Parallel Programming Application: Matrix Multiplication

(senol@be.itu.edu.tr)
Outline

• Matrix Multiplication (BLAS3 Operation)
• Cannon Algorithm
• Fox Algorithm
Matrix Multiplication (BLAS3)

A and B nxn matrices => C=AXB

```c
void Serial_mat_mult(mt_A,mt_B,mt_C,int n)
{
    int i,j,k;
    for (i=0;i<n;i++){
        for (j=0;j<n;j++){
            C[i][j]=0.0;
            for (k=0;k<n;k++)
                C[i][j] = C[i][j] + A[i][k]*B[k][j];
        }
    }
}
```

- This algorithm requires $n^3$ multiplications and $n^3$ additions, leading to a sequential time complexity of $O(n^3)$
Serial Matrix Multiplication

- During the first iteration of loop variable $i$ the first matrix $A$ row and all the columns of matrix $B$ are used to compute the elements of the first result matrix $C$ row
Partitioning Matrices: Block Stripping

- Uniform block-striped partitioning of 16 x 16 matrix on 4 processors
### Partitioning Matrices: Checkerboard

(a) **Block-checkboard mapping**

(b) **Cyclic-checkboard partitioning**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0</td>
<td>0, 1</td>
<td>0, 2</td>
<td>0, 3</td>
<td>0, 4</td>
<td>0, 5</td>
<td>0, 6</td>
</tr>
<tr>
<td>1, 0</td>
<td>1, 1</td>
<td>1, 2</td>
<td>1, 3</td>
<td>1, 4</td>
<td>1, 5</td>
<td>1, 6</td>
</tr>
<tr>
<td>2, 0</td>
<td>2, 1</td>
<td>2, 2</td>
<td>2, 3</td>
<td>2, 4</td>
<td>2, 5</td>
<td>2, 6</td>
</tr>
<tr>
<td>3, 0</td>
<td>3, 1</td>
<td>3, 2</td>
<td>3, 3</td>
<td>3, 4</td>
<td>3, 5</td>
<td>3, 6</td>
</tr>
<tr>
<td>4, 0</td>
<td>4, 1</td>
<td>4, 2</td>
<td>4, 3</td>
<td>4, 4</td>
<td>4, 5</td>
<td>4, 6</td>
</tr>
<tr>
<td>5, 0</td>
<td>5, 1</td>
<td>5, 2</td>
<td>5, 3</td>
<td>5, 4</td>
<td>5, 5</td>
<td>5, 6</td>
</tr>
<tr>
<td>6, 0</td>
<td>6, 1</td>
<td>6, 2</td>
<td>6, 3</td>
<td>6, 4</td>
<td>6, 5</td>
<td>6, 6</td>
</tr>
<tr>
<td>7, 0</td>
<td>7, 1</td>
<td>7, 2</td>
<td>7, 3</td>
<td>7, 4</td>
<td>7, 5</td>
<td>7, 6</td>
</tr>
</tbody>
</table>

| 0, 0 | 0, 1 | 0, 2 | 0, 3 | 0, 4 | 0, 5 | 0, 6 | 0, 7 |
| 1, 0 | 1, 1 | 1, 2 | 1, 3 | 1, 4 | 1, 5 | 1, 6 | 1, 7 |
| 2, 0 | 2, 1 | 2, 2 | 2, 3 | 2, 4 | 2, 5 | 2, 6 | 2, 7 |
| 3, 0 | 3, 1 | 3, 2 | 3, 3 | 3, 4 | 3, 5 | 3, 6 | 3, 7 |
| 4, 0 | 4, 1 | 4, 2 | 4, 3 | 4, 4 | 4, 5 | 4, 6 | 4, 7 |
| 5, 0 | 5, 1 | 5, 2 | 5, 3 | 5, 4 | 5, 5 | 5, 6 | 5, 7 |
| 6, 0 | 6, 1 | 6, 2 | 6, 3 | 6, 4 | 6, 5 | 6, 6 | 6, 7 |
| 7, 0 | 7, 1 | 7, 2 | 7, 3 | 7, 4 | 7, 5 | 7, 6 | 7, 7 |
Parallel Matrix Multiplication

