Matrix Multiplication
Chapter I – Matrix Multiplication

By Gokturk Poyrazoglu

The State University of New York at Buffalo – BEST Group – Winter Lecture Series

Outline

1. Basic Algorithms and Notation
2. Structure and Efficiency
3. Block Matrices
4. Matrix-Vector Products
5. Parallel Matrix Multiplication

Matrix Notation

$$A \in \mathbb{R}^{m\times n} \quad \Longleftrightarrow \quad A = (a_{ij}) = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, \quad a_{ij} \in \mathbb{R}$$

- Matrix Operations
  - Transposition : $C = A^T \implies c_{ij} = a_{ji},$
  - Addition : $C = A + B \implies c_{ij} = a_{ij} + b_{ij},$
  - Scalar-matrix Multiplication : $C = \alpha A \implies c_{ij} = \alpha a_{ij},$
  - Matrix-matrix Multiplication : $C = AB \implies c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}.$
  - Pointwise Multiplication : $C = A .* B \implies c_{ij} = a_{ij} b_{ij}$
  - Pointwise Division : $C = A./B \implies c_{ij} = a_{ij}/b_{ij}.$

Vector Notation

- Column Vector :

$$x \in \mathbb{R}^n \quad \Longleftrightarrow \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad x_i \in \mathbb{R}\,.$$

- Row Vector :

$$x \in \mathbb{R}^{1 \times n} \quad \Longleftrightarrow \quad x = [x_1, \ldots, x_n].$$

Vector Operations

- Scalar-vector Multiplication : $z = ax \implies z_i = ax_i,$
- Vector Addition : $z = x + y \implies z_i = x_i + y_i,$
- Inner Product (dot product) : $c = x^T y \implies c = \sum_{i=1}^{n} x_i y_i.$
- Vector Update : $y = ax + y \implies y_i = ax_i + y_i$
- Pointwise Vector Multiplication : $z = x.*y \implies z_i = x_i y_i,$
- Pointwise Vector Division : $z = x./y \implies z_i = x_i / y_i.$

Dot Product Algorithm

- Dot product:

$$c = x^T y \quad \Longrightarrow \quad c = \sum_{i=1}^{n} x_i y_i .$$

- Algorithm :

$$
\begin{aligned}
&c = 0\\
&\textbf{for } i = 1{:}n\\
&\qquad c = c + x(i)y(i)\\
&\textbf{end}
\end{aligned}
$$

n multiplications

n additions

- The dot product operation is an *O(n)* operation.
- *The amount of work scales linearly with the dimension*

Saxpy (Vector update) Algorithm

- Vector update : $y = ax + y \implies y_i = ax_i + y_i$
- Algorithm :

$$\begin{array}{l} \textbf{for } i = 1{:}n \\ \qquad y(i) = y(i) + ax(i) \\ \textbf{end} \end{array}$$

- The vector update operation is also an *O(n)* operation.
- *The amount of work scales linearly with the dimension*

Matrix-vector Multiplication

$$y = y + Ax \quad \Rightarrow \quad y_i = y_i + \sum_{j=1}^{n} a_{ij} x_j, \qquad i = 1{:}m.$$

where y and x are vectors and A is a matrix

- Algorithm:

$$\begin{array}{l} \textbf{for } i = 1{:}m \\ \quad \textbf{for } j = 1{:}n \\ \quad\quad y(i) = y(i) + A(i,j)x(j) \\ \quad \textbf{end} \\ \textbf{end} \end{array}$$

- This algorithm is called Row-oriented gaxpy.
- This algorithm involves *O(nm)* work.

Matrix-vector Multiplication

$$y = y + Ax$$

where y and x are vectors and A is a matrix.

