# Lecture 6: Matrix Multiplication

Lecture Outline

(Reading: Chapter 7 of textbook)

1. Introduction
2. Algorithms on processor arrays
3. Algorithms for multiprocessors

1. Introduction

As sorting and searching, matrix multiplication is fundamental components of many numerical and nonnumerical algorithms.

(1) **Sequential matrix multiplications**. Let $A$ be a $l \times m$ matrix, and $B$ be a $m \times n$ matrix

```
MATRIX MULTIPLICATION (SISD)
begin
    for i ← 0 to l − 1 do
        for j ← 0 to n − 1 do
            t ← 0
            for k ← 0 to m − 1 do
                t ← t + a_{i,k} × b_{k,j}
            endfor
            c_{i,j} ← t
        endfor
    endfor
end
```

(2) **Time complexity.** $l \times m \times n$ addition and multiplication operations. Time complexity of multiplying two $n \times n$ matrices is $\Theta(n^3)$. Faster sequential algorithms exist (Strassen's algorithm has time complexity $O(n^{\log 7})$, $\log 7 \approx 2.81$).

2. Algorithms on processor arrays

(1) **Matrix multiplication on SIMD-MC$^2$ model**

a. **A $\Omega(n)$ lower bound**

(a) **Definition.** Given a data item originally available at a single processor in some model of parallel computation, let the function $\sigma(k)$ be the maximum number of processors to which the data can be transmitted in $k$ or fewer data routing steps.

(b) **Example.** In the SIMD-MC$^2$ model, $\sigma(0) = 1$, $\sigma(1) = 5$, $\sigma(2) = 13$. In general $\sigma(k) = 2k^2 + 2k + 1$.

(c) **Lemma.** (Gentleman [1978]) Suppose that two $n \times n$ matrices $A$ and $B$ are to be multiplied, and assume that every element of $A$ and $B$ is stored exactly once and that no processor contains more than one element of either matrix. If we ignore any data broadcasting facility, multiplying $A$ and $B$ to produce the $n \times n$ matrix $C$ requires at least $s$ data routing steps, where $\sigma(2s) \geq n^2$.

**Proof:**

· Consider an arbitrary element $c_{i,j}$ of the product matrix. This element is the inner product of row $i$ of matrix $A$ and column $j$ of matrix $B$. There must be a path from the processors where each of these elements is stored to the processors where the result $c_{i,j}$ is stored. Let $s$ be the length of the longest such path. The creation of matrix product $C$ takes at least $s$ data routing steps.

· Note that these paths also can be used to define a set of paths of length at most $2s$ from any element $b_{u,v}$ to every element $a_{i,j}$, where $0 \leq i, j \leq n-1$. This is because there is a path of length at most $s$ from $b_{u,v}$ to the processor where $c_{i,v}$ is found, and there is also a path of length at most $s$ from $a_{i,j}$ to $c_{i,v}$ (Fig.6-1b). Hence there is a path of length at most $2s$ from any element $b_{u,v}$ to every element $a_{i,j}$. Similarly, these paths define a set of paths of length at most $2s$ from any element $a_{u,v}$ to every element $b_{i,j}$, $0 \leq i, j \leq n-1$.

· The $n^2$ elements of $A$ are stored in unique processors. Since there exist paths of length at most $2s$ from the processor storing $b_{u,v}$ to the processors storing the elements of $A$, it follows from the definition of $\sigma$ that $\sigma(2s) \geq n^2$. □

(d) **Theorem.** (Gentleman [1978]) Matrix multiplication on the SIMD-MC$^2$ model requires $\Omega(n)$, or for large values of $n$, approximately $s \geq 0.35n$, data routing steps.

**Proof:**

· From the previous lemma we have $\sigma(2s) \geq n^2$. Because on the SIMD-MC$^2$ model, $\sigma(k) = 2k^2 + 2k + 1$, we have $\sigma(2s) = 2(2s)^2 + 2(2s) + 1$. Therefore we have

$$\begin{aligned} & 8s^2 + 4s + 1 \geq n^2 \\ \Rightarrow \quad & s^2 + \frac{s}{2} + \frac{1}{8} \geq \frac{n^2}{8} \\ \Rightarrow \quad & (s + \frac{1}{4})^2 + \frac{1}{16} \geq \frac{n^2}{8} \end{aligned}$$

$$\Rightarrow \quad s \geq \frac{\sqrt{2n^2-1}}{4} - \frac{1}{4}$$

b. **An optimal algorithm.** Given an SIMD-MC$^2$ model with wraparound connections, we have an algorithm that performs matrix multiplication of two $n \times n$ matrix in time $\Theta(n)$ using $n^2$ processors.