- **Partitioning into Submatrices**
  - Suppose the matrix is divided into $s^2$ submatrices. Each submatrix has $n/s \times n/s$ elements. Using the notation $A_{p,q}$ as the submatrix in submatrix row $p$ and submatrix column $q$:
    ```c
    for (p = 0; p < s; p++)
        for (q = 0; q < s; q++) {
            Cp,q = 0; /* clear elements of submatrix */
            for (r = 0; r < m; r++) /* submatrix multiplication and */
                Cp,q = Cp,q + Ap,r * Br,q; /* add to accumulating
                submatrix */
        }
    • The line
      
      $C_{p,q} = C_{p,q} + A_{p,r} * B_{r,q}$
      
      means multiply submatrix $A_{p,r}$ and $B_{r,q}$ using matrix multiplication
      and add to submatrix $C_{p,q}$ using matrix addition.
Algorithm
Cannon Algorithm

1. Initially processor $P_{i,j}$ has elements $a_{i,j}$ and $b_{i,j}$ ($0 \leq i < n$, $0 \leq k < n$).
Cannon Algorithm

2. Elements are moved from their initial position to an "aligned" position. The complete $i$th row of $A$ is shifted $i$ places left and the complete $j$th column of $B$ is shifted $j$ places upward.
Cannon Algorithm
Cannon Algorithm

3. Each processor $P_{i,j}$ multiply its elements.

4. The $i$th row of $A$ is shifted one place right, and the $j$th column of $B$ is shifted one place upward.
Cannon Algorithm
Cannon Algorithm

5. Each processor $P_{i,j}$ multiplies its elements brought to it and adds the results to the accumulating sum.

6. Step 4 and 5 are repeated until the final result is obtained ($n-1$ shifts with $n$ rows and $n$ columns of elements).
Cannon Algorithm

- Initially the matrix A:
  - Row0 is unchanged.
  - Row1 is shifted 1 place left.
  - Row2 is shifted 2 places left.
  - Row3 is shifted 3 places left.

- Initially the matrix B:
  - Column 0 is unchanged.
  - Column1 is shifted 1 place up.
  - Column2 is shifted 2 places up.
  - Column3 is shifted 3 places up.
Cannon Algorithm
Cannon Algorithm

\[
\begin{bmatrix}
a_{00} & a_{01} & a_{02} & a_{03} \\
a_{10} & a_{11} & a_{12} & a_{13} \\
a_{20} & a_{21} & a_{22} & a_{23} \\
a_{30} & a_{31} & a_{32} & a_{33}
\end{bmatrix}
\times
\begin{bmatrix}
b_{00} & b_{01} & b_{02} & b_{03} \\
b_{10} & b_{11} & b_{12} & b_{13} \\
b_{20} & b_{21} & b_{22} & b_{23} \\
b_{30} & b_{31} & b_{32} & b_{33}
\end{bmatrix}
= 
\begin{bmatrix}
c_{00} & c_{01} & c_{02} & c_{03} \\
c_{10} & c_{11} & c_{12} & c_{13} \\
c_{20} & c_{21} & c_{22} & c_{23} \\
c_{30} & c_{31} & c_{32} & c_{33}
\end{bmatrix}
\]
Cannon Algorithm (0)

\[
\begin{align*}
    a_{00} & \quad a_{01} & \quad a_{02} & \quad a_{03} \\
    b_{00} & \quad b_{11} & \quad b_{22} & \quad b_{33} \\
    a_{11} & \quad a_{12} & \quad a_{13} & \quad a_{10} \\
    b_{10} & \quad b_{21} & \quad b_{32} & \quad b_{03} \\
    a_{22} & \quad a_{23} & \quad a_{20} & \quad a_{21} \\
    b_{20} & \quad b_{31} & \quad b_{02} & \quad b_{13} \\
    a_{33} & \quad a_{30} & \quad a_{31} & \quad a_{32} \\
    b_{30} & \quad b_{01} & \quad b_{12} & \quad b_{23}
\end{align*}
\]