- 2^{nd} Approach : Column Oriented Gaxpy

Example :

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 1\cdot 7 + 2\cdot 8 \\ 3\cdot 7 + 4\cdot 8 \\ 5\cdot 7 + 6\cdot 8 \end{bmatrix} = 7\begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} + 8\begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 23 \\ 53 \\ 83 \end{bmatrix}$$

Algorithm :

```
for j = 1:n
    for i = 1:m
        y(i) = y(i) + A(i,j)·x(j)
    end
end
```

Interchanging the order of the loop

Partitioning a Matrix

- Consider a matrix A as a stack of row vectors

$$A \in \mathbb{R}^{m \times n} \quad \Longleftrightarrow \quad A = \begin{bmatrix} r_1^T \\ \vdots \\ r_m^T \end{bmatrix}, \quad r_k \in \mathbb{R}^n .$$

- Row Partition Example:
- A= $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$, so that $r_1^T = [\,1\ \ 2\,]$, $r_2^T = [\,3\ \ 4\,]$, $r_3^T = [\,5\ \ 6\,]$.
- New algorithm for vector update

$$\begin{array}{l} \textbf{for } i = 1{:}m \\ \qquad y_i = y_i + r_i^T x \\ \textbf{end} \end{array}$$

Partitioning a Matrix

- Column Partition Example
- A= $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$, so that $c_1 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$, $c_2 = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$
- Algorithm for vector update with column partition

for $j = 1{:}n$ → Count number of columns

$\quad y = y + x_j c_j$ → Multiply a scalar with a vector

end

Colon Notation

- k^{th} row of matrix A:

$$A(k,:) = [a_{k1}, \ldots, a_{kn}] .$$

- k^{th} column of matrix A:

$$A(:,k) = \begin{bmatrix} a_{1k} \\ \vdots \\ a_{mk} \end{bmatrix} .$$

- Rewrite the algorithm with row partitioning

$$\begin{array}{l} \textbf{for } i = 1{:}m \\ \quad y_i = y_i + \underline{r_i^T} x \\ \textbf{end} \end{array} \Rightarrow \begin{array}{l} \textbf{for } i = 1{:}m \\ \quad y(i) = y(i) + \underline{A(i,:)} \cdot x \\ \textbf{end} \end{array}$$

Outer Product Update

- Update matrix A with multiplication of two vectors

$$A = A + xy^T, \qquad A \in \mathbb{R}^{m\times n},\ x \in \mathbb{R}^m,\ y \in \mathbb{R}^n.$$

- Algorithm for outer product update:

```
for i = 1:m
    for j = 1:n
        a_ij = a_ij + x_i y_j
    end
end
```

- The algorithm involves *O(nm)* operations.

Outer Product Update with Vector Mult.

for $i = 1{:}m$
  for $j = 1{:}n$
    $a_{ij} = a_{ij} + x_i y_j$
  end
end

Row update

for $i = 1{:}m$
  $A(i,:) = A(i,:) + x(i) \cdot y^T$
end

Column Update

for $j = 1{:}n$
  $A(:,j) = A(:,j) + y(j) \cdot x$
end

Matrix – Matrix Multiplication

- Dot Product

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{bmatrix}$$

- Linear Combination of left matrix columns

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

- Sum of Outer Products

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \begin{bmatrix} 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \end{bmatrix} \begin{bmatrix} 7 & 8 \end{bmatrix}$$

Matrix – Matrix Multiplication

- Consider a matrix update by matrix multiplication

$$C = C + AB, \qquad C \in \mathbb{R}^{m\times n},\ A \in \mathbb{R}^{m\times r},\ B \in \mathbb{R}^{r\times n}.$$

- Triply Nested Loop Algorithm

```
for i = 1:m
    for j = 1:n
        for k = 1:r
            C(i,j) = C(i,j) + A(i,k)·B(k,j)
        end
    end
end
```

- This algorithm involves *O(mnr)* operations.

