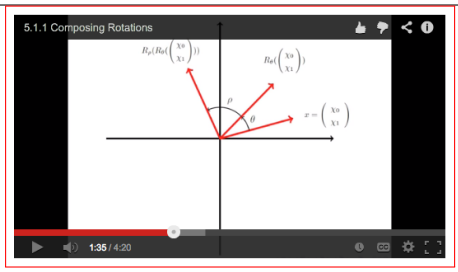
Week 5

Matrix-Matrix Multiplication

5.1 Opening Remarks

5.1.1 Composing Rotations



☛ View at edX

Homework 5.1.1.1 Which of the following statements are *true*:

- $\begin{pmatrix} \cos(\rho+\sigma+\tau) \\ \sin(\rho+\sigma+\tau) \end{pmatrix} = \left(\begin{array}{c|c} \cos(\tau) & -\sin(\tau) \\ \hline \sin(\tau) & \cos(\tau) \end{array}\right) \begin{pmatrix} \cos(\rho+\sigma) \\ \sin(\rho+\sigma) \end{pmatrix}$

  True/False

- $\begin{pmatrix} \cos(\rho+\sigma+\tau) \\ \sin(\rho+\sigma+\tau) \end{pmatrix} = \left(\begin{array}{c|c} \cos(\tau) & -\sin(\tau) \\ \hline \sin(\tau) & \cos(\tau) \end{array}\right) \begin{pmatrix} \cos\rho\cos\sigma-\sin\rho\sin\sigma \\ \sin\rho\cos\sigma+\cos\rho\sin\sigma. \end{pmatrix}$

  True/False

- $\begin{array}{rcl} \cos(\rho+\sigma+\tau) & = & \cos(\tau)(\cos\rho\cos\sigma-\sin\rho\sin\sigma)-\sin(\tau)(\sin\rho\cos\sigma+\cos\rho\sin\sigma) \\ \sin(\rho+\sigma+\tau) & = & \sin(\tau)(\cos\rho\cos\sigma-\sin\rho\sin\sigma)+\cos(\tau)(\sin\rho\cos\sigma+\cos\rho\sin\sigma) \end{array}$

  True/False

5.1.2 Outline

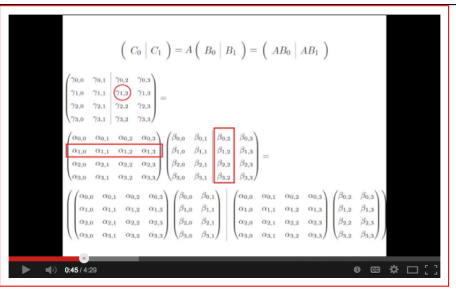
5.1.3 What You Will Learn

Upon completion of this unit, you should be able to

- **Recognize that matrix-matrix multiplication is not commutative.**
- Relate composing rotations to matrix-matrix multiplication.
- Fluently compute a matrix-matrix multiplication.
- Perform matrix-matrix multiplication with partitioned matrices.
- Identify, apply, and prove properties of matrix-matrix multiplication, such as $(AB)^T = B^T A^T$.
- Exploit special structure of matrices to perform matrix-matrix multiplication with special matrices, such as identity, triangular, and diagonal matrices.
- Identify whether or not matrix-matrix multiplication preserves special properties in matrices, such as symmetric and triangular structure.
- Express a matrix-matrix multiplication in terms of matrix-vector multiplications, row vector times matrix multiplications, and rank-1 updates.
- Appreciate how partitioned matrix-matrix multiplication enables high performance. (Optional, as part of the enrichment.)

5.2 Observations

5.2.1 Partitioned Matrix-Matrix Multiplication



Theorem 5.1 *Let $C \in \mathbb{R}^{m\times n}$, $A \in \mathbb{R}^{m\times k}$, and $B \in \mathbb{R}^{k\times n}$. Let*

- $m = m_0 + m_1 + \cdots m_{M-1}$, $m_i \geq 0$ *for* $i = 0, \ldots, M-1$;
- $n = n_0 + n_1 + \cdots n_{N-1}$, $n_j \geq 0$ *for* $j = 0, \ldots, N-1$; *and*
- $k = k_0 + k_1 + \cdots k_{K-1}$, $k_p \geq 0$ *for* $p = 0, \ldots, K-1$.

Partition

$$C = \left(\begin{array}{c|c|c|c} C_{0,0} & C_{0,1} & \cdots & C_{0,N-1} \\ \hline C_{1,0} & C_{1,1} & \cdots & C_{1,N-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline C_{M-1,0} & C_{M-1,1} & \cdots & C_{M-1,N-1} \end{array}\right), A = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,K-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,K-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,K-1} \end{array}\right),$$

$$\text{and } B = \left(\begin{array}{c|c|c|c} B_{0,0} & B_{0,1} & \cdots & B_{0,N-1} \\ \hline B_{1,0} & B_{1,1} & \cdots & B_{1,N-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline B_{K-1,0} & B_{K-1,1} & \cdots & B_{K-1,N-1} \end{array}\right),$$

with $C_{i,j} \in \mathbb{R}^{m_i\times n_j}$, $A_{i,p} \in \mathbb{R}^{m_i\times k_p}$, and $B_{p,j} \in \mathbb{R}^{k_p\times n_j}$. Then $C_{i,j} = \sum_{p=0}^{K-1} A_{i,p}B_{p,j}$.

If one partitions matrices C, A, and B into blocks, **and** one makes sure the dimensions match up, **then** blocked matrix-matrix multiplication proceeds exactly as does a regular matrix-matrix multiplication **except** that individual multiplications of scalars commute while (in general) individual multiplications with matrix blocks (submatrices) do not.

Example 5.2 Consider

$$A=\left(\begin{array}{cc|cc} -1 & 2 & 4 & 1 \\ 1 & 0 & -1 & -2 \\ 2 & -1 & 3 & 1 \\ 1 & 2 & 3 & 4 \end{array}\right), B=\left(\begin{array}{ccc} -2 & 2 & -3 \\ 0 & 1 & -1 \\ \hline -2 & -1 & 0 \\ 4 & 0 & 1 \end{array}\right), \text{ and } \quad AB=\left(\begin{array}{ccc} -2 & -4 & 2 \\ -8 & 3 & -5 \\ -6 & 0 & -4 \\ 8 & 1 & -1 \end{array}\right):$$

If

$$A_0=\left(\begin{array}{cc} -1 & 2 \\ 1 & 0 \\ 2 & -1 \\ 1 & 2 \end{array}\right), A_1=\left(\begin{array}{cc} 4 & 1 \\ -1 & -2 \\ 3 & 1 \\ 3 & 4 \end{array}\right), B_0\left(\begin{array}{ccc} -2 & 2 & -3 \\ 0 & 1 & -1 \end{array}\right), \text{ and } B_1=\left(\begin{array}{ccc} -2 & -1 & 0 \\ 4 & 0 & 1 \end{array}\right).$$

Then

$$AB=\left(\begin{array}{cc} A_0 & A_1 \end{array}\right)\left(\begin{array}{c} B_0 \\ B_1 \end{array}\right)=A_0B_0+A_1B_1:$$

$$\underbrace{\left(\begin{array}{cc|cc} -1 & 2 & 4 & 1 \\ 1 & 0 & -1 & -2 \\ 2 & -1 & 3 & 1 \\ 1 & 2 & 3 & 4 \end{array}\right)}_{A} \underbrace{\left(\begin{array}{ccc} -2 & 2 & -3 \\ 0 & 1 & -1 \\ \hline -2 & -1 & 0 \\ 4 & 0 & 1 \end{array}\right)}_{B}$$

$$= \underbrace{\left(\begin{array}{cc} -1 & 2 \\ 1 & 0 \\ 2 & -1 \\ 1 & 2 \end{array}\right)}_{A_0} \underbrace{\left(\begin{array}{ccc} -2 & 2 & -3 \\ 0 & 1 & -1 \end{array}\right)}_{B_0} + \underbrace{\left(\begin{array}{cc} 4 & 1 \\ -1 & -2 \\ 3 & 1 \\ 3 & 4 \end{array}\right)}_{A_1} \underbrace{\left(\begin{array}{ccc} -2 & -1 & 0 \\ 4 & 0 & 1 \end{array}\right)}_{B_1}$$

$$= \underbrace{\left(\begin{array}{ccc} 2 & 0 & 1 \\ -2 & 2 & -3 \\ -4 & 3 & -5 \\ -2 & 4 & -5 \end{array}\right)}_{A_0B_0} + \underbrace{\left(\begin{array}{ccc} -4 & -4 & 1 \\ -6 & 1 & -2 \\ -2 & -3 & 1 \\ 10 & -3 & 4 \end{array}\right)}_{A_1B_1} = \underbrace{\left(\begin{array}{ccc} -2 & -4 & 2 \\ -8 & 3 & -5 \\ -6 & 0 & -4 \\ 8 & 1 & -1 \end{array}\right)}_{AB}.$$

5.2.2 Properties

No video for this unit.

Is matrix-matrix multiplication associative?

Homework 5.2.2.1 Let $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 0 & 2 & -1 \\ 1 & 1 & 0 \end{pmatrix}$, and $C = \begin{pmatrix} 0 & 1 \\ 1 & 2 \\ 1 & -1 \end{pmatrix}$. Compute

- $AB =$
- $(AB)C =$
- $BC =$
- $A(BC) =$

Homework 5.2.2.2 Let $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times k}$, and $C \in \mathbb{R}^{k\times l}$. $(AB)C = A(BC)$.
Always/Sometimes/Never

If you conclude that $(AB)C = A(BC)$, then we can simply write ABC since lack of parenthesis does not cause confusion about the order in which the multiplication needs to be performed.

In a previous week, we argued that $e_i^T(Ae_j)$ equals $\alpha_{i,j}$, the (i, j) element of A. We can now write that as $\alpha_{i,j} = e_i^T Ae_j$, since we can drop parentheses.

Is matrix-matrix multiplication distributive?

