CS601: Software Development for Scientific Computing

Autumn 2022

Week5: Motifs – Matrix Computations with Dense Matrices

Last week..

- Demo of make program
- Motif – Matrix Computation with Dense Matrices
  - Matrix Representation (2D arrays on stack and heap)
  - Matrix storage format (row-major and column-major)
  - Visualizing performance gap with different layouts (demo)
  - Understanding the performance gap:
    - Memory hierarchy
    - Performance API (demo)

Matrix Multiplication

- Three fundamental ways to think of the computation
  1. Dot product

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \times \begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}1.5+2.7 & 1.6+2.8\\3.5+4.7 & 3.6+4.8\end{bmatrix}$$

  2. Linear combination of the columns of the left matrix

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \times \begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \left[5\begin{bmatrix}1\\3\end{bmatrix} + 7\begin{bmatrix}2\\4\end{bmatrix} \quad 6\begin{bmatrix}1\\3\end{bmatrix} + 8\begin{bmatrix}2\\4\end{bmatrix}\right]$$

  3. Sum of outer products

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \times \begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \left[\begin{bmatrix}1\\3\end{bmatrix}\begin{bmatrix}5 & 6\end{bmatrix} + \begin{bmatrix}2\\4\end{bmatrix}\begin{bmatrix}7 & 8\end{bmatrix}\right]$$

Dot Product

- Vector $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, Vector $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$ $\mathrm{x_i}, \mathrm{y_i} \in \mathbb{R}$
- $x^T = [x_1 \quad x_2 \quad .. \quad x_n]$
- Dot Product or Inner Product: $c = x^T y$ $\mathrm{x}^{\mathrm{T}} \in \mathbb{R}^{1\times n}, y \in \mathbb{R}^{n\times 1}, c\ is\ scalar$

$$[x_1 \quad x_2 \quad .. \quad x_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = [x_1 y_1 + x_2 y_2 + .. + x_n y_n]$$

- E.g. $[1 \quad 2 \quad 3] \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = [1 \times 4 + 2 \times 5 + 3 \times 6] = 32$

AXPY

- Computing the more common (a times x plus y): $y = y + ax$
- $\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} y \\ y_2 \\ \vdots \\ y_n \end{bmatrix} + a \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

```
..
for i=1 to n
   y[i] = y[i] +  a*x[i]
..
```

- Cost? n multiplications and n additions = **`2n`** `or` **`O(n)`**

Matrix Vector Product

- Computing Matrix-Vector product: $c = c + Ax, \; A \in \mathbb{R}^{m\times r}, x \in \mathbb{R}^{r\times 1}$

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} + \begin{bmatrix} a_{11} & a_{12} & .. & a_{1r} \\ a_{21} & a_{22} & .. & a_{2r} \\ & & \vdots & \\ a_{m1} & a_{m2} & .. & a_{mr} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_r \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} + \begin{bmatrix} a_{11}x_1 + & a_{12}x_2 + & .. & +a_{1r}x_r \\ a_{21}x_1 + & a_{22}x_2 + & .. & +a_{2r}x_r \\ & & \vdots & \\ a_{m1}x_1 + & a_{m2}x_2 + & .. & +a_{mr}x_r \end{bmatrix}$$

$\rightarrow r$, $\downarrow m$ $\qquad$ $\rightarrow 1$, $\downarrow m$

- Rewriting Matrix-Vector product using dot products:

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} + \begin{bmatrix} a_{11} & a_{12} & .. & a_{1r} \\ a_{21} & a_{22} & .. & a_{2r} \\ & & \vdots & \\ a_{m1} & a_{m2} & .. & a_{mr} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_r \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} + \begin{bmatrix} a_1^T x \\ a_2^T x \\ \vdots \\ a_m^T x \end{bmatrix}$$

- Cost? m rows involving dot products and having the form $c_i = c_i + x^T y$ (Per row cost = `2r` `(because` $a_i, x \in \mathbb{R}^r$`)`, Total cost = **`2mr`** `or` **`O(mr)`**)