(a) **Basic ideas.** To acheive $\Theta(n)$ time complexity, we should have at most $\Theta(n)$ iterations. During each iterations, we should let all the $n^2$ processors be active to perform $n^2$ multiplications and additions (Fig.7-3,7-4,7-5,p.182,183,184).

(b) **The algorithm** (Fig.7-5,p.184)

MATRIX MULTIPLICATION (SIMD-MC$^2$)
{ $A(i,j)$ contains $a_{i,j}$, $B(i,j)$ contains $b_{i,j}$, and $C(i,j)$ contains $c_{i,j}$, $P(i,j)$ }
{ denote the processor at row $i$, column $j$ }
**begin**
{ Stagger matrices }
**for** $k \leftarrow 1$ **to** $n-1$ **do**
**for all** $P(i,j)$ **do**
**if** $i > k$ **then**
$A(i,j) \Leftarrow A(i,j+1)$
**endif**
**if** $j > k$ **then**
$B(i,j) \Leftarrow B(i+1,j)$
**endif**
**endfor**
**endfor**
{ Compute the dot products }
**for all** $P(i,j)$ **do**
$C(i,j) \leftarrow A(i,j) \times B(i,j)$
**endfor**
**for** $k \leftarrow 1$ **to** $n-1$ **do**
**for all** $P(i,j)$ **do**
$A(i,j) \Leftarrow A(i,j+1)$
$B(i,j) \Leftarrow B(i+1,j)$
$C(i,j) \leftarrow C(i,j) + A(i,j) \times B(i,j)$
**endif**
**endfor**
**end**