Homework 5.2.2.3 Let $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 2 & -1 \\ 1 & 0 \end{pmatrix}$, and $C = \begin{pmatrix} -1 & 1 \\ 0 & 1 \end{pmatrix}$. Compute

- $A(B+C) =$.
- $AB + AC =$.
- $(A+B)C =$.
- $AC + BC =$.

Homework 5.2.2.4 Let $A \in \mathbb{R}^{m\times k}$, $B \in \mathbb{R}^{k\times n}$, and $C \in \mathbb{R}^{k\times n}$. $A(B+C) = AB+AC$.
Always/Sometimes/Never

Homework 5.2.2.5 If $A \in \mathbb{R}^{m\times k}$, $B \in \mathbb{R}^{m\times k}$, and $C \in \mathbb{R}^{k\times n}$, then $(A+B)C = AC+BC$.
True/False

5.2.3 Transposing a Product of Matrices

No video for this unit.

Homework 5.2.3.1 Let $A = \begin{pmatrix} 2 & 0 & 1 \\ -1 & 1 & 0 \\ 1 & 3 & 1 \\ -1 & 1 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 2 & 1 & 2 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$. Compute

- $A^T A =$
- $AA^T =$
- $(AB)^T =$
- $A^T B^T =$
- $B^T A^T =$

Homework 5.2.3.2 Let $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{k\times n}$. $(AB)^T = B^T A^T$.
Always/Sometimes/Never

Homework 5.2.3.3 Let A, B, and C be conformal matrices so that ABC is well-defined. Then $(ABC)^T = C^T B^T A^T$.
Always/Sometimes/Never

5.2.4 Matrix-Matrix Multiplication with Special Matrices

No video for this unit.

Multiplication with an identity matrix

Homework 5.2.4.1 Compute

- $\begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} =$
- $\begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} =$
- $\begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} =$
- $\begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} =$
- $\begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \\ -1 & 3 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} =$

Homework 5.2.4.2 Compute

- $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} =$
- $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -2 \\ 0 \\ 3 \end{pmatrix} =$
- $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ -1 \end{pmatrix} =$
- $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \\ -1 & 3 & -1 \end{pmatrix} =$

Homework 5.2.4.3 Let $A \in \mathbb{R}^{m \times n}$ and let I denote the identity matrix of appropriate size. $AI = IA = A$.

Always/Sometimes/Never

Multiplication with a diagonal matrix

Homework 5.2.4.4 Compute

- $\begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} =$
- $\begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} =$
- $\begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ -3 \end{pmatrix} =$
- $\begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -3 \end{pmatrix} =$

Homework 5.2.4.5 Compute

- $\begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} =$
- $\begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -3 \end{pmatrix} \begin{pmatrix} -2 \\ 0 \\ 3 \end{pmatrix} =$
- $\begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -3 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ -1 \end{pmatrix} =$
- $\begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -3 \end{pmatrix} \begin{pmatrix} 1 & -2 & -1 \\ 2 & 0 & 2 \\ -1 & 3 & -1 \end{pmatrix} =$

Homework 5.2.4.6 Let $A \in \mathbb{R}^{m\times n}$ and let D denote the diagonal matrix with diagonal elements $\delta_0, \delta_1, \cdots, \delta_{n-1}$. Partition A by columns: $A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)$.

$$AD = \left(\begin{array}{c|c|c|c} \delta_0 a_0 & \delta_1 a_1 & \cdots & \delta_{n-1} a_{n-1} \end{array} \right).$$

Always/Sometimes/Never

Homework 5.2.4.7 Let $A \in \mathbb{R}^{m\times n}$ and let D denote the diagonal matrix with diagonal elements $\delta_0, \delta_1, \cdots, \delta_{m-1}$. Partition A by rows: $A = \left(\begin{array}{c} \widetilde{a}_0^T \\ \hline \widetilde{a}_1^T \\ \hline \vdots \\ \hline \widetilde{a}_{m-1}^T \end{array} \right)$.

$$DA = \left(\begin{array}{c} \delta_0 \widetilde{a}_0^T \\ \hline \delta_1 \widetilde{a}_1^T \\ \hline \vdots \\ \hline \delta_{m-1} \widetilde{a}_{m-1}^T \end{array} \right).$$

Always/Sometimes/Never

Triangular matrices

Homework 5.2.4.8 Compute $\left(\begin{array}{ccc} 1 & -1 & -2 \\ 0 & 2 & 3 \\ 0 & 0 & 1 \end{array} \right) \left(\begin{array}{ccc} -2 & 1 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{array} \right) =$

Homework 5.2.4.9 Compute the following, using what you know about partitioned matrix-matrix multiplication:

$$\left(\begin{array}{cc|c} 1 & -1 & -2 \\ 0 & 2 & 3 \\ \hline 0 & 0 & 1 \end{array} \right) \left(\begin{array}{cc|c} -2 & 1 & -1 \\ 0 & 1 & 2 \\ \hline 0 & 0 & 1 \end{array} \right) =$$

Homework 5.2.4.10 Let $U, R \in \mathbb{R}^{n\times n}$ be upper triangular matrices. UR is an upper triangular matrix.

Always/Sometimes/Never

Homework 5.2.4.11 The product of an $n \times n$ lower triangular matrix times an $n \times n$ lower triangular matrix is a lower triangular matrix.

Always/Sometimes/Never

Homework 5.2.4.12 The product of an $n \times n$ lower triangular matrix times an $n \times n$ upper triangular matrix is a diagonal matrix.

Always/Sometimes/Never

Symmetric matrices

Homework 5.2.4.13 Let $A \in \mathbb{R}^{m \times n}$. $A^T A$ is symmetric.

Always/Sometimes/Never

Homework 5.2.4.14 Evaluate

- $\begin{pmatrix} -1 \\ 1 \\ 2 \end{pmatrix} \begin{pmatrix} -1 & 1 & 2 \end{pmatrix} =$
- $\begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix} \begin{pmatrix} 2 & 0 & -1 \end{pmatrix} =$
- $\left(\begin{array}{c|c} -1 & 2 \\ 1 & 0 \\ 2 & -1 \end{array}\right) \left(\begin{array}{ccc} -1 & 1 & 2 \\ \hline 2 & 0 & -1 \end{array}\right) =$
- $\begin{pmatrix} 1 \\ -2 \\ 2 \end{pmatrix} \begin{pmatrix} 1 & -2 & 2 \end{pmatrix} =$
- $\left(\begin{array}{cc|c} -1 & 2 & 1 \\ 1 & 0 & -2 \\ 2 & -1 & 2 \end{array}\right) \left(\begin{array}{ccc} -1 & 1 & 2 \\ 2 & 0 & -1 \\ \hline 1 & -2 & 2 \end{array}\right) =$

Homework 5.2.4.15 Let $x \in \mathbb{R}^n$. The outer product xx^T is symmetric.

Always/Sometimes/Never

Homework 5.2.4.16 Let $A \in \mathbb{R}^{n \times n}$ be symmetric and $x \in \mathbb{R}^n$. $A + xx^T$ is symmetric.

Always/Sometimes/Never

Homework 5.2.4.17 Let $A \in \mathbb{R}^{m\times n}$. Then AA^T is symmetric. (In your reasoning, we want you to use insights from previous homeworks.)

Always/Sometimes/Never

Homework 5.2.4.18 Let $A, B \in \mathbb{R}^{n\times n}$ be symmetric matrices. AB is symmetric.

Always/Sometimes/Never

A generalization of $A + xx^T$ with symmetric A and vector x, is given by

$$A := \alpha xx^T + A,$$

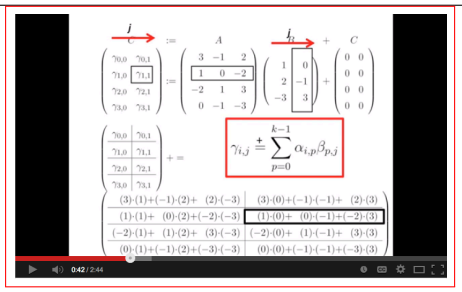
where α is a scalar. This is known as a *symmetric rank-1 update*.
The last exercise motivates the fact that the result itself is symmetric. The reason for the name "rank-1 update" will become clear later in the course, when we will see that a matrix that results from an outer product, yx^T, has rank at most equal to one.
This operation is sufficiently important that it is included in the `laff` library as function

```
[ y_out ] = laff_syr( alpha, x, A )
```

which updates $A := \alpha xx^T + A$.

5.3 Algorithms for Computing Matrix-Matrix Multiplication

5.3.1 Lots of Loops



☛ View at edX

In Theorem 5.1, partition C into elements (scalars), and A and B by rows and columns, respectively. In other words, let $M = m$, $m_i = 1$, $i = 0, \ldots, m-1$; $N = n$, $n_j = 1$, $j = 0, \ldots, n-1$; and $K = 1$, $k_0 = k$. Then

$$\left(\begin{array}{c|c|c|c} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n-1} \\ \hline \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,n-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,n-1} \end{array}\right), A = \left(\begin{array}{c} \tilde{a}_0^T \\ \hline \tilde{a}_1^T \\ \hline \vdots \\ \hline \tilde{a}_{m-1}^T \end{array}\right), \text{ and } B = \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array}\right)$$

so that

$$
\begin{array}{rcl}
C & = & \left(\begin{array}{c|c|c|c} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n-1} \\ \hline \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,n-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,n-1} \end{array}\right) = \left(\begin{array}{c} \tilde{a}_0^T \\ \hline \tilde{a}_1^T \\ \hline \vdots \\ \hline \tilde{a}_{m-1}^T \end{array}\right) \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array}\right) \\
 & = & \left(\begin{array}{c|c|c|c} \tilde{a}_0^T b_0 & \tilde{a}_0^T b_1 & \cdots & \tilde{a}_0^T b_{n-1} \\ \hline \tilde{a}_1^T b_0 & \tilde{a}_1^T b_1 & \cdots & \tilde{a}_1^T b_{n-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \tilde{a}_{m-1}^T b_0 & \tilde{a}_{m-1}^T b_1 & \cdots & \tilde{a}_{m-1}^T b_{n-1} \end{array}\right).
\end{array}
$$

As expected, $\gamma_{i,j} = \tilde{a}_i^T b_j$: the dot product of the ith row of A with the jth row of B.