Matrix-Matrix Product

- Computing Matrix-Matrix product $C = C + AB, A \in \mathbb{R}^{m \times r}, B \in \mathbb{R}^{r \times n}, C \in \mathbb{R}^{m \times n}$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ & \vdots & & \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix} = \underset{\downarrow m \;\; \rightarrow n}{\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ & \vdots & & \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix}} + \underset{\downarrow m \;\; \rightarrow r}{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1r} \\ a_{21} & a_{22} & \cdots & a_{2r} \\ & \vdots & & \\ a_{m1} & a_{m2} & \cdots & a_{mr} \end{bmatrix}} \underset{\downarrow r \;\; \rightarrow n}{\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ & \vdots & & \\ b_{r1} & b_{r2} & \cdots & b_{rn} \end{bmatrix}}$$

- Consider the AB part first.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1r} \\ a_{21} & a_{22} & \cdots & a_{2r} \\ & \vdots & & \\ a_{m1} & a_{m2} & \cdots & a_{mr} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ & \vdots & & \\ b_{r1} & b_{r2} & \cdots & b_{rn} \end{bmatrix}$$

Matrix-Matrix Product

$$\begin{array}{cc} A & B \\ \begin{bmatrix} a_{11} & a_{12} & .. & a_{1r} \\ a_{21} & a_{22} & .. & a_{2r} \\ & \vdots & & \\ a_{m1} & a_{m2} & .. & a_{mr} \end{bmatrix} & \begin{bmatrix} b_{11} & b_{12} & .. & b_{1n} \\ b_{21} & b_{22} & .. & b_{2n} \\ & \vdots & & \\ b_{r1} & b_{r2} & .. & b_{rn} \end{bmatrix} \end{array}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + .. + a_{1r}b_{r1} & . & . & a_{11}b_{1n} + a_{12}b_{2n} + .. + a_{1r}b_{rn} \\ . & & . & \\ a_{m1}b_{11} + a_{m2}b_{21} + .. + a_{mr}b_{r1} & . & . & a_{m1}b_{1n} + a_{m2}b_{2n} + .. + a_{mr}b_{rn} \end{bmatrix}$$

Notice that:

- subscript on a varies from 1 to m in a column (i.e. m rows exist)
- subscript on a varies from 1 to r in a row (i.e. r columns exist)

Suppose that we treat a_i as a vector of size r and there exist m vectors

$$= \begin{bmatrix} a_1^T b_1 & . & . & a_1^T b_n \\ . & . & & \\ a_m^T b_1 & . & . & a_m^T b_n \end{bmatrix}$$

$a_i^T \in \mathbb{R}^{1\times r}, b_j \in \mathbb{R}^{r\times 1}$

i ranges from 1 to m

j ranges from 1 to n

Matrix-Matrix Product using Dot Product Formulation

- Pseudocode - Matrix-Matrix product: $C = C + AB,\ A \in \mathbb{R}^{m \times r}, B \in \mathbb{R}^{r \times n}, C \in \mathbb{R}^{m \times n}$

```
..
for i=1 to m
    for j=1 to n
        //compute updates involving dot products
```

$$c_{ij} = c_{ij} + a_i^T b_j$$

Matrix-Matrix Product using Dot Product Formulation – Data Access

- Pseudocode - Matrix-Matrix product: $C = C + AB,\ A \in \mathbb{R}^{m\times r}, B \in \mathbb{R}^{r\times n}, C \in \mathbb{R}^{m\times n}$

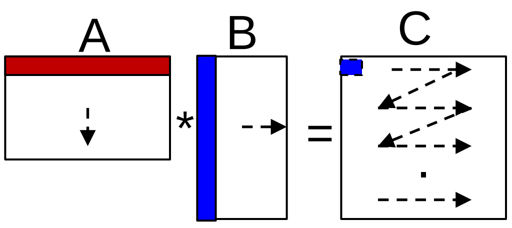
```
..
for i=1 to m
    for j=1 to n
        //compute updates involving dot products
```

$$c_{ij} = c_{ij} + a_i^T b_j$$

- Expanded:

```
..
for i=1 to m
    for j=1 to n
        for k=1 to r
```

$$c_{ij} = c_{ij} + a_{ik} b_{kj}$$



Elements of C matrix are computed from top to bottom, left to right. Per element computation, you need a row of A and a column of B.

