Introduction to DENSE linear algebra

Previously we considered linear algebra operations that involved sparse matices and (dense) vectors. We will move into a discussion of operations involving DENSE matrices next!

☆ What are examples of such operations ?

→ "DENSE" variants of operations we saw before

→ Matrix-vector products

$$y \leftarrow Ax$$

where A is DENSE

→ Triangular solves (upper or lower) where the matrix is dense triangular

e.g. Solve $Lx = b$ or $Ux = b$

L: lower tri

U: upper tri

→ Some new operations involving Dense Matrices and Vectors (still categorized BLAS Level 2)

→ Rank-1 Update of a general matrix

$$\underset{\text{DENSE}}{A} \longleftarrow A + \underbrace{u \cdot v^T}_{\text{vectors}}$$

→ Rank-1 Update of a symmetric matrix

$$\underset{\text{DENSE symmetric matrix}}{S} \longleftarrow S + \underbrace{u \cdot u^T}_{\text{product is still symmetric!}}$$

→ Matrix-vector multiply with symmetric matrix (saving storage...)

$$y \longleftarrow S \cdot x$$

symmetric matric, stored in space-saving "packed" format

→ "NEW" operations involving multiple dense matrices

⇒ These are the most interesting / consequential for parallel computation!

They generally fall in 2 categories:

A. BLAS Level 3 routines (the "simplest" category)

→ The "GEMM" operation

↳ Prime target for Tensor-based GPUS

Matrix-Matrix product

$$C \leftarrow \underbrace{\alpha}_{\text{scalars}} \cdot A \cdot B + \beta C$$

scalars (α, β) DENSE Matrices (A, B, C)

Variants:

Transposition $C \leftarrow \alpha \cdot A^T B^T + \beta C$

Triangular factor $B \leftarrow \alpha L \cdot B$

↳ triangular!

Symmetric factor $C \leftarrow \alpha \underbrace{A \cdot B}_{} + \beta C$

either A or B is symmetric

→ Rank-k updates (symmetric only!)

$$\underbrace{C \leftarrow \alpha A \cdot A^T + \beta C}_{C \text{ is symmetric}}$$

(more variants in LAPACK - - will see later)

→ Triangular matrix equations

$$L X = B$$

X, B: dense $N \times M$ matrices

L: lower triangular, $N \times N$

⇒ essentially, a back subsitution (or forward) with multiple right hand sides.

If $B = \underbrace{[\, b_1 \,|\, b_2 \,|\, b_3 \,|\, \dots \,|\, b_M \,]}_{M \text{ columns}}$

and
$$\left.\begin{array}{l} L x_1 = b_1 \\ \vdots \\ L x_M = b_M \end{array}\right\} \Rightarrow \text{solve } M \text{ forward substitutions..}$$

then $X = [\, x_1 \,|\, x_2 \,|\, \dots \,|\, x_M \,]$

Variants: $UX = B$, $XL = B$, $L^T X = B$ ...

B. Factorization / Solve routines from LAPACK

(The part of the LAPACK library that is included / optimized in MKL also includes routines for <u>least-squares</u> and <u>eigenanalysis</u> which we will skip for now)

<u>Examples</u>

→ PLU factorization

$$A = P \cdot L \cdot U$$

- U: Upper triangular, dense
- L: Lower triangular, dense
- P: "permutation" matrix (Sparse, special encoding)

LAPACK provides routines to:

→ Compute the factors of this decomposition

→ Solve a system $Ax = b$ with the previously computed factorization (or $AX = B$, with X/B being matrices)

→ Compute the matrix inverse A^{-1} (rarely used)

⇒ Factorization of symmetric matrices

$$S = L \cdot L^T$$ ("Cholesky" decomposition, if S is positive-definite)

$$S = L \cdot D \cdot L^T$$ ("LDL" decomposition, general symmetric S)

⇒ Triangular matrix solve (with "packed" storage)

$$LX = B$$

↳ triangular matrix, "packed" storage (we'll see details...)

There are many more operations involving dense matrices, but we'll focus on the ones above, since

→ They are available in MKL

→ They are the most relevant for parallel programming design

→ They are the most frequently used ones.

Why are dense algebra routines so interesting for parallel computing?

⇒ Highly useful and impactful

⇒ Prime example of a program that *might* be compute bound, e.g. GEMM on NxN matrices

$$C \leftarrow A \cdot B$$

Theory suggests: $C_{ij} = \sum_{k=1}^{N} A_{ik} B_{kj}$

Pseudocode

$C \leftarrow 0$

```
for i = 1...N
    for j = 1...N
        for k = 1...N
            C_ij += A_ik · B_jk
```

$O(N^2)$ storage
$O(N^3)$ computation
⇒ CACHE-permitting this could be compute bound!

⇒ Performant implementations need to be conscious of:

* Storage layout
* SIMD/Vectorization
* Cache efficiency
* Instruction latency & registers (NEW!)

Storage formats

⇒ Full (non-symmetric, non-triangular matrices)

Column Major or Row major

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & & \ddots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{bmatrix}$$

Row major → $[a_{11}, a_{12}, \ldots, a_{1N}, a_{21}, a_{22} \ldots, a_{2N}, a_{M1} \ldots a_{MN}]$

Col major → $[a_{11}, a_{21} \ldots a_{M1}, a_{12}, a_{22} \ldots a_{M2}, a_{1N} \ldots a_{MN}]$

⇒ Triangular / Symmetric

→ Various "paked" formats, e.g.