(2) **Matrix Multiplication on the SIMD-CC Model**

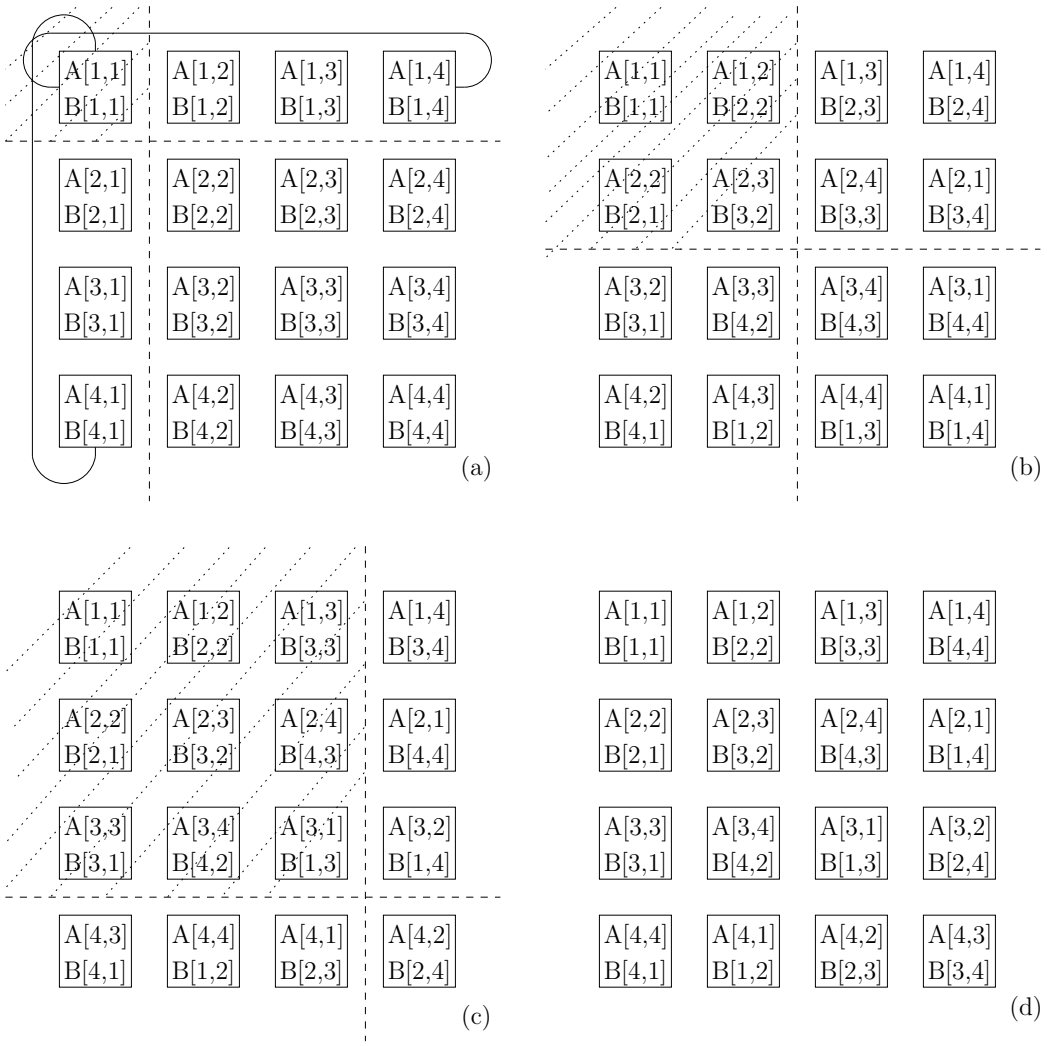


Figure 76: Stagging elements of two $4 \times 4$ arrays $A$ and $B$

a. **Theorem.** (Dekel, Nassimi, and Sahni [1981]) Given the SIMD-CC model with $n^3 = 2^{3q}$ processors, two $n \times n$ matrices can be multiplied in $\Theta(\log n)$ time

   (a) Note: when $n = 2, q = 1$; $n = 4, q = 2$; $n = 8, q = 3$, and $\cdots$.

   (b) Fig.7-9,p.187 for an example of $n = 2, q = 1$.

b. Each processor $P(x)$ of the $n^3$ processors has local variables $a, b, c, s$, and $t$.

c. Initially the matrix elements $a_{i,j}$ and $b_{i,j}$ $(0 \leq i, j \leq n-1)$ are stored in variables of $a$ and $b$ of processor $P(2^q i + j)$. After the completion of the algorithm, matrix element $c_{i,j}$ $(0 \leq i, j \leq n-1)$ is stored in variable $c$ of processor $P(2^q i + j)$.

