INTRODUCTION TO SAMPLE APPLICATION FOR GRID

SHWETA PHABBA

System Software Development Group,
CDAC, Bangalore.

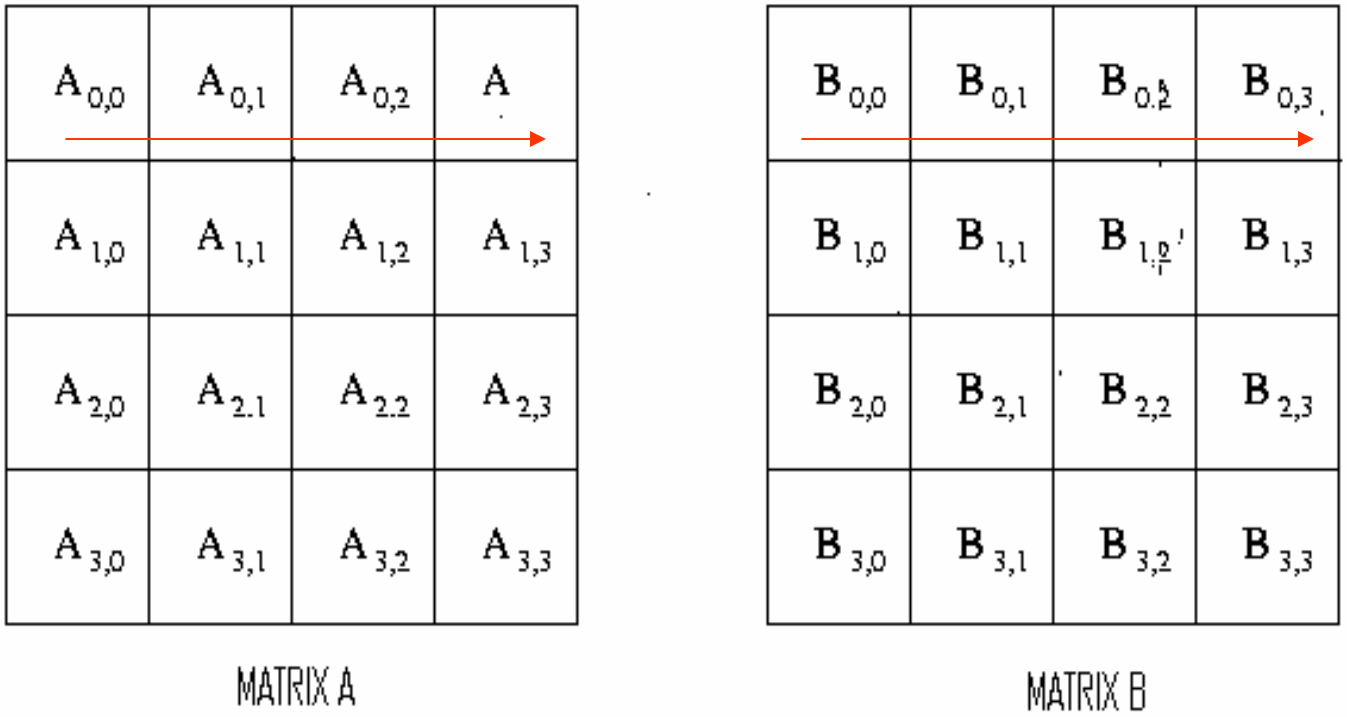
Presentation Outline

- Matrix Computation.
- Matrix Multiplication By Sequential Method.
- Matrix Multiplication By Parallel Method.
- Different Algorithms For Matrix Multiplication.
- Complexity Of Matrix Multiplication By Using Sequential Method.
- Scientific Applications Where Matrix Computation Is Used.

