Mapping Techniques for Load Balancing

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Parallelization

- Decomposition
- Mapped to processes
- Overhead reduction

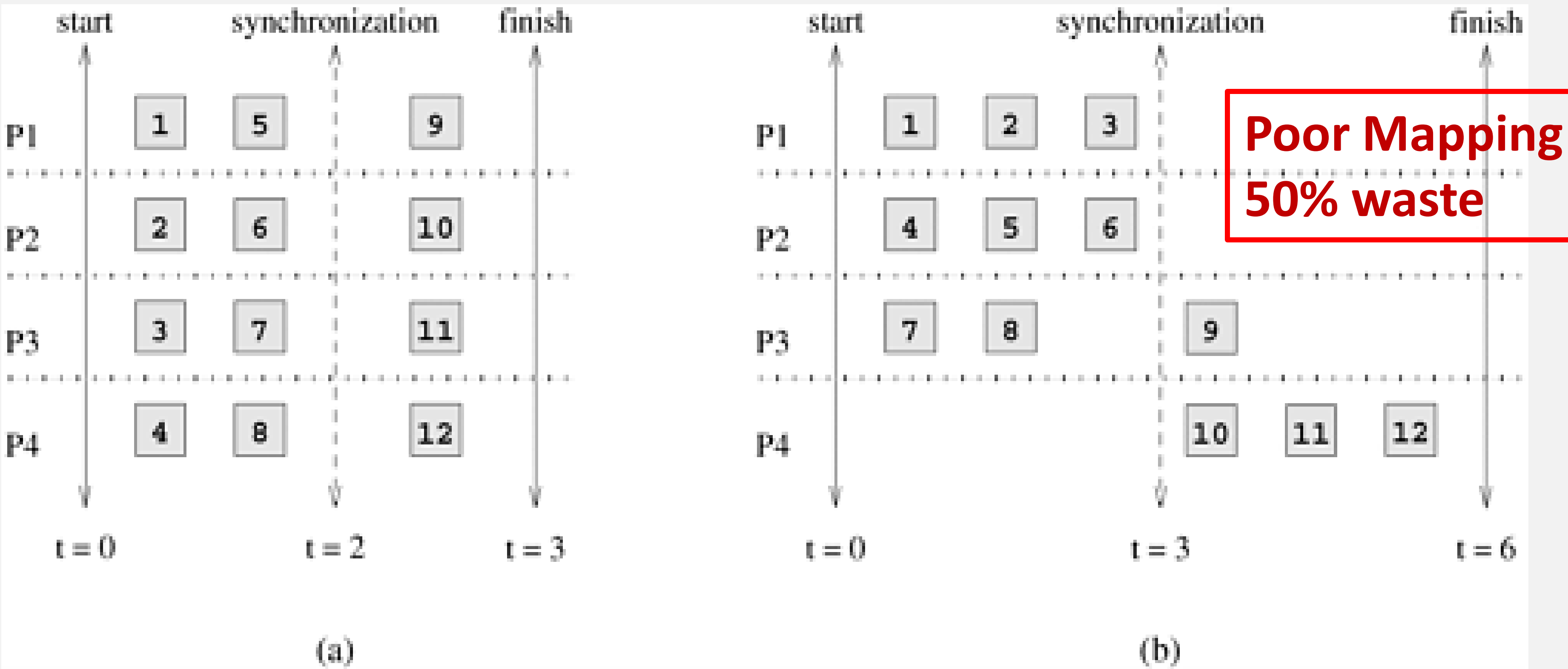
Good Mapping

- Finding a good mapping is a nontrivial problem
- Twin objectives
  - reducing the amount of time processes spend in interacting with each other,
  - reducing the total amount of time some processes are idle while the others are engaged in performing some tasks.
- They conflict each other
  - Less interaction---->unbalanced workload
  - Good balance--→ high interaction

Good Mapping

- A good mapping must ensure that the computations and interactions among processes at each stage of the execution of the parallel algorithm are well balanced

Two mappings of a hypothetical decomposition with a synchronization.