Dot Product Matrix Multiplication

for $i = 1{:}m$
  for $j = 1{:}n$
    $C(i,j) = C(i,j) + A(i,:)\cdot B(:,j)$
  end
end

for $i = 1{:}m$
  for $j = 1{:}n$
    $c_{ij} = c_{ij} + a_i^T b_j$
  end
end

for $i = 1{:}m$
  $c_i^T = c_i^T + a_i^T B$
end

for $i = 1{:}m$
  $C(i,:) = C(i,:) + A(i,:)\cdot B$
end

Saxpy Formulation

- Suppose matrix-A and C are column partitioned

$$A = [\, a_1 \,|\, \cdots \,|\, a_r \,], \qquad C = [\, c_1 \,|\, \cdots \,|\, c_n \,].$$

- Each column of C can be updated as follows :

$$c_j = c_j + \sum_{k=1}^{r} a_k b_{kj}, \qquad j = 1{:}n.$$

Algorithm :

$$\begin{array}{l} \textbf{for } j = 1{:}n \\ \quad \textbf{for } k = 1{:}r \\ \quad\quad C(:,j) = C(:,j) + A(:,k)\cdot B(k,j) \\ \quad \textbf{end} \\ \textbf{end} \end{array}$$

Simplified Algorithm:

$$\begin{array}{l} \textbf{for } j = 1{:}n \\ \quad C(:,j) = C(:,j) + AB(:,j) \\ \textbf{end} \end{array}$$

Flops

Flop is a measure of;

1. Addition
2. Subtraction
3. Multiplication
4. Division

Operation	Dimension	Flops
$\alpha = x^T y$	$x, y \in \mathbb{R}^n$	$2n$
$y = y + ax$	$a \in \mathbb{R}$, $x, y \in \mathbb{R}^n$	$2n$
$y = y + Ax$	$A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$	$2mn$
$A = A + yx^T$	$A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$	$2mn$
$C = C + AB$	$A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{r \times n}$, $C \in \mathbb{R}^{m \times n}$	$2mnr$

Complex Matrices

- Consider matrix A:

$$A = B + iC \in \mathbb{C}^{m\times n},$$

- B is the real part of A;
- C is the imaginary part of A
- Matrix Operations:
  - Transposition of A becomes *conjugate transposition.*

$$\boxed{C = A^H \quad \Longrightarrow \quad c_{ij} = \bar{a}_{ji}.}$$

Structure and Efficiency

Outline

1. Band Matrices
2. Triangular Matrices
3. Diagonal Matrices
4. Symmetric Matrices
5. Permutation Matrices

Band Matrices

- Consider a matrix A;

$$A = \begin{bmatrix} \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & 0 \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

- Matrix A has *lower bandwidth p* if $a_{ij}=0$ when $i > j + p$
- Matrix A has *upper bandwidth q* if $a_{ij}=0$ when $j > i + q$

Triangular Matrix Multiplication

- Consider a matrix update by matrix multiplication

$$AB = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ 0 & a_{22}b_{22} & a_{22}b_{23} + a_{23}b_{33} \\ 0 & 0 & a_{33}b_{33} \end{bmatrix}.$$

- So the update takes the form of

$$\boxed{c_{ij} = c_{ij} + \sum_{k=i}^{j} a_{ik} b_{kj}}$$

- k starts from i to j so we ignore the zero elements of [AB]

Colon Notation Triangular Matrix Mult.

for $i = 1{:}n$
  for $j = i{:}n$
    for $k = i{:}j$
      $C(i,j) = C(i,j) + A(i,k) \cdot B(k,j)$
    end
  end
end

for $i = 1{:}n$
  for $j = i{:}n$
    $C(i,j) = C(i,j) + A(i,i{:}j) \cdot B(i{:}j,j)$
  end
end

Diagonal Matrices

- Lower and upper bandwidth is Zero for all diagonal matrices.
- Notation :

$$D = \mathrm{diag}(d_1, \ldots, d_q), \quad q = \min\{m, n\} \quad \Longleftrightarrow \quad d_i = d_{ii}.$$