\[
\begin{align*}
    c_{00} &= a_{00} & b_{00} & c_{10} &= a_{11} & b_{10} \\
    c_{01} &= a_{01} & b_{11} & c_{11} &= a_{12} & b_{21} \\
    c_{02} &= a_{02} & b_{22} & c_{12} &= a_{13} & b_{32} \\
    c_{03} &= a_{03} & b_{33} & c_{13} &= a_{10} & b_{03} \\
    c_{20} &= a_{22} & b_{20} & c_{30} &= a_{33} & b_{30} \\
    c_{21} &= a_{23} & b_{31} & c_{31} &= a_{30} & b_{01} \\
    c_{22} &= a_{20} & b_{02} & c_{32} &= a_{31} & b_{12} \\
    c_{23} &= a_{21} & b_{13} & c_{33} &= a_{32} & b_{23}
\end{align*}
\]
## Cannon Algorithm (1)

<table>
<thead>
<tr>
<th>a_{01}</th>
<th>a_{02}</th>
<th>a_{03}</th>
<th>a_{00}</th>
</tr>
</thead>
<tbody>
<tr>
<td>b_{10}</td>
<td>b_{21}</td>
<td>b_{32}</td>
<td>b_{03}</td>
</tr>
<tr>
<td>a_{12}</td>
<td>a_{13}</td>
<td>a_{10}</td>
<td>a_{11}</td>
</tr>
<tr>
<td>b_{20}</td>
<td>b_{31}</td>
<td>b_{02}</td>
<td>b_{13}</td>
</tr>
<tr>
<td>a_{23}</td>
<td>a_{20}</td>
<td>a_{21}</td>
<td>a_{22}</td>
</tr>
<tr>
<td>b_{30}</td>
<td>b_{01}</td>
<td>b_{12}</td>
<td>b_{23}</td>
</tr>
<tr>
<td>a_{30}</td>
<td>a_{31}</td>
<td>a_{32}</td>
<td>a_{33}</td>
</tr>
<tr>
<td>b_{00}</td>
<td>b_{11}</td>
<td>b_{22}</td>
<td>b_{33}</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
    c_{00} + &= a_{01} & b_{10} & c_{10} + &= a_{12} & b_{20} \\
    c_{01} + &= a_{02} & b_{21} & c_{11} + &= a_{13} & b_{31} \\
    c_{02} + &= a_{03} & b_{32} & c_{12} + &= a_{10} & b_{02} \\
    c_{03} + &= a_{00} & b_{03} & c_{13} + &= a_{11} & b_{13} \\
    c_{20} + &= a_{23} & b_{30} & c_{30} + &= a_{30} & b_{00} \\
    c_{21} + &= a_{20} & b_{01} & c_{31} + &= a_{31} & b_{11} \\
    c_{22} + &= a_{21} & b_{12} & c_{32} + &= a_{32} & b_{22} \\
    c_{23} + &= a_{22} & b_{23} & c_{33} + &= a_{33} & b_{33}
\end{align*}
\]
Cannon Algorithm (2)