Matrix-Matrix Product using Dot Product Formulation - Cost

- Pseudocode - Matrix-Matrix product: $C = C + AB,\ A \in \mathbb{R}^{m\times r}, B \in \mathbb{R}^{r\times n}, C \in \mathbb{R}^{m\times n}$

```
..
for i=1 to m
    for j=1 to n
        //compute updates involving dot products
```

$$c_{ij} = c_{ij} + a_i^T b_j$$

- Cost?
  - Per dot-product cost = `2r` $(a_i, b_j \in \mathbb{R}^r)$ Total cost = **`2mnr`** `or` **`O(mnr)`**

Common Computational Patterns

Some patterns that we see while doing Matrix-Matrix product:

1. Dot Product or Inner Product: $x^T y$ ← Slide 27, Method 1
2. Scalar **a** times **x** **p**lus **y**: y=y+ax OR saxpy
   Slide 27, Method 2
   - Scalar times **x**: αx
3. Matrix times x plus y: y=y+Ax ← Slide 27, Method 1
   - generalized axpy OR gaxpy
4. Outer product: $C=C+xy^T$ ← Slide 27, Method 3
5. Matrix times Matrix plus Matrix
   - GEMM or generalized matrix multiplication

What is dense linear algebra?

- Not just matrix multiplication (matmul!)
- Solving system of equations: $Ax=b$ (e.g. using Gaussian Elimination)
- Computing Least Squares: choose x to minimize $||Ax-b||_2$
  - Overdetermined or underdetermined; Unconstrained, constrained, or weighted
- Computing Eigenvalues and Eigenvectors of Matrices (Symmetric and Unsymmetric)
    - Standard ($Ax = \lambda x$), Generalized ($Ax=\lambda Bx$)
- Representing Different matrix structures
  - Real, complex; Symmetric, Hermitian, positive definite; dense, triangular, banded …
- Capturing level of detail
  - error bounds, extra-precision, other options

slide source: www.cs.berkeley.edu/~demmel (CS267)

Linear Algebra Software

- **Goals**: programmer productivity, readability, robustness, portability, machine efficiency
- Examples
  - EISPACK (for computing eigenvalue problems)
  - BLAS
  - LAPACK
  - Many more..