- Pre-multiplication of D scales rows of A

$$B = DA \quad \Longleftrightarrow \quad B(i,:) = d_i \cdot A(i,:),\ i = 1{:}m$$

- Post-multiplication of D scales columns of A

$$B = AD \quad \Longleftrightarrow \quad B(:,j) = d_j \cdot A(:,j),\ j = 1{:}n.$$

Symmetry

$$A^T = A$$

Symmetric: $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$,

$$A^H = A$$

Hermitian: $\begin{bmatrix} 1 & 2-3i & 4-5i \\ 2+3i & 6 & 7-8i \\ 4+5i & 7+8i & 9 \end{bmatrix}$,

$$A^T = -A$$

Skew-Symmetric: $\begin{bmatrix} 0 & -2 & 3 \\ 2 & 0 & -5 \\ -3 & 5 & 0 \end{bmatrix}$,

$$A^H = -A$$

Skew-Hermitian: $\begin{bmatrix} i & -2+3i & -4+5i \\ 2+3i & 6i & -7+8i \\ 4+5i & 7+8i & 9i \end{bmatrix}$

Identity and Permutation Matrix

- Identity Matrix:

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- Permutation Matrix

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Famous Permutation Matrices

- Exchange Permutation

$$y = \mathcal{E}_4 x = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_4 \\ x_3 \\ x_2 \\ x_1 \end{bmatrix}$$

- Downshift Permutation

$$y = \mathcal{D}_4 x = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_4 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Block Matrix Terminology

- Special case of column and row partitioning.

$$A = \begin{array}{cc} \left[\begin{array}{ccc} A_{11} & \ldots & A_{1r} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qr} \end{array}\right] & \begin{array}{c} m_1 \\ \\ m_q \end{array} \\ \begin{array}{ccc} n_1 & \quad\quad & n_r \end{array} & \end{array}$$

- Examples

$$\mathrm{diag}(A_{11}, A_{22}, A_{33}) = \begin{bmatrix} A_{11} & 0 & 0 \\ 0 & A_{22} & 0 \\ 0 & 0 & A_{33} \end{bmatrix}$$

$$L = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix}, \quad U = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}, \quad T = \begin{bmatrix} T_{11} & T_{12} & 0 \\ T_{21} & T_{22} & T_{23} \\ 0 & T_{32} & T_{33} \end{bmatrix}$$

Block Matrix Operations

- Scalar Multiplication :

$$\mu \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{bmatrix} = \begin{bmatrix} \mu A_{11} & \mu A_{12} \\ \mu A_{21} & \mu A_{22} \\ \mu A_{31} & \mu A_{32} \end{bmatrix}$$

- Transposition :

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{bmatrix}^T = \begin{bmatrix} A_{11}^T & A_{21}^T & A_{31}^T \\ A_{12}^T & A_{22}^T & A_{32}^T \end{bmatrix}$$

- Addition:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{bmatrix} + \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix} = \begin{bmatrix} A_{11} + B_{11} & A_{12} + B_{12} \\ A_{21} + B_{21} & A_{22} + B_{22} \\ A_{31} + B_{31} & A_{32} + B_{32} \end{bmatrix}$$

Block Matrix Multiplication

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22} \\ A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22} \\ A_{31}B_{11}+A_{32}B_{21} & A_{31}B_{12}+A_{32}B_{22} \end{bmatrix}$$

- Column dimensions of A

  must match with

  row dimensions of B.

Block Vector Update

- Consider Block Matrix A and a block vector y

$$A = \begin{array}{c} \begin{bmatrix} A_1 \\ \vdots \\ A_q \end{bmatrix} \end{array} \begin{array}{c} m_1 \\ \\ m_q \end{array}$$

$$y = \begin{array}{c} \begin{bmatrix} y_1 \\ \vdots \\ y_q \end{bmatrix} \end{array} \begin{array}{c} m_1 \\ \\ m_q \end{array}$$

- Block Vector update :

$$\begin{bmatrix} y_1 \\ \vdots \\ y_q \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_q \end{bmatrix} + \begin{bmatrix} A_1 \\ \vdots \\ A_q \end{bmatrix} x$$