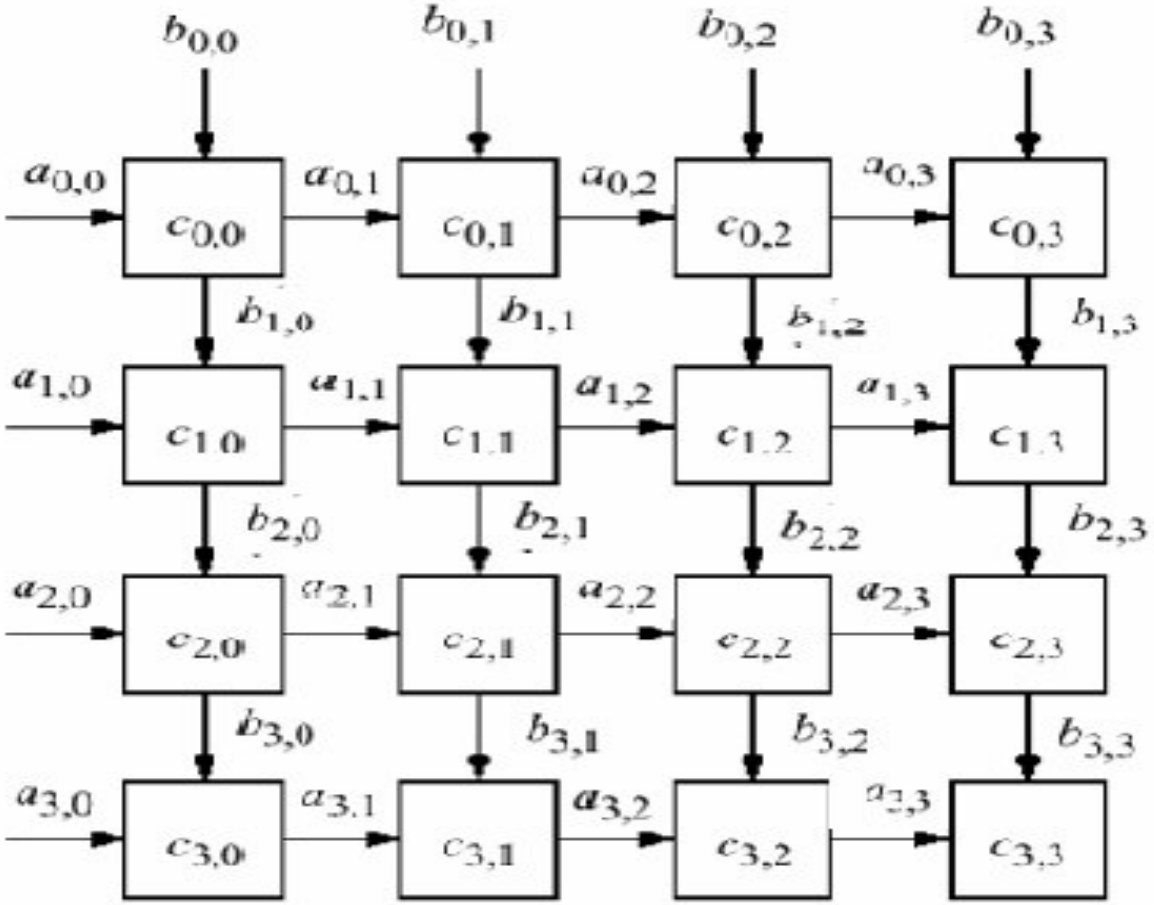
Matrix Computation

- Why Matrix Computation?
- Matrix – Matrix Addition.
- Matrix – Matrix Multiplication.