<table>
<thead>
<tr>
<th></th>
<th>$a_{02}$</th>
<th>$a_{03}$</th>
<th>$a_{00}$</th>
<th>$a_{01}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_{20}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{31}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{13}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{30}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{01}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{20}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{00}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{21}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{11}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{31}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{32}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{21}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{33}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{22}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{30}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{03}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$c_{00} + = a_{02}$</th>
<th>$b_{20} = a_{13}$</th>
<th>$b_{30} = a_{13}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{01}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_{01} + = a_{03}$</td>
<td>$b_{31} = a_{10}$</td>
<td>$b_{01} = a_{10}$</td>
<td></td>
</tr>
<tr>
<td>$a_{12}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_{02} + = a_{00}$</td>
<td>$b_{02} = a_{11}$</td>
<td>$b_{12} = a_{11}$</td>
<td></td>
</tr>
<tr>
<td>$a_{13}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_{03} + = a_{01}$</td>
<td>$b_{13} = a_{12}$</td>
<td>$b_{23} = a_{12}$</td>
<td></td>
</tr>
<tr>
<td>$a_{23}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_{20} + = a_{20}$</td>
<td>$b_{00} = a_{31}$</td>
<td>$b_{10} = a_{31}$</td>
<td></td>
</tr>
<tr>
<td>$a_{22}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_{21} + = a_{21}$</td>
<td>$b_{11} = a_{32}$</td>
<td>$b_{21} = a_{32}$</td>
<td></td>
</tr>
<tr>
<td>$a_{23}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_{22} + = a_{22}$</td>
<td>$b_{22} = a_{33}$</td>
<td>$b_{32} = a_{33}$</td>
<td></td>
</tr>
<tr>
<td>$a_{33}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_{23} + = a_{23}$</td>
<td>$b_{33} = a_{30}$</td>
<td>$b_{03} = a_{30}$</td>
<td></td>
</tr>
</tbody>
</table>

ULUSAL YÜKSEK BAŞARIMLI HESAPLAMA MERKEZİ / İSTANBUL TEKNIK ÜNİVERSİTESİ, AYAZAĞA YERLEŞKESİ
Cannon Algorithm (3)

\[
\begin{array}{cccc}
\begin{array}{ccc}
 a_{03} & a_{00} & a_{01} \\
b_{30} & b_{00} & b_{12} \\
a_{10} & a_{11} & a_{12} \\
b_{00} & b_{11} & b_{22} \\
a_{21} & a_{22} & a_{23} \\
b_{10} & b_{21} & b_{32} \\
a_{32} & a_{33} & a_{30} \\
b_{20} & b_{31} & b_{02} \\
\end{array} &
\begin{array}{ccc}
a_{02} & a_{13} & a_{20} \\
b_{23} & b_{33} & b_{03} \\
a_{31} & a_{30} & a_{02} \\
b_{13} & b_{02} & b_{21} \\
\end{array} &
\begin{array}{ccc}
c_{00} + = a_{03} & b_{30} & c_{10} + = a_{10} \\
\end{array} &
\begin{array}{ccc}
b_{00} \\
c_{01} + = a_{00} & b_{01} & c_{11} + = a_{11} \\
\end{array} &
\begin{array}{ccc}
b_{11} \\
c_{02} + = a_{01} & b_{12} & c_{12} + = a_{12} \\
\end{array} &
\begin{array}{ccc}
b_{22} \\
c_{03} + = a_{02} & b_{23} & c_{13} + = a_{13} \\
\end{array} &
\begin{array}{ccc}
b_{33} \\
\end{array} &
\begin{array}{ccc}
c_{20} + = a_{21} & b_{10} & c_{30} + = a_{32} \\
\end{array} &
\begin{array}{ccc}
b_{20} \\
c_{21} + = a_{22} & b_{21} & c_{31} + = a_{33} \\
\end{array} &
\begin{array}{ccc}
b_{31} \\
c_{22} + = a_{23} & b_{32} & c_{32} + = a_{30} \\
\end{array} &
\begin{array}{ccc}
b_{02} \\
c_{23} + = a_{20} & b_{03} & c_{33} + = a_{31} \\
\end{array} &
\begin{array}{ccc}
b_{13} \\
\end{array}
\end{array}
\]
1D-2D-3D Topologies

Interconnection Networks

- Figure 1.10 Ring.

- Figure 1.13 Three-dimensional hypercube.

- Figure 1.11 Two-dimensional array (mesh).
Creating Topology

- Grid Elements:
  - the dimension: 1, 2, 3 etc
  - the sizes of each dimension
  - the periodicity if the extreme are adjacent
- MPI Methods:
  - `MPI_Cart_create()` to create the grid
  - `MPI_Card_coords()` to get the coordinates
  - `MPI_Card_rank()` to find the rank
Computation Analysis

• Each submatrix multiplication requires $m^3$ multiplications and $m^3$ additions.

