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

* What are examples of such operations?

→ "\underline{DENSE}" variants of operations we saw before

→ Matrix-vector products

\[
\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}
\]
where \( \mathbf{A} \) is \underline{DENSE}

→ Triangular solves (upper or lower)
where the matrix is \underline{dense} triangular

\[
e.g. \text{ Solve } \mathbf{L}\mathbf{x} = \mathbf{b} \text{ or } \mathbf{U}\mathbf{x} = \mathbf{b}
\]
\[
\uparrow \quad \uparrow
\]
\text{lower tri} \quad \text{upper tri}
Some new operations involving Dense Matrices and Vectors (still categorized BLAS Level 2)

→ Rank-1 Update of a general matrix

\[ A \leftarrow A + u \cdot v^T \]

\[ \uparrow \quad \text{DENSE} \quad \uparrow \quad \text{vectors} \]

→ Rank-1 Update of a symmetric matrix

\[ S \leftarrow S + u \cdot u^T \]

\[ \uparrow \quad \text{DENSE symmetric matrix} \quad \rightarrow \quad \text{product is still symmetric!} \]

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

\[ y \leftarrow S \cdot x \]

\[ \uparrow \quad \text{symmetric matrix, stored in space-saving "packed" format} \]
"NEW" operations involving multiple dense matrices

These are the most interesting / consequential for parallel computation!

They generally fall in 2 categories:

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

2. The "GEMM" operation

Matrix-Matrix product

\[ C \leftarrow \alpha \cdot A \cdot B + \beta \cdot C \]

 Scalars Dense Matrices

Variants:

Transposition \[ C \leftarrow \alpha \cdot A^T \cdot B^T + \beta \cdot C \text{ } \]

Triangular factor \[ B \leftarrow \alpha \cdot L \cdot B \text{ } \]

Symmetric factor \[ C \leftarrow \alpha \cdot A \cdot B + \beta \cdot C \text{ } \]

either A or B is symmetric
→ Rank-k updates (symmetric only!)

\[ C \leftarrow \alpha A \cdot A^T + \beta C \]

\[ C \text{ is symmetric} \]

(more variants in LAPACK — will see later)

→ Triangular matrix equations
dense \( N \times M \) matrices

\[ L \cdot X = B \]

\[ L \text{ lower triangular, } N \times N \]

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

If \( B = \begin{bmatrix} b_1 & b_2 & b_3 & \cdots & b_M \end{bmatrix} \)

\( M \) columns

and \( L \cdot x_1 = b_1 \)

\[ \vdots \]

\( L \cdot x_M = b_M \) \( \Rightarrow \) solve \( M \) forward substitutions...

then \( X = [x_1 \mid x_2 \mid \cdots \mid x_M] \)

Variants: \( U \cdot X = B \), \( X \cdot L = B \), \( L^T \cdot X = B \) etc.
B. Factorization/Solve routines from LAPACK

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

Examples

- PLU factorization

\[ A = P \cdot L \cdot U \]

- Upper triangular, dense
- Lower triangular, dense
- "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)

\[ L \cdot X = B \]

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 N×N 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 \ldots N
  for j = 1 \ldots N
    for k = 1 \ldots N
      C_{ij} += A_{ik} \cdot B_{kj}
```

- Performant implementations need to be conscious of:
  * Storage layout
  * SIMD/Vectorization
  * Cache efficiency
  * Instruction latency & registers (NEW!)

\(O(N^3)\) storage
\(O(N^3)\) computation
Rightarrow CACHE-permitting this could be compute bound!
Storage formats

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

  Column Major or Row major

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

  Row major → [\(a_{11}, a_{12}, \ldots, a_{1N}, a_{21}, a_{22}, \ldots, a_{2N}, a_{31}, \ldots a_{MN}\)]

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

- Triangular/Symmetric

- Various "packed" formats, e.g.

  ![Diagram of packed format](flip this way)