General Matrix Matrix Addition



Resultant Matrix Of A+B



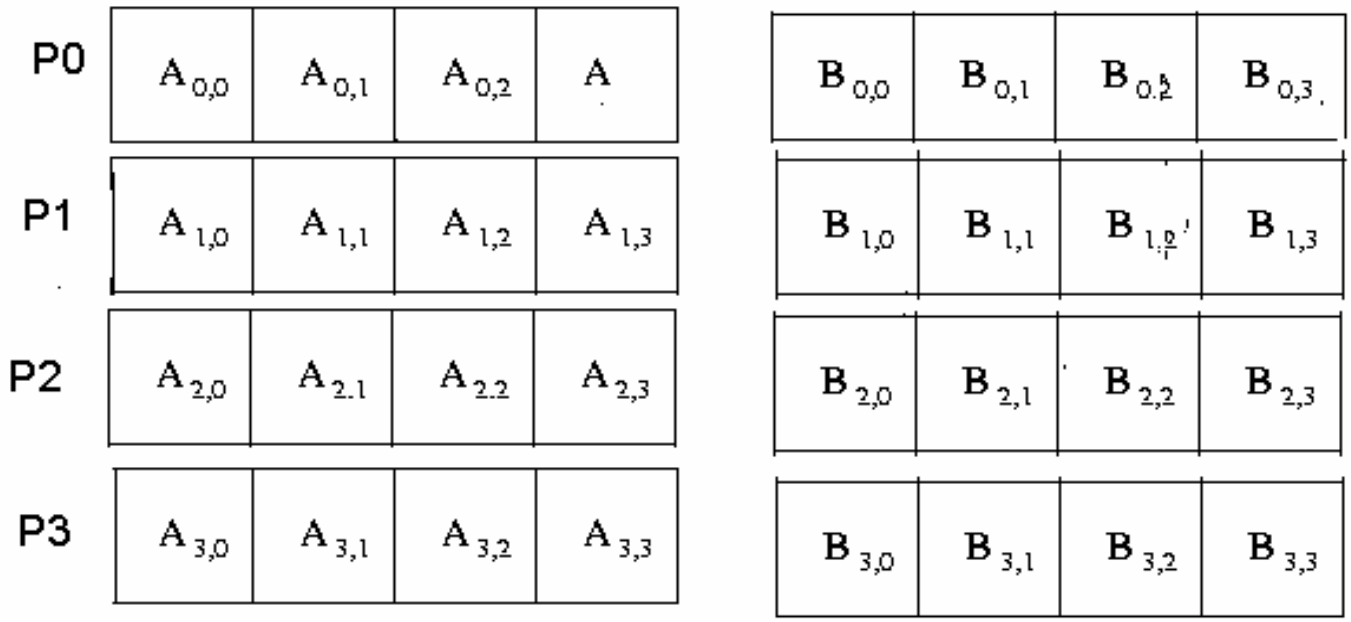
Code For Matrix-Matrix Addition

```
for (i=0 ; i<NoofRows_A ;i++)
for (j=0 ; j<NoofCols_A ; j++)
{
   C[i][j] = A[i][j]  + B[i][j];
}
```

If The Time Required For One Addition is 2 usec

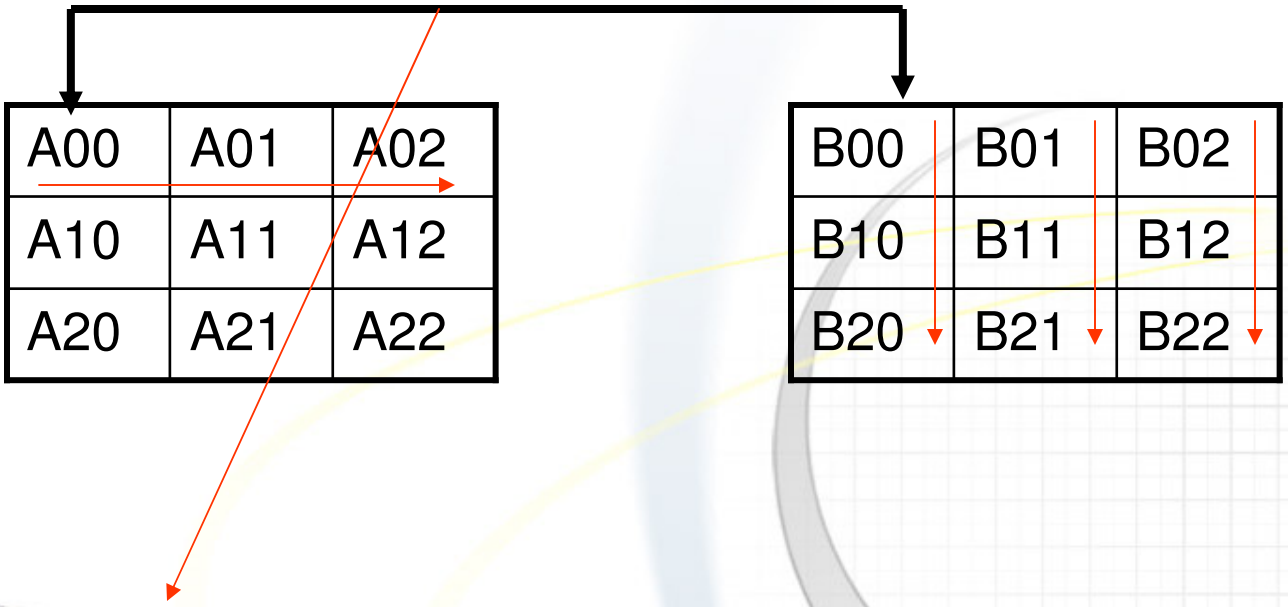
Time Required For 4 * 4 Matrix Addition = 16 * 2 usec