- Algorithm :

$\alpha = 0$
for $i = 1{:}q$
$\quad idx = \alpha+1 : \alpha+m_i$
$\quad y(idx) = y(idx) + A(idx,:)\cdot x$
$\quad \alpha = \alpha + m_i$
end

Block Matrix Update

- Consider a matrix update by matrix multiplication.

$$C = C + AB$$

$A = (A_{\alpha\beta})$, $B = (B_{\alpha\beta})$, and $C = (C_{\alpha\beta})$ as N-by-N block matrices with ℓ-by-ℓ blocks.

$$C_{\alpha\beta} = C_{\alpha\beta} + \sum_{\gamma=1}^{N} A_{\alpha\gamma} B_{\gamma\beta}, \qquad \alpha = 1{:}N, \quad \beta = 1{:}N.$$

- Algorithm:

```
for α = 1:N
    i = (α − 1)ℓ + 1:αℓ
    for β = 1:N
        j = (β − 1)ℓ + 1:βℓ
        for γ = 1:N
            k = (γ − 1)ℓ + 1:γℓ
            C(i, j) = C(i, j) + A(i, k)·B(k, j)
        end
    end
end
```

Complex Matrix Multiplication

- Consider matrix A, B, and C are complex matrices where all matrices are real and $i^2 = -1$

$$\boxed{C_1 + iC_2 = (C_1 + iC_2) + (A_1 + iA_2)(B_1 + iB_2)}$$

- Multiplication:

$$\begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} + \begin{bmatrix} A_1 & -A_2 \\ A_2 & A_1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

- Note: Complex Matrix Multiplication has expanded dimension.

Hamiltonian Matrix

- Form :

$$M = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix}$$

1. A, F, and G are square matrices
2. F and G are symmetric matrices

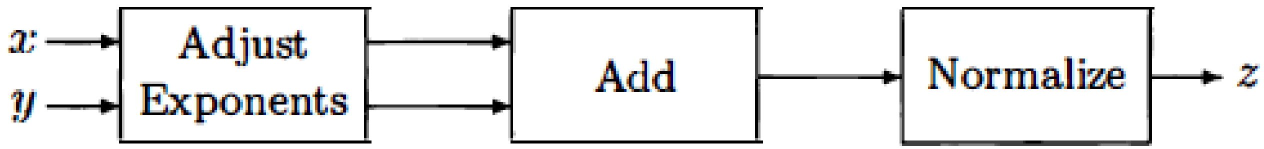
Hamiltonian Check

1. Consider a permutation matrix J as $\boxed{J = \begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix}}$

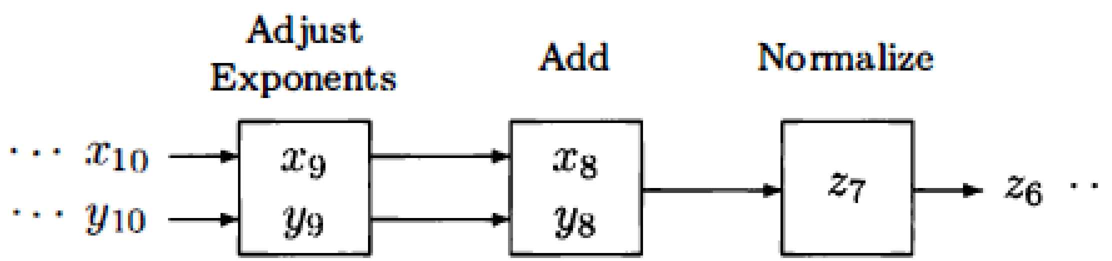
2. If $JMJ^T = -M^T,$ then M is Hamiltonian Matrix.