Example 5.3

$$
\left(\begin{array}{rrr} -1 & 2 & 4 \\ \hline 1 & 0 & -1 \\ \hline 2 & -1 & 3 \end{array}\right)\left(\begin{array}{r|r} -2 & 2 \\ 0 & 1 \\ -2 & -1 \end{array}\right) = \left(\begin{array}{c|c} \left(\begin{array}{rrr} -1 & 2 & 4 \end{array}\right)\left(\begin{array}{r} -2 \\ 0 \\ -2 \end{array}\right) & \left(\begin{array}{rrr} -1 & 2 & 4 \end{array}\right)\left(\begin{array}{r} 2 \\ 1 \\ -1 \end{array}\right) \\ \hline \left(\begin{array}{rrr} 1 & 0 & -1 \end{array}\right)\left(\begin{array}{r} -2 \\ 0 \\ -2 \end{array}\right) & \left(\begin{array}{rrr} 1 & 0 & -1 \end{array}\right)\left(\begin{array}{r} 2 \\ 1 \\ -1 \end{array}\right) \\ \hline \left(\begin{array}{rrr} 2 & -1 & 3 \end{array}\right)\left(\begin{array}{r} -2 \\ 0 \\ -2 \end{array}\right) & \left(\begin{array}{rrr} 2 & -1 & 3 \end{array}\right)\left(\begin{array}{r} 2 \\ 1 \\ -1 \end{array}\right) \end{array}\right)
$$

$$
= \left(\begin{array}{r|r} -6 & -4 \\ \hline 0 & 3 \\ \hline -10 & 0 \end{array}\right)
$$

This motivates the following two algorithms for computing $C = AB + C$. In both, the outer two loops visit all elements $\gamma_{i,j}$ of C, and the inner loop updates a given $\gamma_{i,j}$ with the dot product of the ith row of A and the jth column of A. They differ in that the first updates C one column at a time (the outer loop is over the columns of C and B) while the second updates C one row at a time (the outer loop is over the rows of C and A).

for $j = 0, \ldots, n-1$
  for $i = 0, \ldots, m-1$
    for $p = 0, \ldots, k-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$\Big\} \gamma_{i,j} := \tilde{a}_i^T b_j + \gamma_{i,j}$

or

for $i = 0, \ldots, m-1$
  for $j = 0, \ldots, n-1$
    for $p = 0, \ldots, k-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$\Big\} \gamma_{i,j} := \tilde{a}_i^T b_j + \gamma_{i,j}$

Homework 5.3.1.1 Consider the MATLAB function

```
function [ C_out ] = MatMatMult( A, B, C )

[ m, n ] = size( C );
[ m_A, k ] = size( A );
[ m_B, n_B ] = size( B );

for j = 1:n
   for i = 1:m
      for p = 1:k
         C( i,j ) = A( i, p ) * B( p, j ) + C( i, j );
      end
   end
end
```

- Download the files `MatMatMult.m` and `test_MatMatMult.m` into, for example,

  `LAFFSpring2015 -> Programming -> Week5`

  (creating the directory if necessary).

- Examine the script `test_MatMatMult.m` and then execute it in the MATLAB Command Window: `test_MatMatMult`.

- Now, exchange the order of the loops:

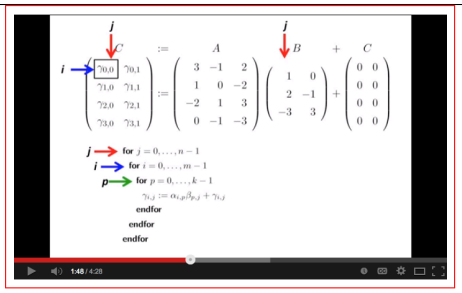
```
for j = 1:n
   for p = 1:k
      for i = 1:m
         C( i,j ) = A( i, p ) * B( p, j ) + C( i, j );
      end
   end
end
```

  save the result, and execute `test_MatMatMult` again. What do you notice?

- How may different ways can you order the "triple-nested loop"?

- Try them all and observe how the result of executing `test_MatMatMult` does or does not change.

5.3.2 Matrix-Matrix Multiplication by Columns



☛ View at edX

Homework 5.3.2.1 Let A and B be matrices and AB be well-defined and let B have at least four columns. If the first and fourth columns of B are the same, then the first and fourth columns of AB are the same.

Always/Sometimes/Never

Homework 5.3.2.2 Let A and B be matrices and AB be well-defined and let A have at least four columns. If the first and fourth columns of A are the same, then the first and fourth columns of AB are the same.

Always/Sometimes/Never

In Theorem 5.1 let us partition C and B by columns and not partition A. In other words, let $M = 1$, $m_0 = m$; $N = n$, $n_j = 1$, $j = 0, \ldots, n-1$; and $K = 1$, $k_0 = k$. Then

$$C = \left(\begin{array}{c|c|c|c} c_0 & c_1 & \cdots & c_{n-1} \end{array} \right) \quad \text{and} \quad B = \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array} \right)$$

so that

$$\left(\begin{array}{c|c|c|c} c_0 & c_1 & \cdots & c_{n-1} \end{array} \right) = C = AB = A \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} Ab_0 & Ab_1 & \cdots & Ab_{n-1} \end{array} \right).$$

Homework 5.3.2.3

- $\begin{pmatrix} 1 & -2 & 2 \\ -1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix} \left(\begin{array}{c|cc} -1 & & \\ 2 & & \\ 1 & & \end{array} \right) =$
- $\begin{pmatrix} 1 & -2 & 2 \\ -1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix} \left(\begin{array}{cc|c} -1 & 0 & \\ 2 & 1 & \\ 1 & -1 & \end{array} \right) =$
- $\begin{pmatrix} 1 & -2 & 2 \\ -1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix} \left(\begin{array}{cc|c} -1 & 0 & 1 \\ 2 & 1 & -1 \\ 1 & -1 & 2 \end{array} \right) =$

Example 5.4

$$\left(\begin{array}{ccc} -1 & 2 & 4 \\ 1 & 0 & -1 \\ 2 & -1 & 3 \end{array}\right)\left(\begin{array}{c|c} -2 & 2 \\ 0 & 1 \\ -2 & -1 \end{array}\right) = \left(\left(\begin{array}{ccc} -1 & 2 & 4 \\ 1 & 0 & -1 \\ 2 & -1 & 3 \end{array}\right)\left(\begin{array}{c} -2 \\ 0 \\ -2 \end{array}\right) \middle| \left(\begin{array}{ccc} -1 & 2 & 4 \\ 1 & 0 & -1 \\ 2 & -1 & 3 \end{array}\right)\left(\begin{array}{c} 2 \\ 1 \\ -1 \end{array}\right)\right)$$

$$= \left(\begin{array}{c|c} -6 & -4 \\ 0 & 3 \\ -10 & 0 \end{array}\right)$$

By moving the loop indexed by j to the outside in the algorithm for computing $C = AB + C$ we observe that

for $j = 0, \ldots, n-1$
  for $i = 0, \ldots, m-1$
    for $p = 0, \ldots, k-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

} $c_j := Ab_j + c_j$

or

for $j = 0, \ldots, n-1$
  for $p = 0, \ldots, k-1$
    for $i = 0, \ldots, m-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

} $c_j := Ab_j + c_j$

Exchanging the order of the two inner-most loops merely means we are using a different algorithm (dot product vs. AXPY) for the matrix-vector multiplication $c_j := Ab_j + c_j$.

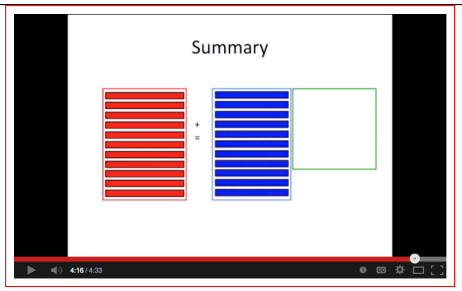
An algorithm that computes $C = AB + C$ one column at a time, represented with FLAME notation, is given in Figure 5.1

Homework 5.3.2.4 Implement the routine

```
[ C_out ] = Gemm_unb_var1( A, B, C )
```

based on the algorithm in Figure 5.1.

5.3.3 Matrix-Matrix Multiplication by Rows



☛ View at edX

Algorithm: $C := \text{GEMM\_UNB\_VAR1}(A, B, C)$

Partition $B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array}\right), C \rightarrow \left(\begin{array}{c|c} C_L & C_R \end{array}\right)$
where B_L has 0 columns, C_L has 0 columns

while $n(B_L) < n(B)$ **do**

Repartition

$$\left(\begin{array}{c|c} B_L & B_R \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array}\right), \left(\begin{array}{c|c} C_L & C_R \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} C_0 & c_1 & C_2 \end{array}\right)$$
where b_1 has 1 column, c_1 has 1 column

$c_1 := Ab_1 + c_1$

Continue with

$$\left(\begin{array}{c|c} B_L & B_R \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array}\right), \left(\begin{array}{c|c} C_L & C_R \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} C_0 & c_1 & C_2 \end{array}\right)$$

endwhile

Figure 5.1: Algorithm for $C = AB + C$, computing C one column at a time.

Homework 5.3.3.1 Let A and B be matrices and AB be well-defined and let A have at least four rows. If the first and fourth rows of A are the same, then the first and fourth rows of AB are the same.