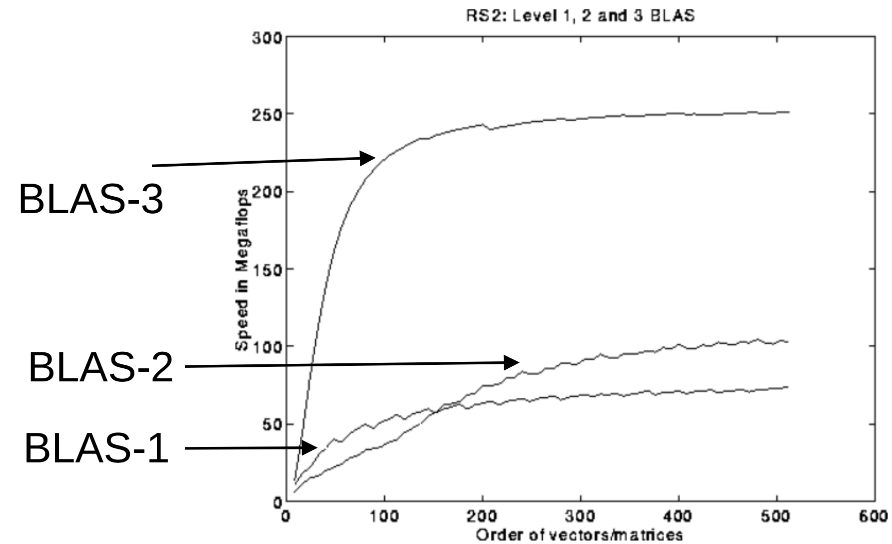
BLAS – Basic Linear Algebra Subroutines

- Level-1 or BLAS-1 (46 operations, routines operating on vectors mostly)
  - axpy, dot product, rotation, scale, etc.
  - 4 versions each: **S**ingle-precision, **d**ouble-precision, **c**omplex, complex-double (**z**)
  - E.g. saxpy, daxpy, caxpy etc.
  - **Do O(n) operations on O(n) data.**
- Level-2 or BLAS-2 (25 operations, routines operating on matrix-vectors mostly)
  - E.g. GEMV ($\alpha A.x + \beta y$), GER (Rank-1 update $A = A + y.x^T$), Triangular solve ($y = T.x, T\ is\ a\ triangular\ matrix$) etc.
  - 4 versions each, **do $O(n^2)$ operations on $O(n^2)$ data.**

BLAS – Basic Linear Algebra Subroutines

- Level-3 or BLAS-3 (9 basic operations, routines operating on matrix-matrix mostly)
  - GEMM ($C = \alpha A.B + \beta C$),
  - Multiple triangular solve ($Y = TX$, T is triangular, X is rectangular)
  - **Do O(n³) operations on O(n²) data.**