• Hence, with $s - 1$ shifts,

$$t_{\text{comp}} = 2s \ m^3 = 2 \ m^2 \ n$$

or a computational time complexity of $O(m^2 \ n)$
Communication Analysis

- Given $s^2$ submatrices, each of size $mxm$, the initial alignment requires a maximum of $s - 1$ shift (communication) operations. After that, there will be $s - 1$ shift operations.
- Each shift operation involves $mxm$ elements.

$$t_{\text{comm}} = 2(s - 1)(t_{\text{startup}} + m^2 t_{\text{data}})$$

or a communication time complexity of $O(sm^2)$ or $O(mn)$.
Fox Algorithm

- Both $n$ matrices $A$ and $B$ are partitioned among $p$ processors so that each processor initially stores $\left(\frac{n}{\sqrt{p}}\right) \times \left(\frac{n}{\sqrt{p}}\right)$.

- The algorithm uses one to all broadcasts of the blocks of matrix $A$ in processor rows, and single-step circular upwards shifts of the blocks of matrix $B$ along processor columns.

- Initially, each diagonal block $A_{ii}$ is selected for broadcast.
Fox Algorithm

• Steps (repeated $\sqrt{p}$ times)
  – Broadcast $A_{i,i}$ to all processors in the row
  – Multiply block of $A$ received with resident block of $B$
  – Send the block of $B$ up one step (with wraparound)
  – Select block $A_{i,(j+1)\mod \sqrt{p}}$ (where $A_{i,j}$ is the block broadcast in the previous step) and broadcast to all processors in row. Go to 2
Fox Algorithm (0)

- Initially broadcast the diagonal elements of A

\[
\begin{array}{cccc}
  a_{00} & b_{01} & b_{02} & b_{03} \\
  b_{00} & a_{11} & b_{12} & b_{13} \\
  b_{10} & b_{11} & a_{22} & b_{23} \\
  b_{20} & b_{21} & b_{22} & a_{33} \\
  b_{30} & b_{31} & b_{32} & b_{33} \\
\end{array}
\]

\[
\begin{align*}
  c_{00} &= a_{00} & b_{00} & c_{10} &= a_{11} & b_{10} \\
  c_{01} &= a_{00} & b_{01} & c_{11} &= a_{11} & b_{11} \\
  c_{02} &= a_{00} & b_{02} & c_{12} &= a_{11} & b_{12} \\
  c_{03} &= a_{00} & b_{03} & c_{13} &= a_{11} & b_{13} \\
  c_{20} &= a_{22} & b_{20} & c_{30} &= a_{33} & b_{30} \\
  c_{21} &= a_{22} & b_{21} & c_{31} &= a_{33} & b_{31} \\
  c_{22} &= a_{22} & b_{22} & c_{32} &= a_{33} & b_{32} \\
  c_{23} &= a_{22} & b_{23} & c_{33} &= a_{33} & b_{33} \\
\end{align*}
\]
**Fox Algorithm (1)**

- Broadcast the next element of A in rows, shift B in column and perform multiplication

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_{10}$</td>
<td>$a_{01}$</td>
<td>$b_{12}$</td>
<td>$b_{13}$</td>
</tr>
<tr>
<td>$b_{20}$</td>
<td>$b_{21}$</td>
<td>$a_{12}$</td>
<td>$b_{23}$</td>
</tr>
<tr>
<td>$b_{30}$</td>
<td>$b_{31}$</td>
<td>$b_{32}$</td>
<td>$a_{23}$</td>
</tr>
<tr>
<td>$a_{30}$</td>
<td>$b_{00}$</td>
<td>$b_{01}$</td>
<td>$b_{02}$</td>
</tr>
</tbody>
</table>