Always/Sometimes/Never

In Theorem 5.1 partition C and A by rows and do not partition B. In other words, let $M = m$, $m_i = 1$, $i = 0, \ldots, m-1$; $N = 1$, $n_0 = n$; and $K = 1$, $k_0 = k$. Then

$$C = \left(\begin{array}{c} \tilde{c}_0^T \\ \hline \tilde{c}_1^T \\ \hline \vdots \\ \hline \tilde{c}_{m-1}^T \end{array}\right) \quad \text{and} \quad A = \left(\begin{array}{c} \tilde{a}_0^T \\ \hline \tilde{a}_1^T \\ \hline \vdots \\ \hline \tilde{a}_{m-1}^T \end{array}\right)$$

so that

$$\left(\begin{array}{c} \tilde{c}_0^T \\ \hline \tilde{c}_1^T \\ \hline \vdots \\ \hline \tilde{c}_{m-1}^T \end{array}\right) = C = AB = \left(\begin{array}{c} \tilde{a}_0^T \\ \hline \tilde{a}_1^T \\ \hline \vdots \\ \hline \tilde{a}_{m-1}^T \end{array}\right) B = \left(\begin{array}{c} \tilde{a}_0^T B \\ \hline \tilde{a}_1^T B \\ \hline \vdots \\ \hline \tilde{a}_{m-1}^T B \end{array}\right).$$

This shows how C can be computed one row at a time.

Example 5.5

$$\left(\begin{array}{ccc} -1 & 2 & 4 \\ \hline 1 & 0 & -1 \\ \hline 2 & -1 & 3 \end{array}\right)\left(\begin{array}{cc} -2 & 2 \\ 0 & 1 \\ -2 & -1 \end{array}\right) = \left(\begin{array}{c} \left(\begin{array}{ccc} -1 & 2 & 4 \end{array}\right)\left(\begin{array}{cc} -2 & 2 \\ 0 & 1 \\ -2 & -1 \end{array}\right) \\ \hline \left(\begin{array}{ccc} 1 & 0 & -1 \end{array}\right)\left(\begin{array}{cc} -2 & 2 \\ 0 & 1 \\ -2 & -1 \end{array}\right) \\ \hline \left(\begin{array}{ccc} 2 & -1 & 3 \end{array}\right)\left(\begin{array}{cc} -2 & 2 \\ 0 & 1 \\ -2 & -1 \end{array}\right) \end{array}\right) = \left(\begin{array}{cc} -6 & -4 \\ \hline 0 & 3 \\ \hline -10 & 0 \end{array}\right)$$

In the algorithm for computing $C = AB + C$ the loop indexed by i can be moved to the outside so that

for $i = 0, \ldots, m-1$
  for $j = 0, \ldots, n-1$
    for $p = 0, \ldots, k-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$\left.\begin{array}{l}\\\\\\\\\end{array}\right\} \tilde{c}_i^T := \tilde{a}_i^T B + \tilde{c}_i^T$

or

for $i = 0, \ldots, m-1$
  for $p = 0, \ldots, k-1$
    for $j = 0, \ldots, n-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$\left.\begin{array}{l}\\\\\\\\\end{array}\right\} \tilde{c}_i^T := \tilde{a}_i^T B + \tilde{c}_i^T$

An algorithm that computes $C = AB + C$ row at a time, represented with FLAME notation, is given in Figure 5.2.

Homework 5.3.3.2

- $\left(\begin{array}{ccc} \hline 1 & -2 & 2 \\ \hline \\ \\ \end{array}\right)\left(\begin{array}{ccc} -1 & 0 & 1 \\ 2 & 1 & -1 \\ 1 & -1 & 2 \end{array}\right) =$
- $\left(\begin{array}{ccc} 1 & -2 & 2 \\ \hline -1 & 2 & 1 \\ \hline \\ \end{array}\right)\left(\begin{array}{ccc} -1 & 0 & 1 \\ 2 & 1 & -1 \\ 1 & -1 & 2 \end{array}\right) =$
- $\left(\begin{array}{ccc} 1 & -2 & 2 \\ -1 & 2 & 1 \\ \hline 0 & 1 & 2 \\ \hline \end{array}\right)\left(\begin{array}{ccc} -1 & 0 & 1 \\ 2 & 1 & -1 \\ 1 & -1 & 2 \end{array}\right) =$

Algorithm: $C := \text{GEMM\_UNB\_VAR2}(A, B, C)$

Partition $A \rightarrow \left(\frac{A_T}{A_B}\right)$, $C \rightarrow \left(\frac{C_T}{C_B}\right)$

where A_T has 0 rows, C_T has 0 rows

while $m(A_T) < m(A)$ **do**

Repartition

$$\left(\frac{A_T}{A_B}\right) \rightarrow \left(\frac{\frac{A_0}{a_1^T}}{A_2}\right), \left(\frac{C_T}{C_B}\right) \rightarrow \left(\frac{\frac{C_0}{c_1^T}}{C_2}\right)$$

where a_1 has 1 row, c_1 has 1 row

$c_1^T := a_1^T B + c_1^T$

Continue with

$$\left(\frac{A_T}{A_B}\right) \leftarrow \left(\frac{\frac{A_0}{a_1^T}}{A_2}\right), \left(\frac{C_T}{C_B}\right) \leftarrow \left(\frac{\frac{C_0}{c_1^T}}{C_2}\right)$$

endwhile

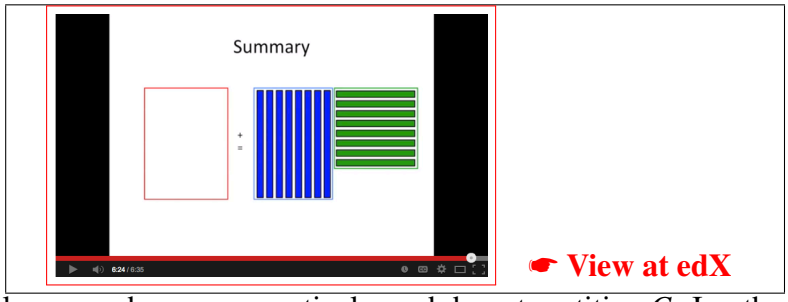
Figure 5.2: Algorithm for $C = AB + C$, computing C one row at a time.

Homework 5.3.3.3 Implement the routine

```
[ C_out ] = Gemm_unb_var2( A, B, C )
```

based on the algorithm in Figure 5.2.

5.3.4 Matrix-Matrix Multiplication with Rank-1 Updates



In Theorem 5.1 partition A and B by columns and rows, respectively, and do not partition C. In other words, let $M = 1$, $m_0 = m$; $N = 1$, $n_0 = n$; and $K = k$, $k_p = 1$, $p = 0, \ldots, k-1$. Then

$$A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{k-1} \end{array} \right) \quad \text{and} \quad B = \left(\begin{array}{c} \tilde{b}_0^T \\ \hline \tilde{b}_1^T \\ \hline \vdots \\ \hline \tilde{b}_{k-1}^T \end{array} \right)$$

so that

$$C = AB = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{k-1} \end{array} \right) \left(\begin{array}{c} \tilde{b}_0^T \\ \hline \tilde{b}_1^T \\ \hline \vdots \\ \hline \tilde{b}_{k-1}^T \end{array} \right) = a_0 \tilde{b}_0^T + a_1 \tilde{b}_1^T + \cdots + a_{k-1} \tilde{b}_{k-1}^T.$$

Notice that each term $a_p \tilde{b}_p^T$ is an outer product of a_p and $\tilde{b}_p$. Thus, if we start with $C := 0$, the zero matrix, then we can compute $C := AB + C$ as

$$C := a_{k-1} \tilde{b}_{k-1}^T + (\cdots + (a_p \tilde{b}_p^T + (\cdots + (a_1 \tilde{b}_1^T + (a_0 \tilde{b}_0^T + C)) \cdots)) \cdots),$$

which illustrates that $C := AB$ can be computed by first setting C to zero, and then repeatedly updating it with rank-1 updates.

Example 5.6

$$\begin{aligned}
&\left(\begin{array}{c|c|c} -1 & 2 & 4 \\ 1 & 0 & -1 \\ 2 & -1 & 3 \end{array} \right) \left(\begin{array}{cc} -2 & 2 \\ \hline 0 & 1 \\ \hline -2 & -1 \end{array} \right) \\
&= \left(\begin{array}{c} -1 \\ 1 \\ 2 \end{array} \right) \left(\begin{array}{cc} -2 & 2 \end{array} \right) + \left(\begin{array}{c} 2 \\ 0 \\ -1 \end{array} \right) \left(\begin{array}{cc} 0 & 1 \end{array} \right) + \left(\begin{array}{c} 4 \\ -1 \\ 3 \end{array} \right) \left(\begin{array}{cc} -2 & -1 \end{array} \right) \\
&= \left(\begin{array}{cc} 2 & -2 \\ -2 & 2 \\ -4 & 4 \end{array} \right) + \left(\begin{array}{cc} 0 & 2 \\ 0 & 0 \\ 0 & -1 \end{array} \right) + \left(\begin{array}{cc} -8 & -4 \\ 2 & 1 \\ -6 & -3 \end{array} \right) = \left(\begin{array}{cc} -6 & -4 \\ 0 & 3 \\ -10 & 0 \end{array} \right)
\end{aligned}$$

Algorithm: $C := \text{GEMM\_UNB\_VAR3}(A, B, C)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right)$, $B \rightarrow \left(\begin{array}{c} B_T \\ \hline B_B \end{array} \right)$

where A_L has 0 columns, B_T has 0 rows

while $n(A_L) < n(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array} \right) \rightarrow \left(\begin{array}{c} B_0 \\ \hline b_1^T \\ \hline B_2 \end{array} \right)$$

where a_1 has 1 column, b_1 has 1 row

$C := a_1 b_1^T + C$

Continue with

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array} \right) \leftarrow \left(\begin{array}{c} B_0 \\ \hline b_1^T \\ \hline B_2 \end{array} \right)$$

endwhile

Figure 5.3: Algorithm for $C = AB + C$, computing C via rank-1 updates.

In the algorithm for computing $C := AB + C$ the loop indexed by p can be moved to the outside so that

for $p = 0, \ldots, k-1$
  for $j = 0, \ldots, n-1$
    for $i = 0, \ldots, m-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$\Big\}\ C := a_p \tilde{b}_p^T + C$ or

for $p = 0, \ldots, k-1$
  for $i = 0, \ldots, m-1$
    for $j = 0, \ldots, n-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$\Big\}\ C := a_p \tilde{b}_p^T + C$

An algorithm that computes $C = AB + C$ with rank-1 updates, represented with FLAME notation, is given in Figure 5.3.