- *Why categorize as BLAS-1, BLAS-2, BLAS-3?*
  - *Performance*



source: http://people.eecs.berkeley.edu/~demmel/cs267/lecture02.html

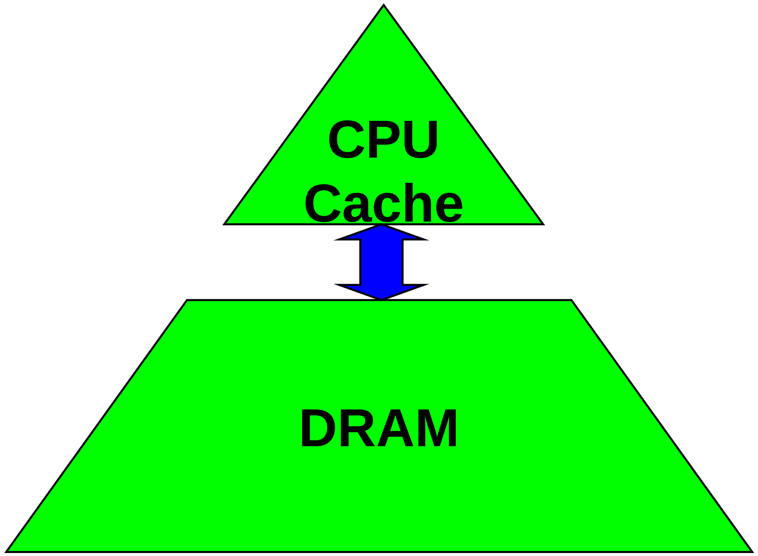
LAPACK – Linear Algebra Package

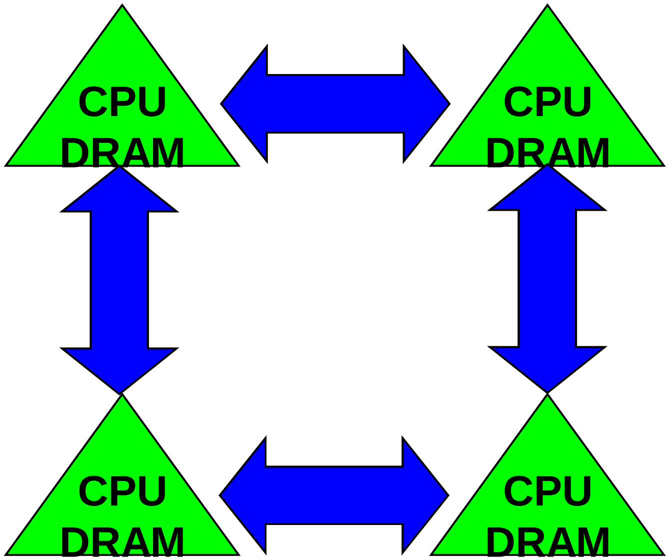
- LAPACK – uses BLAS-3 (1989 – now)
  - Ex: Obvious way to express Gaussian Elimination (GE) is adding multiples of one row to other rows – BLAS-1
    - How do we reorganize GE to use BLAS-3 ?
  - Contents of LAPACK (summary)
    - Algorithms that are (nearly) 100% BLAS-3
      - Linear Systems, Least Squares
    - Algorithms that are only ≈50% BLAS-3
      - Eigenproblems, Singular Value Decomposition (SVD)
    - Generalized problems (eg Ax = l Bx)
    - Error bounds for everything
    - Lots of variants depending on A's structure (banded, $A=A^T$, etc.)
  - How much code? (Release 3.9.0, Nov 2019) (www.netlib.org/lapack)
    - Source: 1982 routines, 827K LOC, Testing: 1210 routines, 545K LOC

Costs Involved

Algorithms have two costs:

1. Arithmetic (FLOPS)
2. Communication: moving data between
   - levels of a memory hierarchy (sequential case)
   - processors over a network (parallel case).





Computational Intensity

- Connection between computation and communication cost
- Average number of operations performed per data element (word) read/written from slow memory
  - E.g. Read/written m words from memory. Perform f operations on m words.
  - Computational Intensity q = f/m (*flops per word*).
- Goal: we want to *maximize* the computational intensity
  - We want to minimize words moved (read/written)
  - We want to minimize messages sent

What is the computational intensity, q, for:
axpy?
***M**atrix-**V**ector product? (e.g. GEMV)*
***M**atrix-**M**atrix product? (e.g. GEMM)*

Computational Intensity - axpy

Note: a slightly changed variant of axpy. There are n scalars (x_i) here.

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} + [x_1 \quad x_2 \quad \cdots \quad x_n]^{\mathsf{T}} \;.\!* \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} + \begin{bmatrix} x_1 \times y_1 \\ x_2 \times y_2 \\ \vdots \\ x_n \times y_n \end{bmatrix}$$

. indicates component-wise multiplication*

```
Read(x) //read x from slow memory
Read(y) //read y from slow memory
Read(c) //read c from slow memory
for i=1 to n
   c[i] = c[i] +  x[i]*y[i]  //do arithmetic on data read
Write(c) //write c back to slow memory
```

- Number of memory operations = 4n (assuming one word of storage for each component (x_i, y_i, c_i) of vectors x, y, c resp.)
- Number of arithmetic operations = 2n (one addition and one multiplication per row.)
- **q=2n/4n = 1/2**

Computational Intensity – matrix-vector

- Assume m=r=n =n

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} + \begin{bmatrix} a_{11} & a_{12} & .. & a_{1r} \\ a_{21} & a_{22} & .. & a_{2r} \\ & \vdots & & \\ a_{m1} & a_{m2} & .. & a_{mr} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_r \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} + \begin{bmatrix} a_{11}x_1 + & a_{12}x_2 + & .. & +a_{1r}x_r \\ a_{21}x_1 + & a_{22}x_2 + & .. & +a_{2r}x_r \\ & \vdots & & \\ a_{m1}x_1 + & a_{m2}x_2 + & .. & +a_{mr}x_r \end{bmatrix}$$

- Number of memory operations = $n^2 + 3n$ = $n^2 + O(n)$
- Number of arithmetic operations = $2n^2$
- $\boldsymbol{q \approx 2n^2/n^2 \ = \ 2}$