- $c_{00} = a_{10}$
- $b_{10}$
- $c_{20} = a_{23}$
- $b_{30}$
- $c_{01} = a_{01}$
- $b_{11}$
- $c_{21} = a_{23}$
- $b_{31}$
- $c_{02} = a_{01}$
- $b_{12}$
- $c_{22} = a_{23}$
- $b_{32}$
- $c_{03} = a_{01}$
- $b_{13}$
- $c_{23} = a_{23}$
- $b_{33}$
- $c_{10} = a_{12}$
- $b_{20}$
- $c_{30} = a_{30}$
- $b_{00}$
- $c_{11} = a_{12}$
- $b_{21}$
- $c_{31} = a_{30}$
- $b_{01}$
- $c_{12} = a_{12}$
- $b_{22}$
- $c_{32} = a_{30}$
- $b_{02}$
- $c_{13} = a_{12}$
- $b_{23}$
- $c_{33} = a_{30}$
- $b_{03}$
Fox Algorithm (2)

- Broadcast the next element of A in rows, shift B in column and perform multiplication

\[
\begin{array}{ccc}
 b_{20} & b_{21} & a_{02} \\
 b_{30} & b_{31} & a_{13} \\
 a_{20} & b_{00} & b_{02} \\
 b_{10} & b_{11} & b_{12} \\
\end{array}
\quad
\begin{array}{c}
 c_{00} + = a_{02} & b_{20} & c_{10} + = a_{13} & b_{30} \\
 c_{01} + = a_{02} & b_{21} & c_{11} + = a_{13} & b_{31} \\
 c_{02} + = a_{02} & b_{22} & c_{12} + = a_{13} & b_{32} \\
 c_{03} + = a_{02} & b_{23} & c_{13} + = a_{13} & b_{33} \\
\end{array}
\]
**Fox Algorithm (3)**

- Broadcast the next element of A in rows, shift B in column and perform multiplication

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_{30}$</td>
<td>$b_{31}$</td>
<td>$b_{32}$</td>
<td>$a_{03}$</td>
</tr>
<tr>
<td>$a_{10}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{00}$</td>
<td>$b_{01}$</td>
<td>$b_{02}$</td>
<td>$b_{03}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{10}$</td>
<td>$b_{11}$</td>
<td>$b_{12}$</td>
<td>$b_{13}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b_{20}$</td>
<td>$b_{21}$</td>
<td>$b_{22}$</td>
<td>$b_{23}$</td>
</tr>
</tbody>
</table>

$c_{00} + = a_{03} \quad b_{30} \quad c_{10} + = a_{10} \quad b_{00}$
$c_{01} + = a_{03} \quad b_{31} \quad c_{11} + = a_{10} \quad b_{01}$
$c_{02} + = a_{03} \quad b_{32} \quad c_{12} + = a_{10} \quad b_{02}$
$c_{03} + = a_{03} \quad b_{33} \quad c_{13} + = a_{10} \quad b_{03}$
$c_{20} + = a_{21} \quad b_{10} \quad c_{30} + = a_{32} \quad b_{20}$
$c_{21} + = a_{21} \quad b_{11} \quad c_{31} + = a_{32} \quad b_{21}$
$c_{22} + = a_{21} \quad b_{12} \quad c_{32} + = a_{32} \quad b_{22}$
$c_{23} + = a_{21} \quad b_{13} \quad c_{33} + = a_{32} \quad b_{23}$
Fox Algorithm (4)

- Shifting is over. Stop the ITERATION

<table>
<thead>
<tr>
<th>$a_{00}$</th>
<th>$b_{00}$</th>
<th>$b_{01}$</th>
<th>$b_{02}$</th>
<th>$b_{03}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_{10}$</td>
<td>$a_{11}$</td>
<td>$b_{11}$</td>
<td>$b_{12}$</td>
<td>$b_{13}$</td>
</tr>
<tr>
<td>$b_{20}$</td>
<td>$b_{21}$</td>
<td>$a_{22}$</td>
<td>$b_{22}$</td>
<td>$b_{23}$</td>
</tr>
<tr>
<td>$b_{30}$</td>
<td>$b_{31}$</td>
<td>$b_{32}$</td>
<td>$a_{33}$</td>
<td>$b_{33}$</td>
</tr>
</tbody>
</table>

- Fox algorithm is a memory efficient method.

- Communication overhead is more than cannon algorithm