Homework 5.3.4.1

- $\left(\left.\begin{array}{c} 1 \\ -1 \\ 0 \end{array}\right|\right)\left(\begin{array}{ccc} -1 & 0 & 1 \end{array}\right) =$
- $\left(\begin{array}{c|c|c} & -2 & \\ & 2 & \\ & 1 & \end{array}\right)\left(\begin{array}{ccc} 2 & 1 & -1 \end{array}\right) =$
- $\left(\left.\begin{array}{c} \\ \\ \end{array}\right|\begin{array}{c} 2 \\ 1 \\ 2 \end{array}\right)\left(\begin{array}{ccc} 1 & -1 & 2 \end{array}\right) =$
- $\left(\begin{array}{ccc} 1 & -2 & 2 \\ -1 & 2 & 1 \\ 0 & 1 & 2 \end{array}\right)\left(\begin{array}{ccc} -1 & 0 & 1 \\ 2 & 1 & -1 \\ 1 & -1 & 2 \end{array}\right) =$

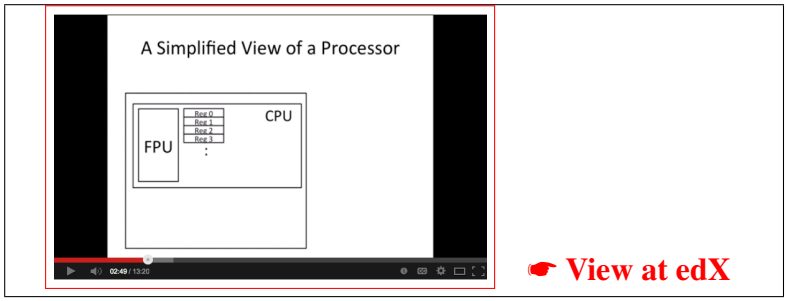
Homework 5.3.4.2 Implement the routine

```
[ C_out ] = Gemm_unb_var2( A, B, C )
```

based on the algorithm in Figure 5.3.

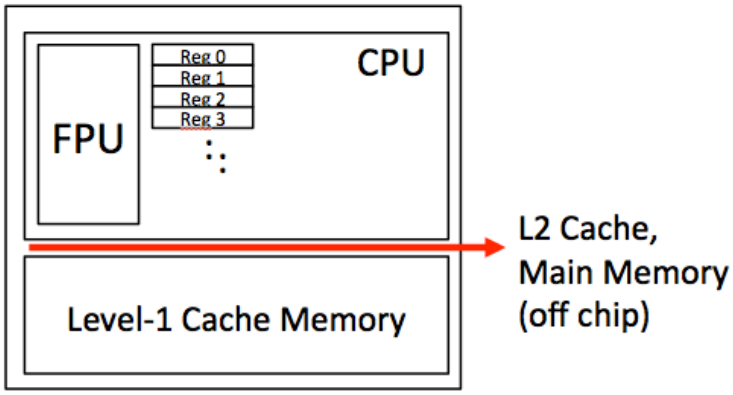
5.4 Enrichment

5.4.1 Slicing and Dicing for Performance



Computer Architecture (Very) Basics

A highly simplified description of a processor is given below.

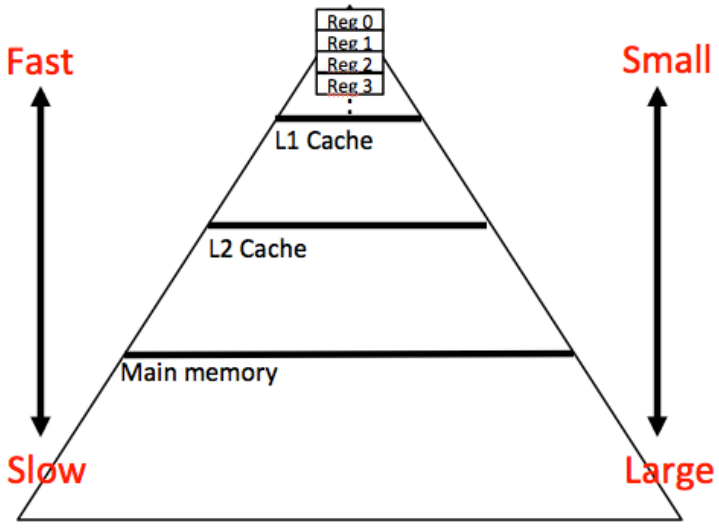


Yes, it is very, very simplified. For example, these days one tends to talk about "cores" and there are multiple cores on a computer chip. But this simple view of what a processor is will serve our purposes just fine.

At the heart of the processor is the Central Processing Unit (CPU). It is where the computing happens. For us, the important parts of the CPU are the Floating Point Unit (FPU), where floating point computations are performed, and the registers, where data with which the FPU computes must reside. A typical processor will have 16-64 registers. In addition to this, a typical processor has a small amount of memory on the chip, called the Level-1 (L1) Cache. The L1 cache can typically hold 16Kbytes (about 16,000 bytes) or 32Kbytes. The L1 cache is fast memory, fast enough to keep up with the FPU as it computes.

Additional memory is available "off chip". There is the Level-2 (L2) Cache and Main Memory. The L2 cache is slower than the L1 cache, but not as slow as main memory. To put things in perspective: in the time it takes to bring a floating point number from main memory onto the processor, the FPU can perform 50-100 floating point computations. **Memory is very slow.** (There might be an L3 cache, but let's not worry about that.) Thus, where in these different layers of the hierarchy of memory data exists greatly affects how fast computation can be performed, since waiting for the data may become the dominating factor. Understanding this memory hierarchy is important.

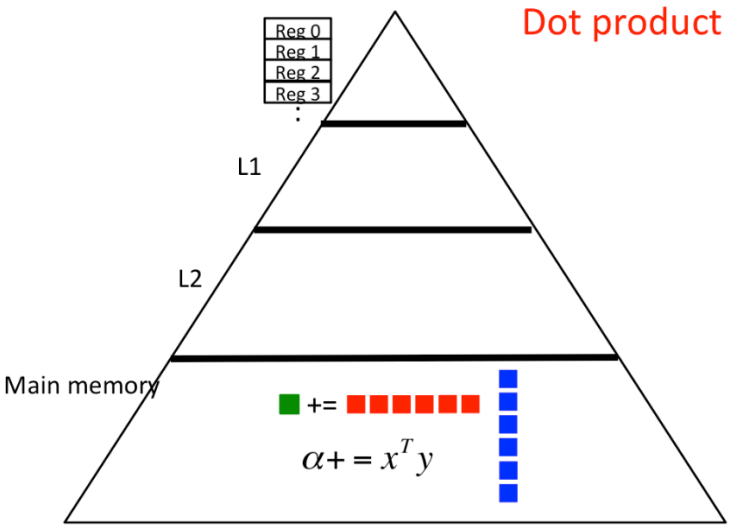
Here is how to view the memory as a pyramid:



At the top, there are the registers. For computation to happen, data must be in registers. Below it are the L1 and L2 caches. At the bottom, main memory. Below that layer, there may be further layers, like disk storage.

Now, the name of the game is to keep data in the faster memory layers to overcome the slowness of main memory. Notice that computation can also hide the "latency" to memory: one can overlap computation and the fetching of data.

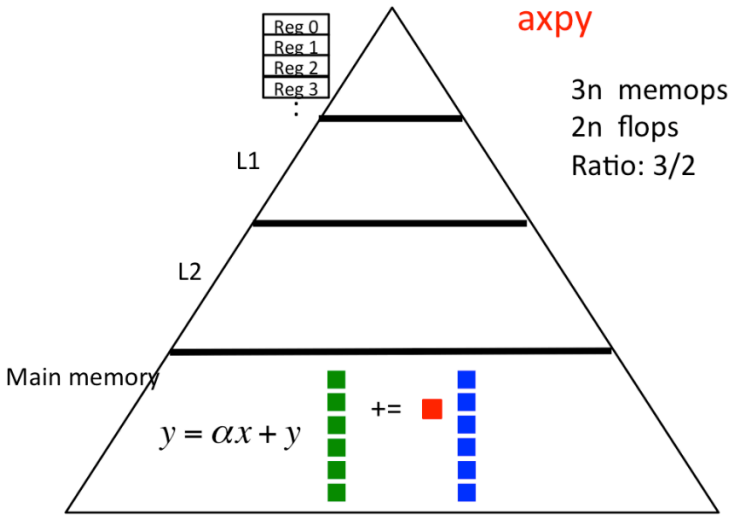
Vector-Vector Computations Let's consider performing the dot product operation $\alpha := x^T y$, with vectors $x, y \in \mathbb{R}^n$ that reside in main memory.



Notice that inherently the components of the vectors must be loaded into registers at some point of the computation, requiring $2n$ memory operations (memops). The scalar α can be stored in a register as the computation proceeds, so that it only needs to be written to main memory once, at the end of the computation. This one memop can be ignored relative to the $2n$ memops required to fetch the vectors. Along the way, (approximately) $2n$ flops are performed: an add and a multiply for each pair of components of x and y.

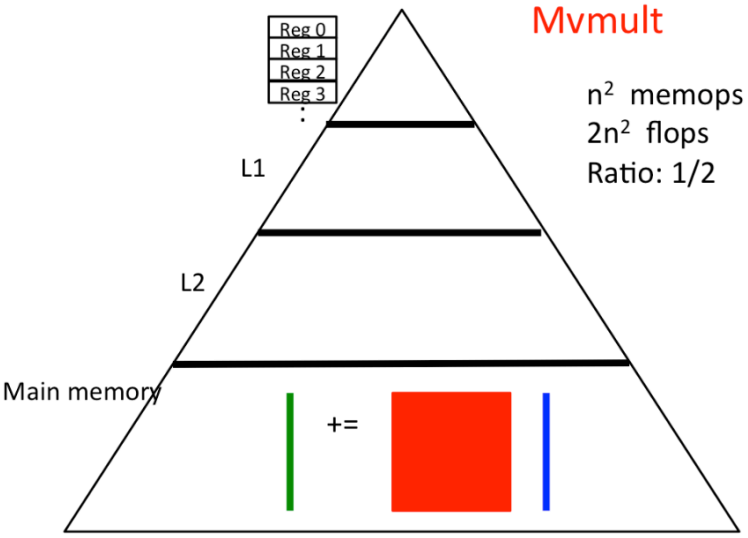
The problem is that the ratio of memops to flops is $2n/2n = 1/1$. Since memops are extremely slow, the cost is in moving the data, not in the actual computation itself. Yes, there is cache memory in between, but if the data starts in main memory, this is of no use: there isn't any reuse of the components of the vectors.