Task 9,10,11,12 can start after completion of first eight tasks

Different completion times for the two mappings

Two categories

- Static
- Dynamic

Static

- Used in conjunction with a decomposition based on data partitioning
- Used for mapping certain problems that are expressed naturally by a static task-dependency graph

Mappings Based on Data Partitioning

- Mappings based on partitioning two of the most common ways of representing data in algorithms, namely, arrays and graphs

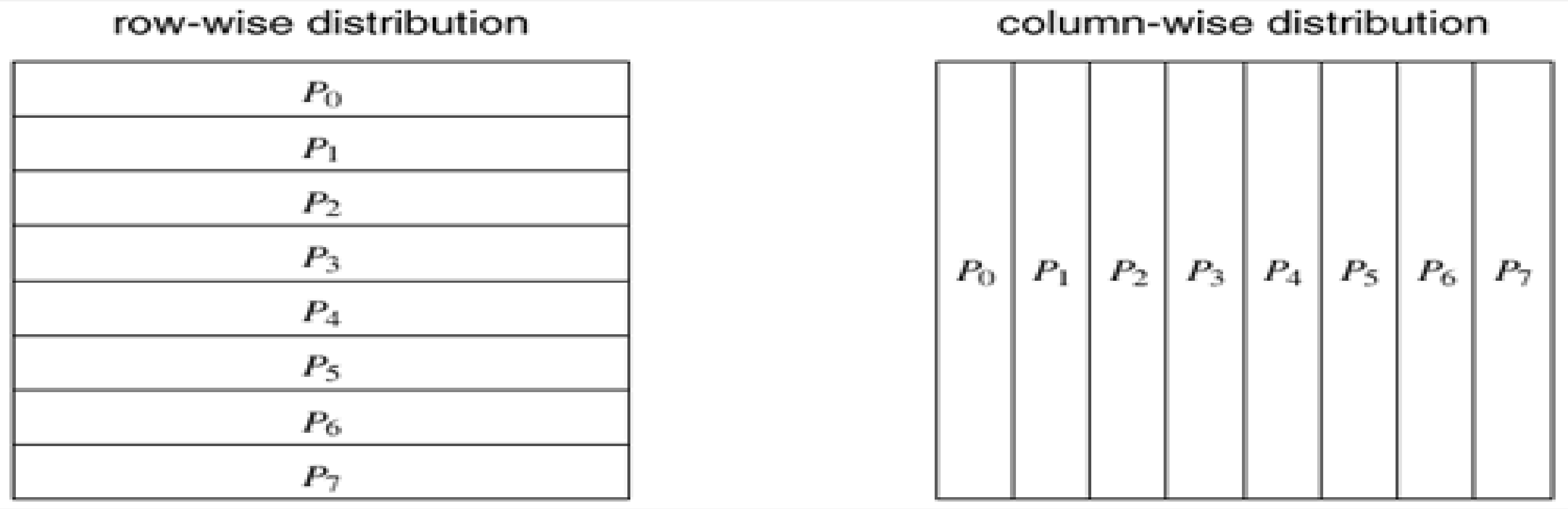
Array Distribution Schemes

- The tasks are closely associated with portions of data by the owner-computes rule
- Therefore, mapping the relevant data onto the processes is equivalent to mapping tasks onto processes

Block Distributions

- A *d*-dimensional array is distributed among the processes
  - such that each process receives a contiguous block of array entries along a specified subset of array dimensions.
- Block distributions of arrays are particularly suitable when there is a locality of interaction, i.e.,
  - computation of an element of an array requires other nearby elements in the array.