Approach For Parallizing Matrix-Matrix Addition



Matrix A And Matrix B are Divided among 4 Processor P0 ,P1, P2, P3.

Matrix-Matrix Multiplication



C00 = A00 * B00 + A01* B10 + A02*B20

C01 = A00 * B01 + A01 * B11 + A02 * B21

C02 = A00 * B02 + A01 * B12 + A02* B22

Sequential Method For Matrix-Matrix Multiplication

The product *C* of the two matrices *A* and *B* is defined by , where *aij, bij,* and *cij* is the element in *i*th row and *j*th column of the matrix *A, B,* and *C* respectively. In order for the matrix multiplication to be defined, the dimensions of the matrices multiplyied as *AB=C* must satisfy (n *m)(m * p)=(n * p).

If the matrices *A*, *B*, and *C* are all $n \times n$ matrices. The sequential algorithm of this matrix multiplication is as follows:

```
for  i = 0  to n-1
     for  j = 0 to n-1
     cij =0
          for  k = 0 to n-1
          cij = cij + aik × bkj
```

Algorithm For Matrix Multiplication

Sequential Code For Matrix-Matrix Multiplication

```
for (i=0; i<NoofCols_B; i++)
 {
    for (j=0; j<NoofRows_A; j++)
    {
       Matrix_C[i][j] = 0.0;
       for (k=0; k<NoofCols_A; k++)
       Matrix_C[i][j]  += Matrix_A[i][k] * Matrix_B[k][j];
    }
 }
```

Strassen's Matrix Multiplication Algorithm

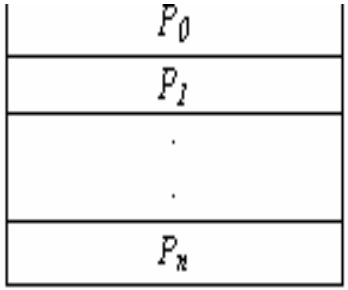
- The Strassen's algorithm is a sequential algorithm reduced the complexity of matrix multiplication algorithm to ***O***(*n^2.81*) .
- $n \times n$ matrix is divided into 4 $n/2 \times n/2$ sub matrices and multiplication is done recursively by multiplying $n/2 \times n/2$ matrices.
- Since this algorithm cuts matrix by 2 and multiplies recursively, the matrix must be the square matrix and its size must be the power of 2 .
- Limitation is overcome at Winograd's variant of Strassen's algorithm. In his algorithm the even size of matrix can be multiplied. If the matrices *A* and *B* are of size *2^dm* $\times$ *2^dm*, Strassen's algorithm can be used recursively *d* times. After *d* recursion, standard algorithm is used to multiply the $m \times m$ matrices.

Parallel Method For Matrix-Matrix Multiplication

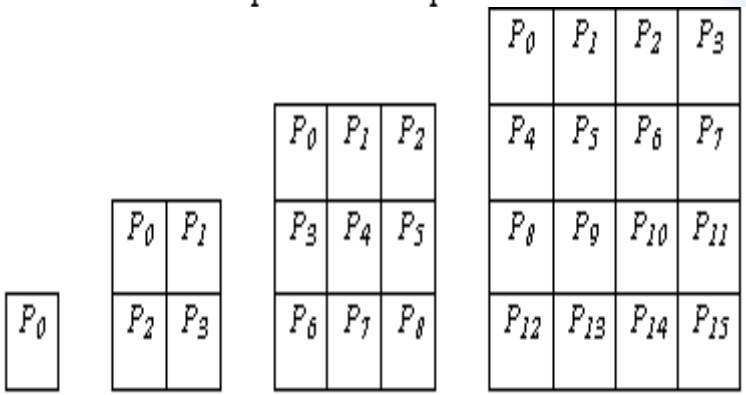
- Why to use Parallel Method for Matrix Multiplication ?
- General Method used for “Parallel Matrix Multiplication”.
- Matrix Decomposition Methods for Parallel Algorithm

One-Dimensional Decomposition.
Two-Dimensional General Decomposition.
Two-Dimensional Square Decomposition.
Two-Dimensional Scattered Decomposition.