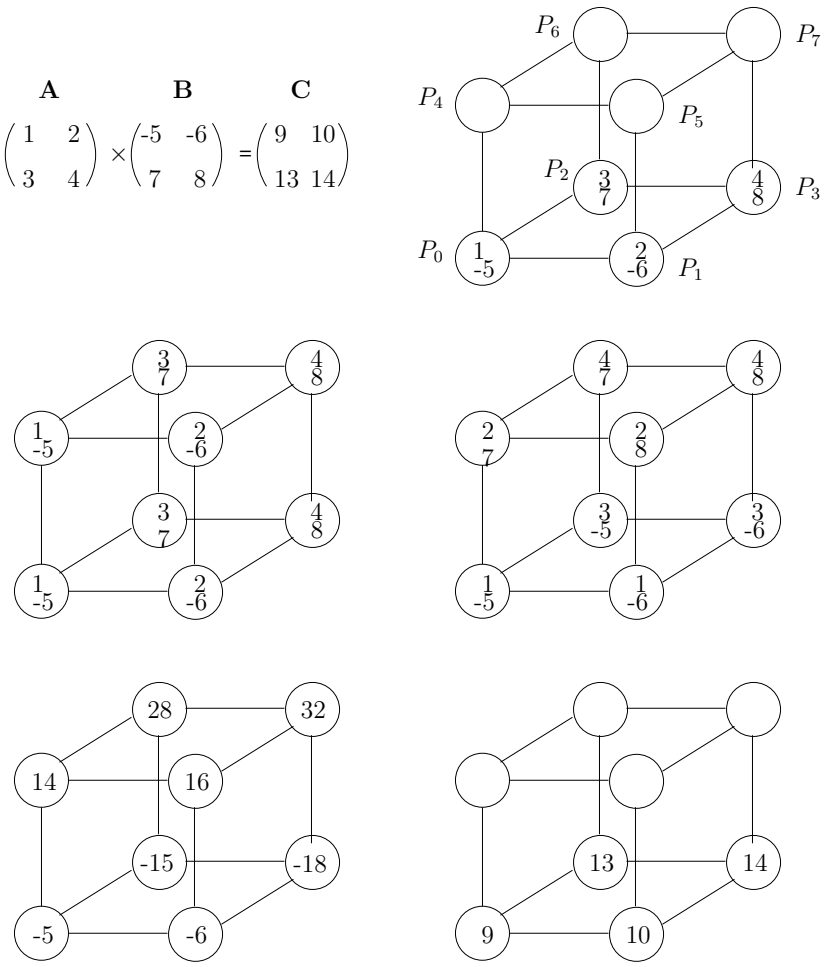


Figure 77: Matrix multiplication on the hypercube SIMD model. (Quinn and Deo 1984). Copyright @1986 Assocation for Computing Machinery. Reprinted by permission (Fig.7-9, p.187)

d. Two new functions, **BIT** and **BIT_COMPLEMENT**, are used. Function BIT$(m, l)$ returns the value of the $l^{th}$ bit (counting from the right to left, i.e. from least significant to most significant bit. The value of $l$ starts from 0, which means the first bit or the least significant bit.) in the binary representation of $m$. Function BIT_COMPLEMENT$(m, l)$ returns the value of the integer formed by complementing the $l^{th}$ bit in the binary representation of $m$.

e. The algorithm has three distinct phases:

**Phase 1** is a routing phase, taking $4q$ routing steps. There are 3 **for** loops. After the 1st for loop,

$$\left.\begin{array}{r} [2^{2q}k + 2^q i + j]a = a_{i,j} \\ [2^{2q}k + 2^q i + j]b = b_{i,j} \end{array}\right\} \text{ for } \ 0 \le k \le n-1$$

After the 2nd for loop,

$$[2^{2q}k + 2^q i + j]a = a_{i,k} \text{ for } 0 \leq j \leq n-1$$

After the 3rd for loop,

$$[2^{2q}k + 2^q i + j]b = b_{k,j} \text{ for } 0 \leq i \leq n-1$$

**Phase 2** all $n^3$ processors perform $a_{i,k} \times b_{k,j}$ simultaneously, taking 1 time units.

**Phase 3** The products are routed and sumarized, taking $q$ routing steps and $q$ addition operations.

```
MATRIX MULTIPLICATION (HYPERCUBE SIMD)
Parameter:  q        {Matrix size is 2^q × 2^q}
Glogal:     l
Local:      a, b, c, s, t
begin
    { Phase 1: Broadcast matrices A and B }
    for l ← 3q − 1 downto 2q do
        for all P_m, where BIT(m, l) = 1 do
            t ← BIT_COMPLEMENT(m, l)
            a ⇐ [t]a
            b ⇐ [t]b
        endfor
    endfor
    for l ← q − 1 downto 0 do
        for all P_m, where BIT(m, l) ≠ BIT(m, 2q + l) do
            t ← BIT_COMPLEMENT(m, l)
            a ⇐ [t]a
        endfor
    endfor
    for l ← 2q − 1 downto q do
        for all P_m, where BIT(m, l) ≠ BIT(m, q + l) do
            t ← BIT_COMPLEMENT(m, l)
            b ⇐ [t]b
        endfor
    endfor
    { Phase 2: Do the multiplications in parallel }
    for all P_m do
        c ← a × b
    endfor
```

```
    { Phase 3: Sum the products }
    for l ← 2q to 3q − 1 do
        for all P_m do
            t ← BIT_COMPLEMENT(m, l)
            s ⇐ [t]c
            c ← c + s
        endfor
    endfor
end
```

**Fig.7-8** Matrix multiplication on the hypercube SIMD model (Fig.7-8, p.186)

f. The time complexity is $\Theta(q) = \Theta(\log n)$ (5 routing steps, one multiplication operations, $q$ addition operations).

(3) **Matrix Multiplication on the SIMD-PS Model Theorem.** (Dekel, Nassimi, and Sahni [1981]) Given the SIMD-PS model with $n^3 = 2^{3q}$ processors, two $n \times n$ matrices can be multiplied in $\Theta(\log n)$ time.