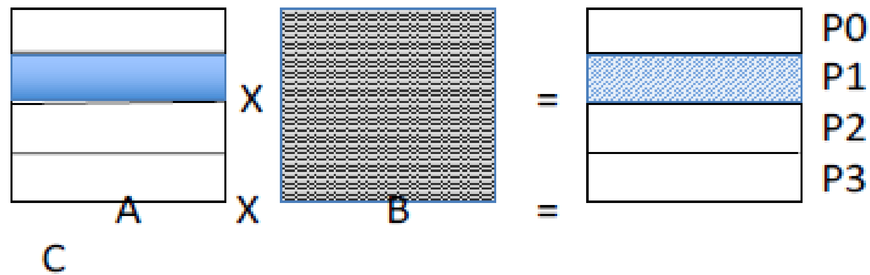
1D partitioning among 8 processes



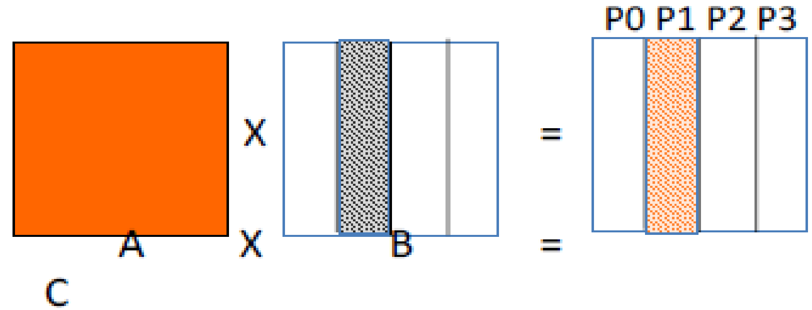
For example, consider an $n \times n$ two-dimensional array A with n rows and n columns. Now select one of these dimensions, e.g., the first dimension, and partition the array into p parts such that the kth part contains rows $kn/p \ldots (k + 1)n/p - 1$, where $0 \leq k < p$. That is, each partition contains a block of n/p consecutive rows of A. Similarly, if we partition A along the second dimension, then each partition contains a block of n/p consecutive columns.