The problem is worse for the AXPY operation, $y := \alpha x + y$:



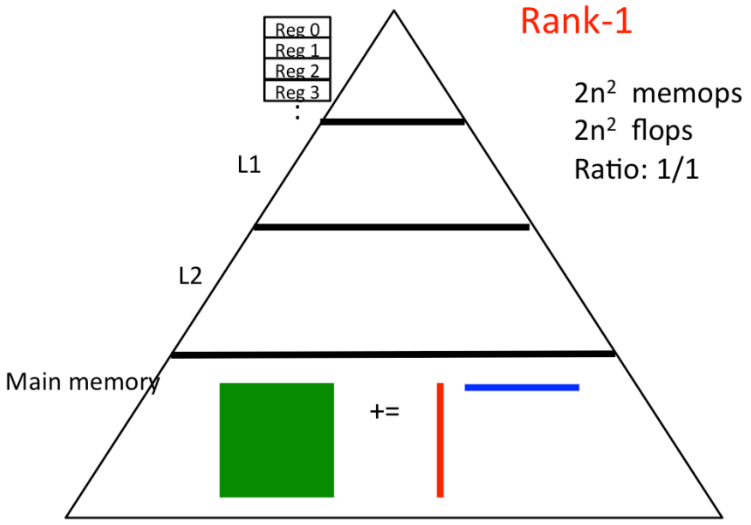
Here the components of the vectors x and y must be read from main memory, and the result y must be written back to main memory, for a total of $3n$ memops. The scalar α can be kept in a register, and therefore reading it from main memory is insignificant. The computation requires $2n$ flops, yielding a ratio of 3 memops for every 2 flops.

Matrix-Vector Computations Now, let's examine how matrix-vector multiplication, $y := Ax + y$, fares. For our analysis, we will assume a square $n \times n$ matrix A. All operands start in main memory.



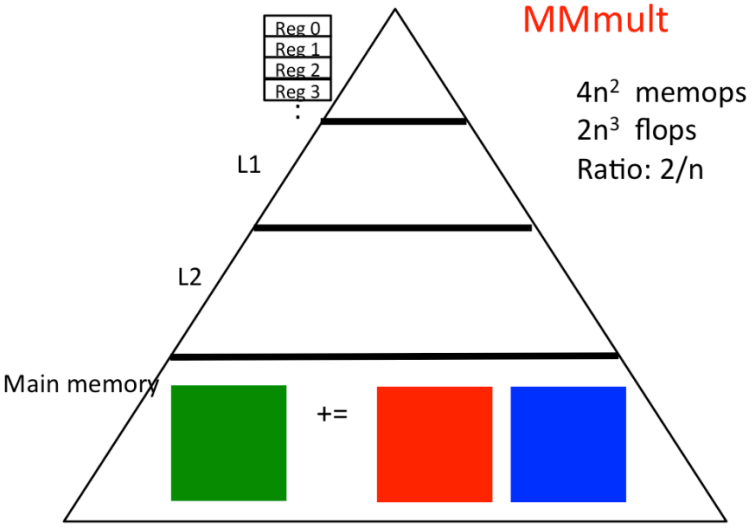
Now, inherently, all $n \times n$ elements of A must be read from main memory, requiring n^2 memops. Inherently, for each element of A only two flops are performed: an add and a multiply, for a total of $2n^2$ flops. There *is* an opportunity to bring components of x and/or y into cache memory and/or registers, and reuse them there for many computations. For example, if y is computed via dot products of rows of A with the vector x, the vector x can be brought into cache memory and reused many times. The component of y being computed can then be kept in a registers during the computation of the dot product. For this reason, we ignore the cost of reading and writing the vectors. Still, the ratio of memops to flops is approximately $n^2/2n^2 = 1/2$. This is only slightly better than the ratio for dot and AXPY.

The story is worse for a rank-1 update, $A := xy^T + A$. Again, for our analysis, we will assume a square $n \times n$ matrix A. All operands start in main memory.



Now, inherently, all $n \times n$ elements of A must be read from main memory, requiring n^2 memops. But now, after having been updated, each element must also be written back to memory, for another n^2 memops. Inherently, for each element of A only two flops are performed: an add and a multiply, for a total of $2n^2$ flops. Again, there *is* an opportunity to bring components of x and/or y into cache memory and/or registers, and reuse them there for many computations. Again, for this reason we ignore the cost of reading the vectors. Still, the ratio of memops to flops is approximately $2n^2/2n^2 = 1/1$.

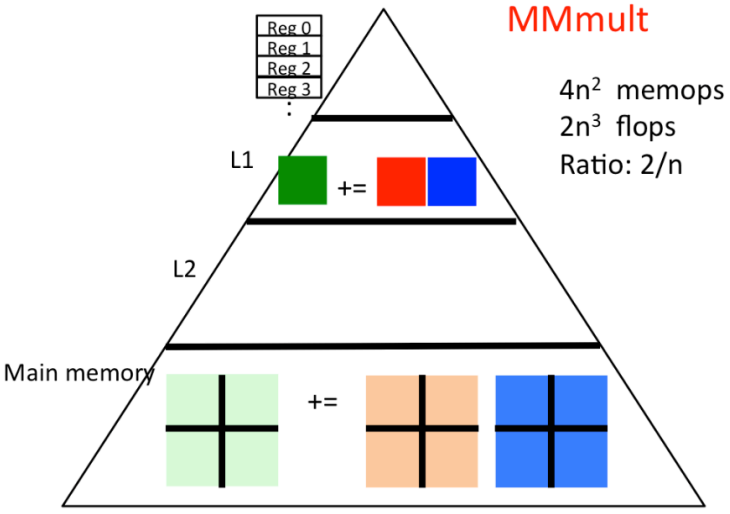
Matrix-Matrix Computations Finally, let's examine how matrix-matrix multiplication, $C := AB + C$, overcomes the memory bottleneck. For our analysis, we will assume all matrices are square $n \times n$ matrices and all operands start in main memory.



Now, inherently, all elements of the three matrices must be read at least once from main memory, requiring $3n^2$ memops, and C must be written at least once back to main memory, for another n^2 memops. We saw that a matrix-matrix multiplication requires a total of $2n^3$ flops. If this can be achieved, then the ratio

of memops to flops becomes $4n^2/2n^3 = 2/n$. If n is large enough, the cost of accessing memory can be overcome. To achieve this, all three matrices must be brought into cache memory, the computation performed while the data is in cache memory, and then the result written out to main memory.

The problem is that the matrices typically are too big to fit in, for example, the L1 cache. To overcome this limitation, we can use our insight that matrices can be partitioned, and matrix-matrix multiplication can be performed with submatrices (blocks).



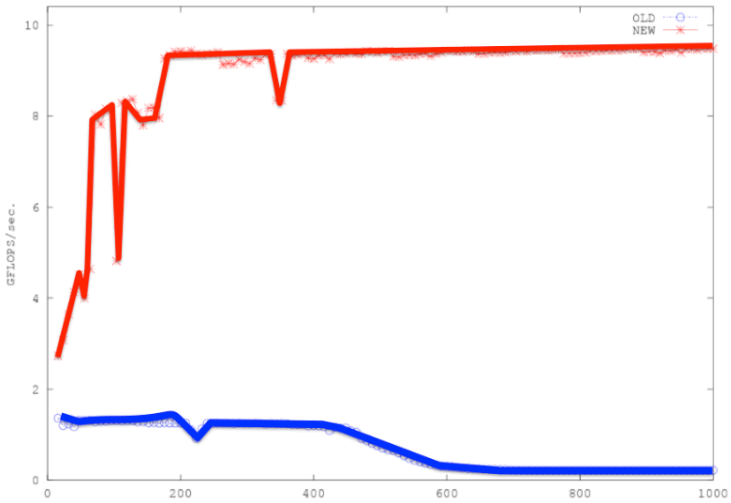
This way, near-peak performance can be achieved.

To achieve very high performance, one has to know how to partition the matrices more carefully, and arrange the operations in a very careful order. But the above describes the fundamental ideas.

5.4.2 How It is Really Done

☛ View at edX

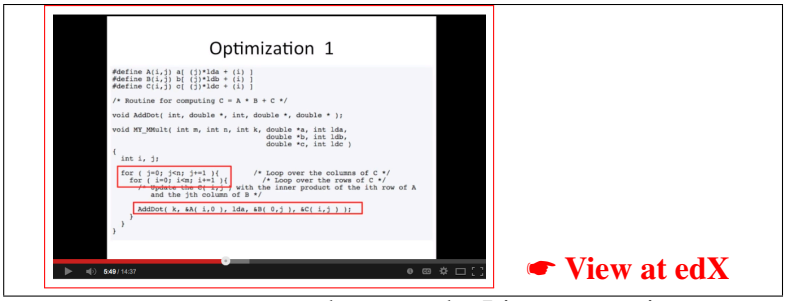
Measuring Performance There are two attributes of a processor that affect the rate at which it can compute: its clock rate, which is typically measured in GHz (billions of cycles per second) and the number of floating point computations that it can perform per cycle. Multiply these two numbers together, and you get the rate at which floating point computations can be performed, measured in GFLOPS/sec (billions of floating point operations per second). The below graph reports performance obtained on a laptop of ours. The details of the processor are not important for this descussion, since the performance is typical.



Along the x-axis, the matrix sizes $m = n = k$ are reported. Along the y-axis performance is reported in GFLOPS/sec. The important thing is that the top of the graph represents the peak of the processor, so that it is easy to judge what percent of peak is attained.