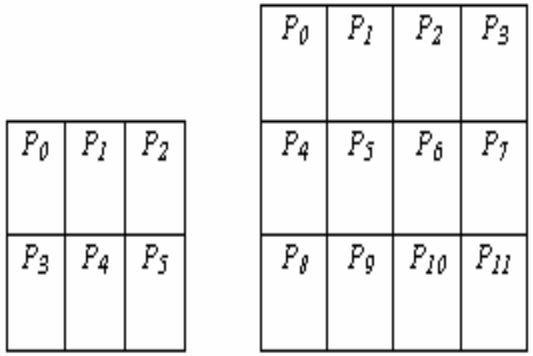
Decomposition Methods For Matrix



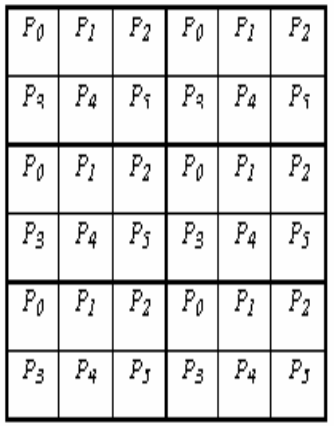
(a) One-dimensional processor template



(b) Square processor template with 1, 4, 9, and 16 processors.



(c) Two-dimensional general processor templates with 6 and 12 processors.



(c) two-dimensional scattered decomposition of 6× 6 matrix scattered into 2 × 3 processor template.

Contd.....

- Factors affecting the performance of Parallel Matrix Multiplication Methods.

1. Number of Processors involved .
2. Data Size Of the program.
3. Inter process Communication.
4. Memory Used.

For ex. If Parallel approach is applied to Strassen's algorithm and the task is divided among n ^ 3 processors for n * n Matrix then time complexity will be O(1)

Different Algorithms For Matrix Multiplication

- Most parallel matrix multiplication algorithms use matrix decomposition based on the number of processors available. These include the

  1. Self Scheduling Algorithm.
  2. Systolic Algorithm.
  3. Cannon's Algorithm
  4. Fox Algorithm

Self Scheduling Algorithm

- one of the most common parallel algorithm prototype is the *Self-Scheduling* or *Master-Slave* algorithm .General Structure for the Master slave approach is as :