Block Distribution and Data Sharing for Dense Matrix Multiplication

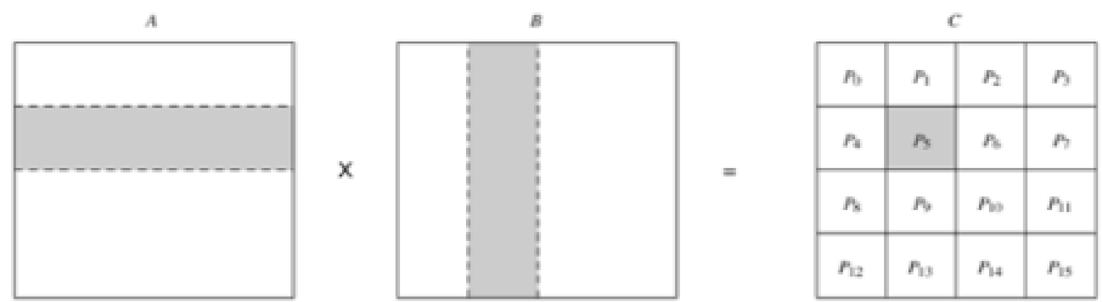
- Row-based 1-D

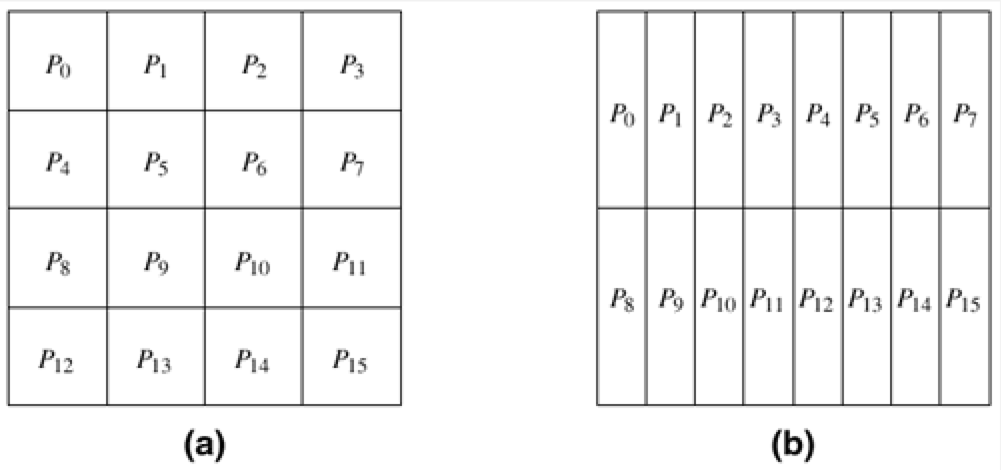


- Column-based 1-D

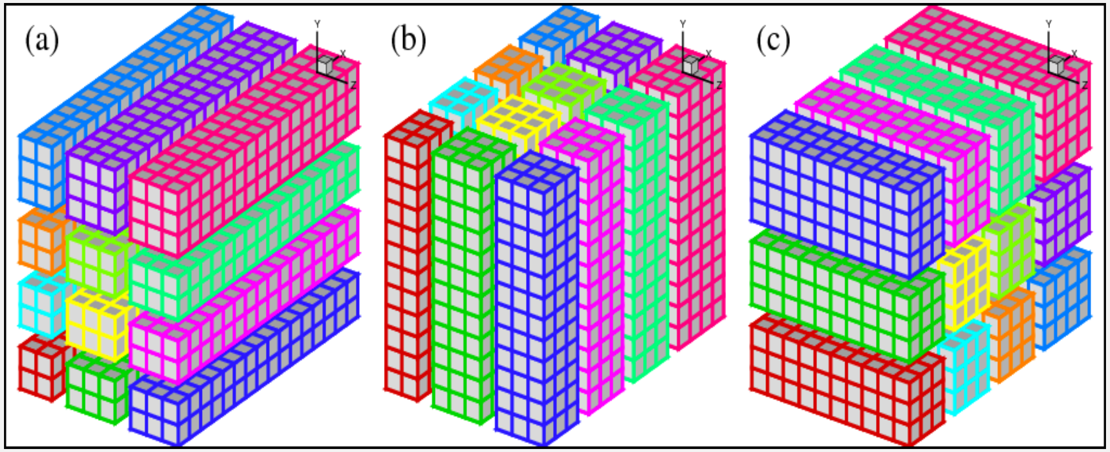


- Row/Col-based 2-D





Two-dimensional distributions of an array, (a) on a 4 x 4 process grid, and (b) on a 2 x 8 process grid.