The blue line represents a basic implementation with a triple-nested loop. When the matrices are small, the data fits in the L2 cache, and performance is (somewhat) better. As the problem sizes increase, memory becomes more and more a bottleneck. Pathetic performance is achieved. The red line is a careful implementation that also blocks for better cache reuse. Obviously, considerable improvement is achieved.

Try It Yourself!



If you know how to program in C and have access to a computer that runs the Linux operating system, you may want to try the exercise on the following wiki page:

http://wiki.cs.utexas.edu/rvdg/HowToOptimizeGemm

Others may still learn something by having a look without trying it themselves.

No, we do not have time to help you with this exercise... You can ask each other questions online, but we cannot help you with this... We are just too busy with the MOOC right now...

Further Reading

- Kazushige Goto is famous for his implementation of matrix-matrix multiplication. The following New York Times article on his work may amuse you:

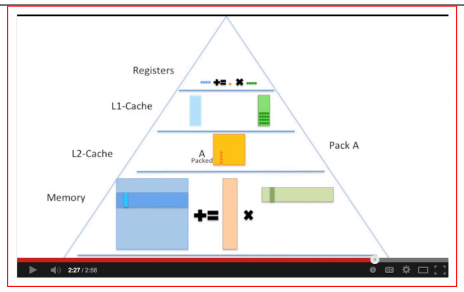
  Writing the Fastest Code, by Hand, for Fun: A Human Computer Keeps ..

- An article that describes his approach to matrix-matrix multiplication is

Kazushige Goto, Robert A. van de Geijn.
Anatomy of high-performance matrix multiplication.
ACM Transactions on Mathematical Software (TOMS), 2008.

It can be downloaded for free by first going to the FLAME publication webpage and clicking on Journal Publication #11. We believe you will be happy to find that you can understand at least the high level issues in that paper.

The following animation of how the memory hierarchy is utilized in Goto's approach may help clarify the above paper:



☛ View at edX

- A more recent paper that takes the insights further is

  Field G. Van Zee, Robert A. van de Geijn.
  BLIS: A Framework for Rapid Instantiation of BLAS Functionality.
  ACM Transactions on Mathematical Software.
  (to appear)

  It is also available from the FLAME publication webpage by clicking on Journal Publication #33.

- A paper that then extends these techniques to what are considered "many-core" architectures is

  Tyler M. Smith, Robert van de Geijn, Mikhail Smelyanskiy, Jeff R. Hammond, and Field G. Van Zee.
  Anatomy of High-Performance Many-Threaded Matrix Multiplication.
  International Parallel and Distributed Processing Symposium 2014. (to appear)

  It is also available from the FLAME publication webpage by clicking on Conference Publication #35. Around 90% of peak on 60 cores running 240 threads... At the risk of being accused of bragging, this is quite exceptional.

Notice that two of these papers have not even been published in print yet. You have arrived at the frontier of National Science Foundation (NSF) sponsored research, after only five weeks.

5.5 Wrap Up

5.5.1 Homework

For all of the below homeworks, only consider matrices that have real valued elements.

Homework 5.5.1.1 Let A and B be matrices and AB be well-defined. $(AB)^2 = A^2B^2$.
Always/Sometimes/Never

Homework 5.5.1.2 Let A be symmetric. A^2 is symmetric.
Always/Sometimes/Never

Homework 5.5.1.3 Let $A, B \in \mathbb{R}^{n\times n}$ both be symmetric. AB is symmetric.
Always/Sometimes/Never

Homework 5.5.1.4 Let $A, B \in \mathbb{R}^{n\times n}$ both be symmetric. $A^2 - B^2$ is symmetric.
Always/Sometimes/Never

Homework 5.5.1.5 Let $A, B \in \mathbb{R}^{n\times n}$ both be symmetric. $(A+B)(A-B)$ is symmetric.
Always/Sometimes/Never

Homework 5.5.1.6 Let $A, B \in \mathbb{R}^{n\times n}$ both be symmetric. ABA is symmetric.
Always/Sometimes/Never

Homework 5.5.1.7 Let $A, B \in \mathbb{R}^{n\times n}$ both be symmetric. $ABAB$ is symmetric.
Always/Sometimes/Never

Homework 5.5.1.8 Let A be symmetric. $A^TA = AA^T$.
Always/Sometimes/Never

Homework 5.5.1.9 If $A = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$ then $A^TA = AA^T$.
True/False

Homework 5.5.1.10 Propose an algorithm for computing $C := UR$ where C, U, and R are all upper triangular matrices by completing the below algorithm.

Algorithm: $[C] := \text{TRTRMM\_UU\_UNB\_VAR1}(U, R, C)$

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right)$, $R \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array}\right)$, $C \rightarrow \left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right)$

where U_{TL} is 0×0, R_{TL} is 0×0, C_{TL} is 0×0

while $m(U_{TL}) < m(U)$ **do**

Repartition

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ \hline R_{20} & r_{21} & R_{22} \end{array}\right),$$

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} C_{00} & c_{01} & C_{02} \\ \hline c_{10}^T & \gamma_{11} & c_{12}^T \\ \hline C_{20} & c_{21} & C_{22} \end{array}\right)$$

where υ_{11} is 1×1, ρ_{11} is 1×1, γ_{11} is 1×1

Continue with

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right), \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ \hline R_{20} & r_{21} & R_{22} \end{array}\right),$$

$$\left(\begin{array}{c|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} C_{00} & c_{01} & C_{02} \\ \hline c_{10}^T & \gamma_{11} & c_{12}^T \\ \hline C_{20} & c_{21} & C_{22} \end{array}\right)$$

endwhile

Hint: consider Homework 5.2.4.10. Then implement and test it.

Challenge 5.5.1.11 Propose many algorithms for computing $C := UR$ where C, U, and R are all upper triangular matrices. Hint: Think about how we created matrix-vector multiplication algorithms for the case where A was triangular. How can you similarly take the three different algorithms discussed in Units 5.3.2-4 and transform them into algorithms that take advantage of the triangular shape of the matrices?

Challenge 5.5.1.12 Propose many algorithms for computing $C := UR$ where C, U, and R are all upper triangular matrices. This time, derive all algorithm systematically by following the methodology in

The Science of Programming Matrix Computations.

(You will want to read Chapters 2-5.)
(You may want to use the blank "worksheet" on the next page.)

Step	**Annotated Algorithm:** $[C] := \text{TRTRMM\_UU\_UNB}\ (U,R,C)$
1a	$\left\{C = \widehat{C}\right\}$
4	**Partition** $U \rightarrow \left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right)$, $R \rightarrow \left(\begin{array}{c\|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array}\right)$, $C \rightarrow \left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right)$ **where** U_{TL} is 0×0, R_{TL} is 0×0, C_{TL} is 0×0
2	$\left\{ \left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) = \right\}$
3	**while** $m(U_{TL}) < m(U)$ **do**
2,3	$\left\{ \left(\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) = \right) \wedge (m(U_{TL}) < m(U)) \right\}$
5a	**Repartition** $\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right)$, $\left(\begin{array}{c\|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ \hline R_{20} & r_{21} & R_{22} \end{array}\right)$, $\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} C_{00} & c_{01} & C_{02} \\ \hline c_{10}^T & \gamma_{11} & c_{12}^T \\ \hline C_{20} & c_{21} & C_{22} \end{array}\right)$ **where** υ_{11} is 1×1, ρ_{11} is 1×1, γ_{11} is 1×1
6	$\left\{ \left(\begin{array}{c\|c\|c} C_{00} & c_{01} & C_{02} \\ \hline c_{10}^T & \gamma_{11} & c_{12}^T \\ \hline C_{20} & c_{21} & C_{22} \end{array}\right) = \right\}$
8	
5b	**Continue with** $\left(\begin{array}{c\|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & \upsilon_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array}\right)$, $\left(\begin{array}{c\|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ \hline R_{20} & r_{21} & R_{22} \end{array}\right)$, $\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} C_{00} & c_{01} & C_{02} \\ \hline c_{10}^T & \gamma_{11} & c_{12}^T \\ \hline C_{20} & c_{21} & C_{22} \end{array}\right)$
7	$\left\{ \left(\begin{array}{c\|c\|c} C_{00} & c_{01} & C_{02} \\ \hline c_{10}^T & \gamma_{11} & c_{12}^T \\ \hline C_{20} & c_{21} & C_{22} \end{array}\right) = \right\}$
2	$\left\{ \left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) = \right\}$
	endwhile
2,3	$\left\{ \left(\left(\begin{array}{c\|c} C_{TL} & C_{TR} \\ \hline C_{BL} & C_{BR} \end{array}\right) = \right) \wedge \neg(m(U_{TL}) < m(U)) \right\}$
1b	$\{C = UR\}$

5.5.2 Summary

Theorem 5.7 *Let $C \in \mathbb{R}^{m\times n}$, $A \in \mathbb{R}^{m\times k}$, and $B \in \mathbb{R}^{k\times n}$. Let*

- $m = m_0 + m_1 + \cdots m_{M-1}$, $m_i \geq 0$ *for* $i = 0, \ldots, M-1$;
- $n = n_0 + n_1 + \cdots n_{N-1}$, $n_j \geq 0$ *for* $j = 0, \ldots, N-1$; *and*
- $k = k_0 + k_1 + \cdots k_{K-1}$, $k_p \geq 0$ *for* $p = 0, \ldots, K-1$.

Partition

$$C = \left(\begin{array}{c|c|c|c} C_{0,0} & C_{0,1} & \cdots & C_{0,N-1} \\ \hline C_{1,0} & C_{1,1} & \cdots & C_{1,N-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline C_{M-1,0} & C_{M-1,1} & \cdots & C_{M-1,N-1} \end{array}\right), A = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,K-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,K-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,K-1} \end{array}\right),$$

$$\text{and } B = \left(\begin{array}{c|c|c|c} B_{0,0} & B_{0,1} & \cdots & B_{0,N-1} \\ \hline B_{1,0} & B_{1,1} & \cdots & B_{1,N-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline B_{K-1,0} & B_{K-1,1} & \cdots & B_{K-1,N-1} \end{array}\right),$$

with $C_{i,j} \in \mathbb{R}^{m_i\times n_j}$, $A_{i,p} \in \mathbb{R}^{m_i\times k_p}$, and $B_{p,j} \in \mathbb{R}^{k_p\times n_j}$. Then $C_{i,j} = \sum_{p=0}^{K-1} A_{i,p}B_{p,j}$.

