab-like syntax here

- At each (i,j): each row a(i, 1:n) causes 16 page faults

Memory Management IV

Total: $1024^2 \times 16$ page faults

- at least 16 million page faults
- Approach 2:

```
for j =1:n
  for k=1:n
    for i=1:n
      c(i,j) = a(i,k)*b(k,j)+c(i,j);
    end
  end
end
```

Memory Management V

- For each j, access all columns of A

  A needs 16 pages, but B and C take spaces as well

  So A must be read for every j
- For each j, 16 page faults for A

  1024×16 page faults

  C, B : 16 page faults
- Approach 3: block algorithms ($\text{nb} = 256$)

Memory Management VI

```
for j =1:nb:n
  for k=1:nb:n
    for jj=j:j+nb-1
      for kk=k:k+nb-1
        c(:,jj) = a(:,kk)*b(kk,jj)+c(:,jj);
      end
    end
  end
end
```

Memory Management VII

$$
\begin{bmatrix} A_{11} & \cdots & A_{14} \\ & \vdots & \\ A_{41} & \cdots & A_{44} \end{bmatrix}
\begin{bmatrix} B_{11} & \cdots & B_{14} \\ & \vdots & \\ B_{41} & \cdots & B_{44} \end{bmatrix}
$$

$$
= \begin{bmatrix} A_{11}B_{11} + \cdots + A_{14}B_{41} & \cdots \\ & \vdots & \ddots \end{bmatrix}
$$

- Each block: 256×256

$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + \cdots + A_{14}B_{41} \\
C_{21} &= A_{21}B_{11} + \cdots + A_{24}B_{41} \\
C_{31} &= A_{31}B_{11} + \cdots + A_{34}B_{41} \\
C_{41} &= A_{41}B_{11} + \cdots + A_{44}B_{41}
\end{aligned}
$$

Memory Management VIII

- Use Approach 2 for $A_{:,1}B_{11}$
- $A(:,1)$: 256 columns, $1024 \times 256/65536 = 4$ pages. $A_{:,1}, \ldots, A_{:,4}$: $4 \times 4 = 16$ page faults in calculating $C_{:,1}$
- For A: 16×4 page faults
- B: 16 page faults, C: 16 page faults

LAPACK I

- LAPACK – Linear Algebra PACKage, based on BLAS
- Routines for solving

  Systems of linear equations

  Least-squares solutions of linear systems of equations

  Eigenvalue problems, and

  Singular value problems.
- Subroutines in LAPACK classified as three levels:

LAPACK II

- driver routines: each solves a complete problem, for example solving a system of linear equations
- computational routines: each performs a distinct computational task, for example an LU factorization
- auxiliary routines: subtasks of block algorithms, commonly required low-level computations, a few extensions to the BLAS
- Provide both single and double versions
- Naming: All driver and computational routines have names of the form XYYZZZ

LAPACK III

- X: data type, S: single, D: double, C: complex, Z: double complex
- YY, indicate the type of matrix, for example

`GB`	`general band`
`GE`	`general (i.e., unsymmetric, in some cases rectangular)`

Band matrix: a band of nonzeros along diagonals

$$\begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

LAPACK IV

- ZZZ indicate the computation performed, for example

`SV`	`simple driver of solving general linear systems`
`TRF`	`factorize`
`TRS`	`use the factorization to solve Ax = b by forward or backward substitution`
`CON`	`estimate the reciprocal of the condition number`

- `SGESV`: simple driver for single general linear systems

  `SGBSV`: simple driver for single general band linear systems

LAPACK V

- Now optimized BLAS and LAPACK available on nearly all platforms

Block Algorithms in LAPACK I

- From LAPACK manual Third edition; Table 3.7
  `http://www.netlib.org/lapack/lug`
- LU factorization DGETRF: $O(n^3)$
- Speed in megaflops (10^6 floating point operations per second)

Block Algorithms in LAPACK II

	No. of CPUs	Block size	n 100	1000
Dec Alpha Miata	1	28	172	370
Compaq AlphaServer DS-20	1	28	353	440
c IBM Power 3	1	32	278	551
IBM PowerPC	1	52	77	148
Intel Pentium II	1	40	132	250
Intel Pentium III	1	40	143	297
SGI Origin 2000	1	64	228	452
SGI Origin 2000	4	64	190	699
Sun Ultra 2	1	64	121	240
Sun Enterprise 450	1	64	163	334

Block Algorithms in LAPACK III

- 100 to 1000: number of operations 1000 times
- Block algorithms not very effective for small-sized problems
- Clock speed of Intel Pentium III: 550 MHz

ATLAS: Automatically Tuned Linear Algebra Software I

- Web page:
  http://math-atlas.sourceforge.net/
- Programs specially compiled for your architecture

  That is, things related to your CPU, size of cache, RAM, etc.

Homework I

- We would like to compare the time for multiplying two $5,000$ by $5,000$ matrices
- Directly using sources of blas

  http://www.netlib.org/blas/
- pre-built optimized blas (Intel MKL for Linux)

  http://software.intel.com/en-us/articles/intel-mkl/

  Use evaluation version
- ATLAS
- BLAS by Kazushige Goto

Homework II

http://www.tacc.utexas.edu/tacc-projects/gotoblas2/

See the NY Times article

http://www.nytimes.com/2005/11/28/technology/28super.html?pagewanted=all

This Goto Blas is not actively maintained now, so it's unclear if the code can be easily used or not.

- You can use BLAS or CBLAS
- Try to comment on the use of multi-core processors.