**Proof:** The algorithm for SIMD-PS model simulates the routes followed in the algorithm for SIMD-CC model. In $13 \log n$ routing steps the SIMD-PS model can perform the same routings done by SIMD-CC in time $5 \log n$ steps. □

3. Algorithms for multiprocessors

(1) *Grain size*: The amount of work performed between processor interactions.

- Goal: minimize the overhead through parallelization, grain size should be maximized whenever possible.
- **Design Strategy 5**. If load balancing is not a problem, maximize grain size.

(2) Parallel matrix algo on UMA multiprocessors

- Either the $j$ loop or the $i$ loop can be parallelized (The only data dependency is the inner for loop).
- If the $j$ loop is parallelized, the algo will execute $n$ synchronizations (one per iteration of the $i$ loop). The grain size is $O(n^2/p)$.
- If the $i$ loop is parallelized, the algo will execute only one synchronization. The grain size is $O(n^3/p)$.
- On most UMA multiprocessor the parallelized loop $i$ version execute fast.

(3) The code (Fig.7-10,p.188).
The variables $i, j, k$, and $t$ represent scalars local to each process.

```
MATRIX MULTIPLICATION(UMA MULTIPROCESSOR):
Global:   n                          {Dimension of matrices}
          a[0..(n-1)][0..(n-1)]      {First factor matrix}
          b[0..(n-1)][0..(n-1)]      {Second factor matrix}
          c[0..(n-1)][0..(n-1)]      {Product matrix}
Local:    i, j, k                    {Loop indices}
          t                          {Accumulates subtotal}
begin
   for all P_m, where 1 ≤ m ≤ p do
      for i ← m to n step p do
         for j ← 1 to n do
            t ← 0
            for k ← 1 to n do
               t ← t + a[i][k] × b[k][j]
            endfor
            c[i][j] ← t
         endfor
      endfor
   endfor
end
```

**Fig.7-10** Matrix multiplication algorithm written for a UMA multiprocessor.

(4) The complexity: Each process calculates $n/p$ rows of matrix $C$; the time needed to calculate a single row is $\Theta(n^2)$.
The processes synchronize exactly once (hence the syn overhead is $\Theta(p)$).
$\Theta(n^3/p + p)$ is the complexity.

(5) Speedup (Fig.7-11,p.189):

a. If memory contention is ignored, speed up is almost linear.

b. If the memory cell is of equal distance to every processor, memory access time can be ignored.

(6) Ignore memory access time in a loosely coupled multiprocessor system can be a problem.

a. In this case, memory reference should be kept as local as possible.

b. The above algo does not keep good locality.

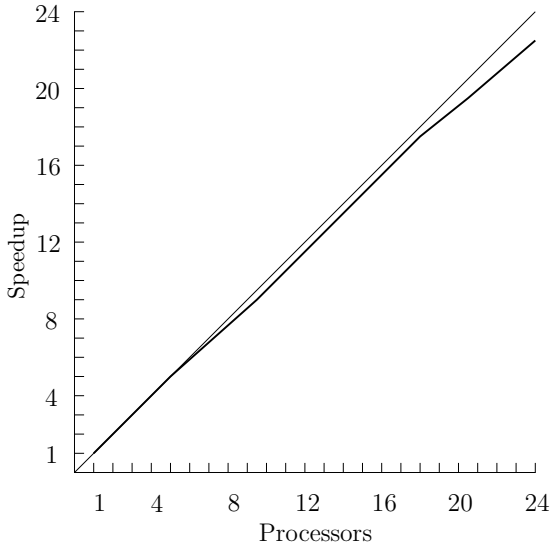


Figure 78: Speedup of parallel matrix multiplication algorithm on lightly loaded Sequent Symmetry. The algorithm is multiplying two $256 \times 256$ floating point matrices (Fig.7-11, p.189)

(7) The method based on **block matrix multiplication** (p.189).

a. Processes are assigned to do the block matrix multiplications

(a) The number of multiplications and additions per matrix-element fetch increases.

(b) Example (Fig.7-12,p.190 showed an illustration): $n = 2k$, $p = (n/k)^2$ processes.

· The matrices $A$ and $B$ are divided into $p$ blocks of size $k \times k$.

