1 MATRIX MULTIPLICATION

Given $A = (a_{ij})$ and $B = (b_{ij})$ are square matrices of order n, then $C = (c_{ij}) = AB$ is also a square matrix of order n, and c_{ij} is obtained by taking the dot product of the ith row of A with the jth column of B. In other words, $c_{ij} = a_{i0}b_{0j} + a_{i1}b_{1j} + ... + a_{i,n-1}b_{n-1,j}$.

Matrix Multiplication Algorithm:

1. for each row of C
    2. for each column of C
        3. C[row][column]=0.0;
        4. for each element of this row of A
            5. Add A[row][element]*B[element][column] to C[row][column]

We can see that a straightforward parallel implementation of matrix multiplication is costly. If we suppose that the number of processes is the same as the order of the matrices; i.e, $p = n$, and suppose that we have distributed the matrices by rows. So process 0 is assigned row 0 of A, B, and C; process 1 is assigned row 1 of A, B, and C; etc. Then in order to form the dot product of the ith row of A with jth column of B, we will need to form the dot product of the jth column with every row of A. So we will have to carry out an allgather (gathers data from all tasks and distribute the combined data to all tasks) rather than a gather (gathers together values from a group of processes), and we will have to do this for every column of B. This involves a lot of expensive communication. Similarly, an algorithm that distributes the matrices by columns will involve large amount of communication. In order to reduce communication, most parallel matrix multiplication algorithms use a checkerboard distribution of the matrices. The matrices are then seen as a grid. Instead of assigning entire rows or columns to each process, we assign small submatrices. For example, if we have a 4×4 matrix, we can assign the elements of the matrix in the following way to 4 processes:

Process 0		Process 1	
a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
Process 2		Process 3	
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

1.1 Fox's Algorithm

Assume that the matrices have order n, and the number of processes, p, equals n^2. Then a checkerboard mapping assigns a_{ij}, b_{ij}, and c_{ij} to process (i, j).
Fox's algorithm proceeds in n stages: one stage for each term $a_{ik}b_{kj}$ in the dot product

$c_{ij} = a_{i0}b_{0j} + a_{i1}b_{1j} + ... + a_{i,n-1}b_{n-1,j}$.

1. Initial stage: each process multiplies the diagonal entry of A in its process row by its element of B:
   Stage 0 on process (i, j): $c_{ij}{=}a_{ii}b_{ij}$.

2. Next stage: each process multiplies the element immediately to the right of the diagonal of A (in its process row) by the process of B directly beneath its own element of B:
   Stage 1 on process (i, j): $c_{ij}{=}c_{ij}{+}a_{i,i+1}b_{i+1,j}$.

3. In general, during the kth stage, each process multiplies the element k columns to the right of the diagonal of A by the element k rows below its own element of B:
   Stage k on process (i, j): $c_{ij}{=}c_{ij}{+}a_{i,i+k}b_{i+k,j}$.

4. We may encounter out of range subscript when adding k to a row or column. Instead, w define $\bar{k}{=}(i{+}k)$ mod n. Hence, for stage k on process (i, j): $c_{ij}{=}c_{ij}{+}a_{i,\bar{k}}b_{\bar{k},j}$.
   c_{ij} will be computed as follows:
   $c_{ij} = a_{ii}b_{ij} + a_{i,i+i}b_{i+1}j + ... + a_{i,n-1}b_{n-1,j} + a_{i0}b_{0j} + ... + a_{i,i-1}b_{i-1,j}$.

We will store sub-matrices rather than matrix elements on each process, provided that the sub-matrices can be multiplied as needed and the number of process rows or process columns, $\sqrt{p}$, evenly divides n. Given this assumption, each process is assigned a square $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ sub-matrix of each of the three matrices. Define $q = \sqrt{p}$ and $\bar{n} = \frac{n}{\sqrt{p}}$.Denote the sub-matrices A_{ij}, B_{ij}, and C_{ij} by $A[i, j]$, $B[i, j]$, and $C[i, j]$, the **pseudo-code of Fox's algorithm is the following:**

1. my process row = i, my process column = j

2. $q = \sqrt{p}$

3. dest = ($(i - 1)$ mod q, j)

4. source = ($(i + 1)$ mod q, j)

5. for (stage = 0; stage < q; stage++)