Vector Processing

- 3-cycle Adder



- Pipelined Addition for a vector operation



Data Motion in Computation Performance

Other than flop counts;

Data motion is also an important factor

when reasoning about performance.

$$
\begin{aligned}
&\mathit{first} = 1\\
&\textbf{while } \mathit{first} \le n\\
&\qquad \mathit{last} = \min\{n, \mathit{first} + v_L - 1\}\\
&\qquad \text{Vector load: } r_1 \leftarrow x(\mathit{first}{:}\mathit{last})\\
&\qquad \text{Vector load: } r_2 \leftarrow y(\mathit{first}{:}\mathit{last})\\
&\qquad \text{Vector add: } r_1 = r_1 + r_2\\
&\qquad \text{Vector store: } z(\mathit{first}{:}\mathit{last}) \leftarrow r_1\\
&\qquad \mathit{first} = \mathit{last} + 1\\
&\textbf{end}
\end{aligned}
$$

Load/Store Operations

Vector Update (Gaxpy)

$$\boxed{y = y + Ax}$$

Load / Store Operations: (3+n)

$$
\begin{aligned}
& r_x \leftarrow x \\
& r_y \leftarrow y \\
& \textbf{for } j = 1{:}n \\
& \qquad r_a \leftarrow A(:,j) \\
& \qquad r_y \ = r_y + r_a r_x(j) \\
& \textbf{end} \\
& y \leftarrow r_y
\end{aligned}
$$

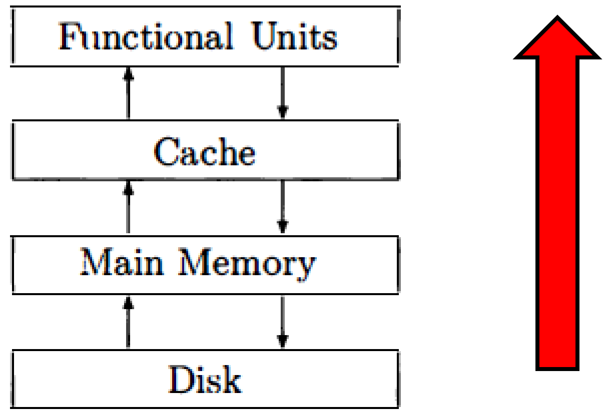
Outer Product Matrix Update

$$\boxed{A = A + yx^T.}$$

Load / Store Operations: (2+2n)

$$
\begin{aligned}
& r_x \leftarrow x \\
& r_y \leftarrow y \\
& \textbf{for } j = 1{:}n \\
& \qquad r_a \leftarrow A(:,j) \\
& \qquad r_a \ = r_a + r_y r_x(j) \\
& \qquad A(:,j) \leftarrow r_a \\
& \textbf{end}
\end{aligned}
$$

Memory Hierarchy



- Each level has a limited capacity.
- There is a cost associated with moving a data between two levels.
- Efficient Implementation should consider data flow between the various levels.

Parallel Matrix Multiplication

- Design of a parallel procedure begins with the breaking up of the given problem into smaller parts that exhibit a measure of INDEPENDENCE.
- Consider block matrices A, B, and C

$$C = \begin{bmatrix} C_{11} & \cdots & C_{1N} \\ \vdots & \ddots & \vdots \\ C_{M1} & \cdots & C_{MN} \end{bmatrix}, \; A = \begin{bmatrix} A_{11} & \cdots & A_{1R} \\ \vdots & \ddots & \vdots \\ A_{M1} & \cdots & A_{MR} \end{bmatrix}, \; B = \begin{bmatrix} B_{11} & \cdots & B_{1N} \\ \vdots & \ddots & \vdots \\ B_{R1} & \cdots & B_{RN} \end{bmatrix}$$

- The task is to compute:

$$\text{Task}(i,j): \qquad C_{ij} = C_{ij} + \sum_{k=1}^{R} A_{ik} B_{kj}.$$

Data Motion in Parallel Multiplication

- In a parallel computing environment, the data that a processor needs can be "far away", and if that is the case too often, then it is possible to lose the multiprocessor advantage.
- Time spent waiting for another processor to finish a task is TIME LOST.