· Each block multiplications requires $2k^2$ matrix fetches, $k^3$ additions, and $k^3$ multiplications. The number of arithmetic operations per access has increased from 2 to $k/\sqrt{p}$

b. The method has been implemented on a Cm* for various sizes. Fig.7-13,p.191 shows the speedup (Note: the speedup is drawn as a linear function of the number of processors. Previously we have seen many *logarithmic* drawing).

(8) **Design Strategy 6**. Reduce average memory latency time by increasing locality.

$$\begin{array}{ccccc} \text{A} & & \text{B} & & \text{C} \\ \begin{pmatrix} 1 & 0 & 2 & 3 \\ 4 & -1 & 1 & 5 \\ -2 & -3 & -4 & 2 \\ -1 & 2 & 0 & 0 \end{pmatrix} & \begin{pmatrix} -1 & 1 & 2 & -3 \\ -5 & -4 & 2 & -2 \\ 3 & -1 & 0 & 2 \\ 1 & 0 & 4 & 5 \end{pmatrix} & = & \begin{pmatrix} 8 & -1 & 14 & 16 \\ 9 & 7 & 26 & 17 \\ 7 & 14 & -2 & 14 \\ -9 & -9 & 2 & -1 \end{pmatrix} \end{array}$$

STEP 1: Compute $C_{i,j} = A_{i,1}B_{1,j}$

$$\left(\begin{array}{cc:cc} 1 & 0 & 2 & -3 \\ 4 & -1 & 2 & -2 \\ \hdashline -2 & -3 & 2 & -3 \\ -1 & 2 & 2 & -2 \end{array}\right) \left(\begin{array}{cc:cc} -1 & 1 & 2 & -3 \\ -5 & -4 & 2 & -2 \\ \hdashline -1 & 1 & -10 & 12 \\ -5 & -4 & 2 & -1 \end{array}\right) = \left(\begin{array}{cc:cc} -1 & 1 & 2 & -3 \\ 1 & 8 & 6 & -10 \\ \hdashline 17 & 10 & -10 & 12 \\ -9 & -9 & 2 & -1 \end{array}\right)$$

STEP 2: Compute $C_{i,j} = C_{i,j} + A_{i,2}B_{2,j}$

$$\left(\begin{array}{cc:cc} 2 & 3 & 0 & 2 \\ 1 & 5 & 4 & 5 \\ \hdashline -4 & 2 & 8 & -1 \\ 0 & 0 & 4 & 5 \end{array}\right) \left(\begin{array}{cc:cc} 3 & -1 & 14 & 16 \\ 1 & 0 & 26 & 17 \\ \hdashline 3 & -1 & -2 & 14 \\ 1 & 0 & 2 & -1 \end{array}\right) = \left(\begin{array}{cc:cc} 8 & -1 & 14 & 16 \\ 9 & 7 & 26 & 17 \\ \hdashline 7 & 14 & -2 & 14 \\ -9 & -9 & 2 & -1 \end{array}\right)$$

Figure 79: Block-matrix approach to parallel-matrix multiplication (Fig.7-12, p.190)

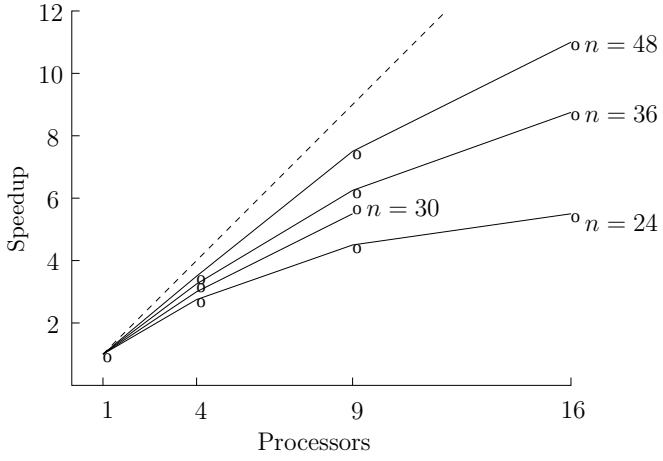


Figure 80: Speedup of matrix multiplication algorithm on $Cm^*$ (Fig.7-13, p.191)