   6. $\bar{k} = (i + stage)\text{mod}q$

   7. Broadcast $A[i, \bar{k}]$ across process row i

   8. $C[i, j] = C[i, j] + A[i, \bar{k}] * B[\bar{k}, j]$

   9. Send $B[\bar{k}, j]$ to dest; Receive

   10. $B[(\bar{k} + 1)$ mod q, $j]$ from source

The algorithm can also be written in the following way:

1. for (stage = 0; stage < q; stage++)

2. Choose a sub-matrix of A of each row of processes

3. In each row of processes broadcast the sub-matrix chosen in that row to the other processes in that row

4. On each process, multiply the newly received sub-matrix of A by the sub-matrix of B currently residing on the process

5. On each process, send the sub-matrix of B to the process directly above. (On processes in the first row, send the sub-matrix to the last row)

1.2 Example Applied to 2×2 Matrices

Assume we have n^2 processes, one for each element in A, B, and C.
Stage 0:

1. $A_{i,i}$ on process $P_{i,j}$, so broadcast the diagonal elements of A across the rows ($A_{ii} \rightarrow Pij$). This will place $A_{0,0}$ on each $P_{0,j}$ and $A_{1,1}$ on each $P_{1,j}$.
   The A elements on the P matrix will be:

A_{00}	A_{00}
A_{11}	A_{11}

2. $B_{i,j}$ on process $P_{i,j}$, so broadcast B across the rows ($B_{ij} \rightarrow P_{ij}$). The A and B values on the P matrix will be:

A_{00} B_{00}	A_{00} B_{01}
A_{11} B_{10}	A_{11} B_{11}

3. Compute $C_{ij} = AB$ for each processor:

A_{00} B_{00} $C_{00} = A_{00}B_{00}$	A_{00} B_{01} $C_{01} = A_{00}B_{01}$
A_{11} B_{10} $C_{10} = A_{11}B_{10}$	A_{11} B_{11} $C_{11} = A_{11}B_{11}$

   Now we are ready for the next stage. In this stage, we broadcast the next column (mod n) of A across the processes and shift up (mod n) the B value.

Stage 1:

1. The next column is $A_{0,1}$ for the 1^{st} row and $A_{1,0}$ for the 2^{nd} row. Broadcast the next A across the rows:

A_{01} B_{00} $C_{00} = A_{00}B_{00}$	A_{01} B_{01} $C_{01} = A_{00}B_{01}$
A_{10} B_{10} $C_{10} = A_{11}B_{10}$	A_{10} B_{11} $C_{11} = A_{11}B_{11}$

2. Shift the B values up. $B_{1,0}$ moves up from process $P_{1,0}$ to process $P_{0,0}$ and $B_{0,0}$ moves up (mod n) from $P_{0,0}$ to $P_{1,0}$. Similarly for $B_{1,1}$ and $B_{0,1}$.

A_{01} B_{10} $C_{00} = A_{00}B_{00}$	A_{01} B_{11} $C_{01} = A_{00}B_{01}$
A_{10} B_{00} $C_{10} = A_{11}B_{10}$	A_{10} B_{01} $C_{11} = A_{11}B_{11}$

3. Compute $C_{ij} = AB$ for each process:

A_{01} B_{10} $C_{00} = C_{00} + A_{01}B_{01}$	A_{01} B_{11} $C_{01} = C_{01} + A_{01}B_{11}$
A_{10} B_{00} $C_{10} = C_{10} + A_{10}B_{00}$	A_{10} B_{01} $C_{11} = C_{11} + A_{10}B_{01}$

1.3 Cannon's Algorithm

In the following algorithm, all the subscripts are modulo n.

1. Initially each $p_{i,j}$ has $a_{i,j}$ and $b_{i,j}$
2. Align elements $a_{i,j}$ and $b_{i,j}$ by reordering so that $a_{i,j+1}$ and $b_{i+j,j}$ are on $p_{i,j}$
3. Each $p_{i,j}$ computes $c_{i,j} = a_{i,j+1} \times b_{i+j,j}$ ($a_{i,j+i}$ and $b_{i+j,j}$ are local on $p_{i,j}$)
4. For $k = 1$ to $n - 1$:
   5. Rotate A left by one column
   6. Rotate B up by one row
   7. Each $p_{i,j}$ computes $c_{i,j} = c_{i,j} + a_{i,j+i+k} \times b_{i+j+k,j}$ ($a_{i,j+i+k}$ and $b_{i+j+k,j}$ are local on $p_{i,j}$ after k iterations).