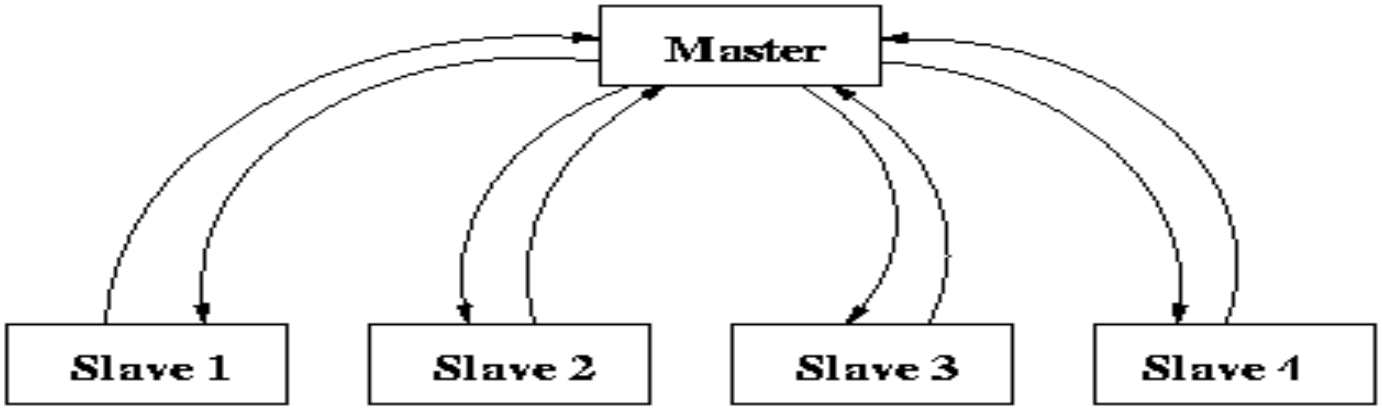
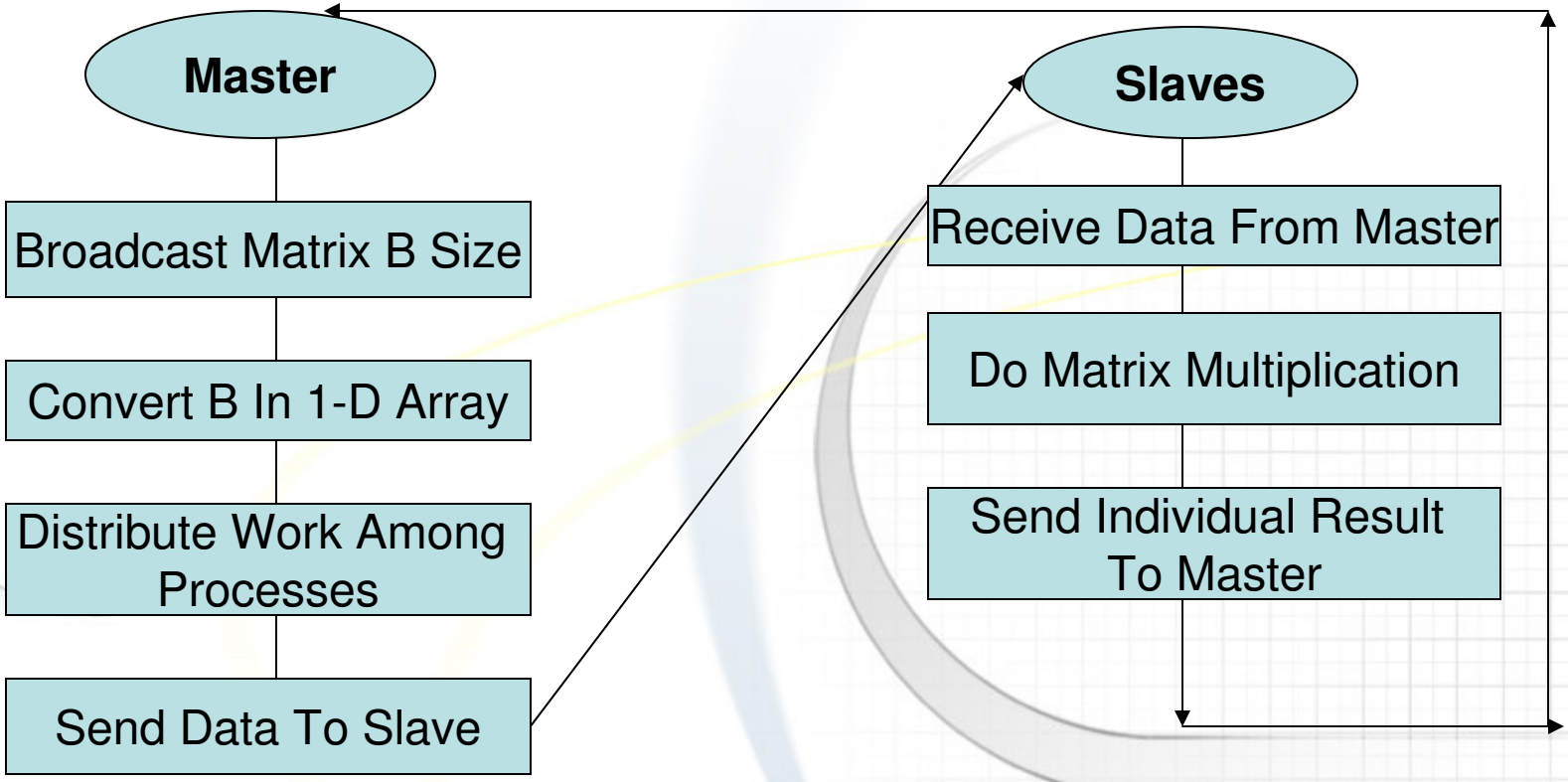