If one partitions matrices C, A, and B into blocks, **and** one makes sure the dimensions match up, **then** blocked matrix-matrix multiplication proceeds exactly as does a regular matrix-matrix multiplication **except** that individual multiplications of scalars commute while (in general) individual multiplications with matrix blocks (submatrices) do not.

Properties of matrix-matrix multiplication

- Matrix-matrix multiplication is *not* commutative: In general, $AB \neq BA$.
- Matrix-matrix multiplication is associative: $(AB)C = A(BC)$. Hence, we can just write ABC.
- Special case: $e_i^T(Ae_j) = (e_i^T A)e_j = e_i^T Ae_j = \alpha_{i,j}$ (the i, j element of A).
- Matrix-matrix multiplication is distributative: $A(B+C) = AB + AC$ and $(A+B)C = AC + BC$.

Transposing the product of two matrices

$$(AB)^T = B^T A^T$$

Product with identity matrix

In the following, assume the matrices are "of appropriate size."

$$IA = AI = A$$

Product with a diagonal matrix

$$\left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array}\right)\begin{pmatrix} \delta_0 & 0 & \cdots & 0 \\ 0 & \delta_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_{n-1} \end{pmatrix} = \left(\begin{array}{c|c|c|c} \delta_0 a_0 & \delta_1 a_1 & \cdots & \delta_1 a_{n-1} \end{array}\right)$$

$$\begin{pmatrix} \delta_0 & 0 & \cdots & 0 \\ 0 & \delta_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_{m-1} \end{pmatrix}\begin{pmatrix} \widetilde{a}_0^T \\ \hline \widetilde{a}_1^T \\ \hline \vdots \\ \hline \widetilde{a}_{m-1}^T \end{pmatrix} = \begin{pmatrix} \delta_0 \widetilde{a}_0^T \\ \hline \delta_1 \widetilde{a}_1^T \\ \hline \vdots \\ \hline \delta_{m-1} \widetilde{a}_{m-1}^T \end{pmatrix}$$

Product of triangular matrices

In the following, assume the matrices are "of appropriate size."

- The product of two lower triangular matrices is lower triangular.
- The product of two upper triangular matrices is upper triangular.

Matrix-matrix multiplication involving symmetric matrices

In the following, assume the matrices are "of appropriate size."

- $A^T A$ is symmetric.
- AA^T is symmetric.
- If A is symmetric then $A + \beta x x^T$ is symmetric.

Loops for computing $C := AB$

$$\begin{aligned} C &= \left(\begin{array}{c|c|c|c} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n-1} \\ \hline \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,n-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,n-1} \end{array}\right) = \begin{pmatrix} \tilde{a}_0^T \\ \hline \tilde{a}_1^T \\ \hline \vdots \\ \hline \tilde{a}_{m-1}^T \end{pmatrix}\left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array}\right) \\ &= \left(\begin{array}{c|c|c|c} \tilde{a}_0^T b_0 & \tilde{a}_0^T b_1 & \cdots & \tilde{a}_0^T b_{n-1} \\ \hline \tilde{a}_1^T b_0 & \tilde{a}_1^T b_1 & \cdots & \tilde{a}_1^T b_{n-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \tilde{a}_{m-1}^T b_0 & \tilde{a}_{m-1}^T b_1 & \cdots & \tilde{a}_{m-1}^T b_{n-1} \end{array}\right). \end{aligned}$$

Algorithms for computing $C := AB + C$ via dot products.

for $j = 0, \ldots, n-1$
  for $i = 0, \ldots, m-1$
    for $p = 0, \ldots, k-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$\gamma_{i,j} := \tilde{a}_i^T b_j + \gamma_{i,j}$

or

for $i = 0, \ldots, m-1$
  for $j = 0, \ldots, n-1$
    for $p = 0, \ldots, k-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$\gamma_{i,j} := \tilde{a}_i^T b_j + \gamma_{i,j}$

Computing $C := AB$ by columns

$$\left(\begin{array}{c|c|c|c} c_0 & c_1 & \cdots & c_{n-1} \end{array} \right) = C = AB = A \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} Ab_0 & Ab_1 & \cdots & Ab_{n-1} \end{array} \right).$$

Algorithms for computing $C := AB + C$:

for $j = 0, \ldots, n-1$
  for $i = 0, \ldots, m-1$
    for $p = 0, \ldots, k-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$c_j := Ab_j + c_j$

or

for $j = 0, \ldots, n-1$
  for $p = 0, \ldots, k-1$
    for $i = 0, \ldots, m-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$c_j := Ab_j + c_j$

Algorithm: $C := \text{GEMM\_UNB\_VAR1}(A, B, C)$

Partition $B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array} \right)$, $C \rightarrow \left(\begin{array}{c|c} C_L & C_R \end{array} \right)$
  where B_L has 0 columns, C_L has 0 columns

while $n(B_L) < n(B)$ **do**

  Repartition

$$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right), \left(\begin{array}{c|c} C_L & C_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} C_0 & c_1 & C_2 \end{array} \right)$$

  where b_1 has 1 column, c_1 has 1 column

$c_1 := Ab_1 + c_1$

  Continue with

$$\left(\begin{array}{c|c} B_L & B_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right), \left(\begin{array}{c|c} C_L & C_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} C_0 & c_1 & C_2 \end{array} \right)$$

endwhile

Computing $C := AB$ by rows

$$\left(\begin{array}{c} \tilde{c}_0^T \\ \hline \tilde{c}_1^T \\ \hline \vdots \\ \hline \tilde{c}_{m-1}^T \end{array}\right) = C = AB = \left(\begin{array}{c} \tilde{a}_0^T \\ \hline \tilde{a}_1^T \\ \hline \vdots \\ \hline \tilde{a}_{m-1}^T \end{array}\right) B = \left(\begin{array}{c} \tilde{a}_0^T B \\ \hline \tilde{a}_1^T B \\ \hline \vdots \\ \hline \tilde{a}_{m-1}^T B \end{array}\right).$$

Algorithms for computing $C := AB + C$ by rows:

for $i = 0, \ldots, m-1$
  for $j = 0, \ldots, n-1$
    for $p = 0, \ldots, k-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

} $\tilde{c}_i^T := \tilde{a}_i^T B + \tilde{c}_i^T$

or

for $i = 0, \ldots, m-1$
  for $p = 0, \ldots, k-1$
    for $j = 0, \ldots, n-1$
      $\gamma_{i,j} := \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

} $\tilde{c}_i^T := \tilde{a}_i^T B + \tilde{c}_i^T$

Algorithm: $C := \text{GEMM\_UNB\_VAR2}(A, B, C)$

Partition $A \rightarrow \left(\begin{array}{c} A_T \\ \hline A_B \end{array}\right)$, $C \rightarrow \left(\begin{array}{c} C_T \\ \hline C_B \end{array}\right)$

where A_T has 0 rows, C_T has 0 rows

while $m(A_T) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c} A_T \\ \hline A_B \end{array}\right) \rightarrow \left(\begin{array}{c} A_0 \\ \hline a_1^T \\ \hline A_2 \end{array}\right), \left(\begin{array}{c} C_T \\ \hline C_B \end{array}\right) \rightarrow \left(\begin{array}{c} C_0 \\ \hline c_1^T \\ \hline C_2 \end{array}\right)$$

where a_1 has 1 row, c_1 has 1 row

$c_1^T := a_1^T B + c_1^T$

Continue with

$$\left(\begin{array}{c} A_T \\ \hline A_B \end{array}\right) \leftarrow \left(\begin{array}{c} A_0 \\ \hline a_1^T \\ \hline A_2 \end{array}\right), \left(\begin{array}{c} C_T \\ \hline C_B \end{array}\right) \leftarrow \left(\begin{array}{c} C_0 \\ \hline c_1^T \\ \hline C_2 \end{array}\right)$$

endwhile

Computing $C := AB$ via rank-1 updates

$$C = AB = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{k-1} \end{array} \right) \left(\begin{array}{c} \tilde{b}_0^T \\ \hline \tilde{b}_1^T \\ \hline \vdots \\ \hline \tilde{b}_{k-1}^T \end{array} \right) = a_0 \tilde{b}_0^T + a_1 \tilde{b}_1^T + \cdots + a_{k-1} \tilde{b}_{k-1}^T.$$

Algorithm for computing $C := AB + C$ via rank-1 updates:

for $p = 0, \ldots, k-1$
  for $j = 0, \ldots, n-1$
    for $i = 0, \ldots, m-1$
      $\gamma_{i,j} := \alpha_{i,p} \beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$\Big\}\ C := a_p \tilde{b}_p^T + C$

or

for $p = 0, \ldots, k-1$
  for $i = 0, \ldots, m-1$
    for $j = 0, \ldots, n-1$
      $\gamma_{i,j} := \alpha_{i,p} \beta_{p,j} + \gamma_{i,j}$
    endfor
  endfor
endfor

$\Big\}\ C := a_p \tilde{b}_p^T + C$

Algorithm: $C := \text{GEMM\_UNB\_VAR3}(A, B, C)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right), B \rightarrow \left(\begin{array}{c} B_T \\ \hline B_B \end{array} \right)$

where A_L has 0 columns, B_T has 0 rows

while $n(A_L) < n(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array} \right) \rightarrow \left(\begin{array}{c} B_0 \\ \hline b_1^T \\ \hline B_2 \end{array} \right)$$

where a_1 has 1 column, b_1 has 1 row

$C := a_1 b_1^T + C$

Continue with

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c} B_T \\ \hline B_B \end{array} \right) \leftarrow \left(\begin{array}{c} B_0 \\ \hline b_1^T \\ \hline B_2 \end{array} \right)$$

endwhile