Fig :Communication pattern among master and slave in self scheduling paradigm

Flow Graph For Algorithm



Contd……

Description Of The Algorithm

1. The Idea is that the master process, distributes the work load to slave processes. When the slave finishes its workload, it informs the master, about the completion of the task assigned and then master assigns a new workload to the slave. This is very simple paradigm where the coordination is done by master. Here, the slave processes do not have to communicate with one another .
2. The master process stores the final output square matrix.
3. The master process receives whole *row* of the product matrix from *any slave,* and sends the next task to that *slave* processes

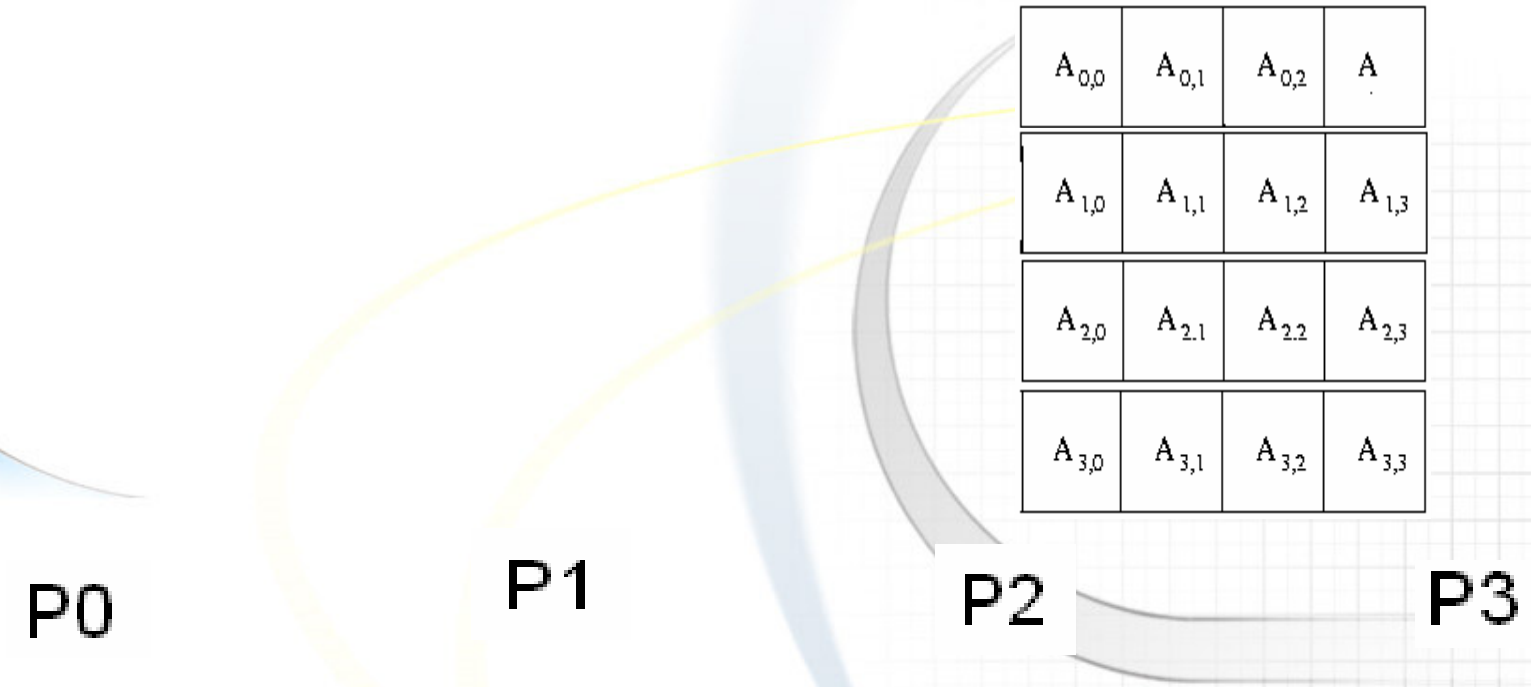
Contd……

- Completion of one task by a *slave* is considered to be a request for the next task. Once all the tasks have been handed out, termination messages are sent .
- Each *slave* process, after receiving second input matrix enters a loop which is terminated after receiving the termination message from *master*.

Advantages Of Using The Self Scheduling Algorithm:

1. It is one of the Simplest Matrix Multiplication Algorithms.
2. It uses simple One Dimensional Decomposition for the Matrix.
3. It considers the MPMD feature of Programming Paradigms.
4. Speed Up Achieved is good.

Matrix Decomposition Method Used In Self Scheduling Algorithm



Important Steps For Self Scheduling Algorithm(Master Process)

```
MPI_Bcast (&NoofColsB, 1, MPI_INT, 0, MPI_COMM_WORLD);

/*Convert Matrix B into 1-D Array */

BufferB = (float *) malloc(NoofRowsB*NoofColsB*sizeof(float));
index  = 0;
for(irow = 0; irow<NoofRowsB; irow++)
for(icol=0; icol<NoofColsB; icol++)
BufferB[index++] = Matrix_B[irow][icol];
MPI_Bcast(BufferB,NoofRowsB*NoofColsB,MPI_FLOAT,0,MPI_COMM
                _WORLD);
/*Allocate Memory For a Row Of A Matrix A */
RowA = (float *)malloc(NoofColsA*sizeof(float));
```

```
/* First Send At least one row to each processors */
  RowtoSend = 0;
  for(iproc = 1 ; iproc < Numprocs ; iproc++)
  {
  for(icol=0; icol<NoofColsA; icol++)
  RowA[icol] = Matrix_A[RowtoSend][icol];
  MPI_Send(RowA, NoofColsA, MPI_FLOAT, iproc, RowtoSend+1,
   MPI_COMM_WORLD);
  RowtoSend++;
  }
```

```
ResultRow = (float *) malloc(NoofColsB*sizeof(float));
for(irow = 0 ; irow < NoofRowsA ; irow++)
MPI_Recv(ResultRow, NoofColsB, MPI_FLOAT,
                MPI_ANY_SOURCE, MPI_ANY_TAG,
                MPI_COMM_WORLD, &status);
memcpy(ResultMatrix[Source_tag], ResultRow,
            NoofColsB*sizeof(float));

if( RowtoSend < NoofRowsA )
memcpy(RowA, Matrix_A[RowtoSend], NoofColsA*sizeof(float));
MPI_Send(RowA, NoofColsA, MPI_FLOAT, Destination,
                Destination_tag, MPI_COMM_WORLD);
```

Important Steps For Self Scheduling Algorithm (Slave Processes)

```
MPI_Recv (RowA, NoofColsA, MPI_FLOAT, Root,
                MPI_ANY_TAG,MPI_COMM_WORLD, &status);
for(icol = 0; icol< NoofColsB; icol++)
{
 ResultRow[icol] = 0;
 for(irow=0; irow< NoofRowsB; irow++)
 {
   index = irow*NoofColsB + icol;
   ResultRow[icol] += BufferB[index] * RowA[irow];
  }
}
MPI_Send(ResultRow, NoofColsB, MPI_FLOAT, 0, Destination_tag,
                MPI_COMM_WORLD);
```

Complexities Of Matrix-Matrix Multiplication By using Sequential Method

- This "Sequential Algorithm" for matrix – matrix multiplication requires *n3* additions and multiplications, therefore its time complexity is ***O***(*n3*).
- Multiplication of large matrices requires a lot of computation time as its complexity is ***O***(*n3*), where *n* is the dimension of the matrix
- Most current applications require higher computational throughput hence some advanced "Sequential Algorithms" are tried but having many limitations.

  Ex. Strassen's algorithm

Scientific Applications Where Matrix Computation Is Used

1. Image Processing.
2. Computer Animations.
3. Quantum Dot Cellular Automata
4. Bioinformatics Application.
5. Inte. Benchmarks for calculating the system Performance.
   Ex. BLAS.

Thank You