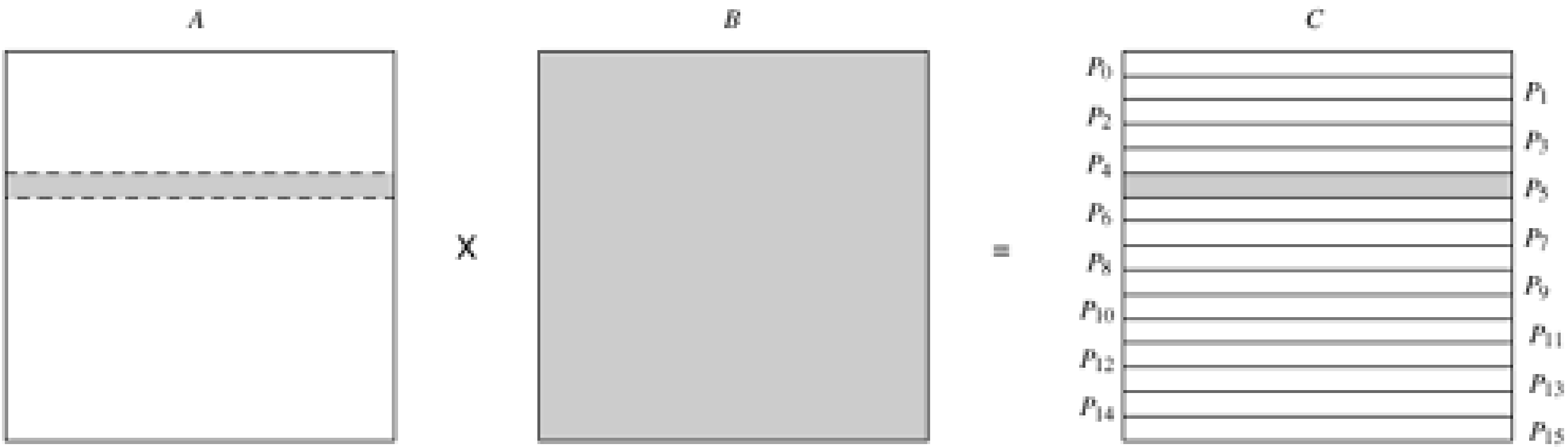


Instead of selecting a single dimension, we can select multiple dimensions to partition. For instance, in the case of array *A* we can select both dimensions and partition the matrix into blocks such that each block corresponds to a n/p_1 x n/p_2 section of the matrix, with $p = p_1 \times p_2$ being the number of processes

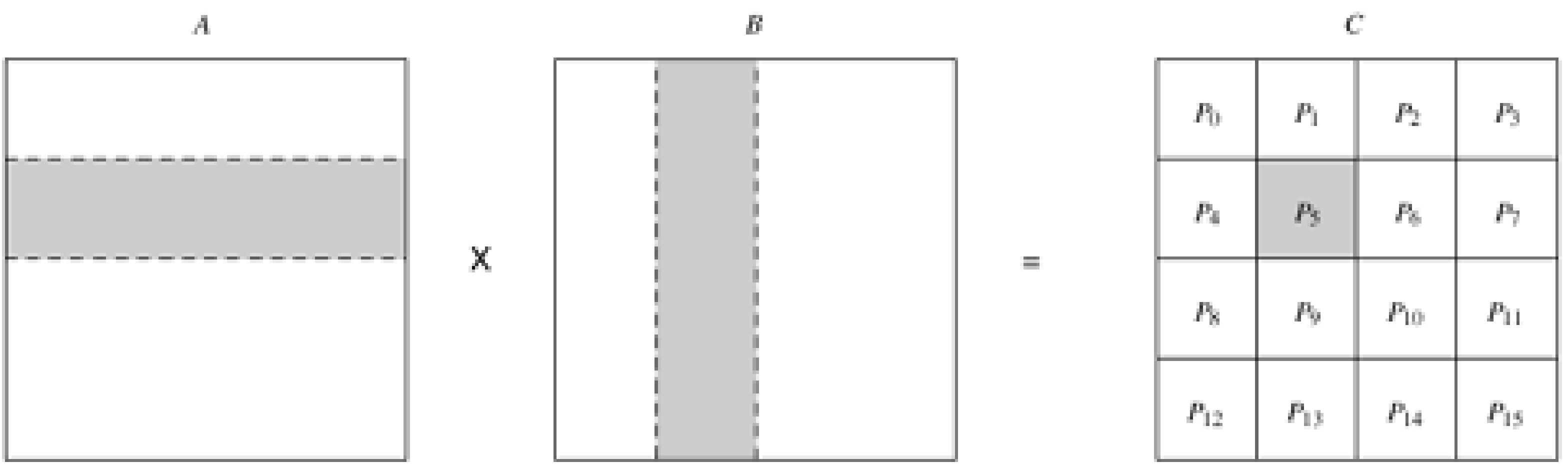
- For example, consider the n x n matrix multiplication $C = A$ x B
- One way of decomposing this computation is to partition the output matrix C .
- Since each entry of C requires the same amount of computation, we can balance the computations by using either a one- or two-dimensional block distribution to partition C uniformly among the p available processes.
- In the first case, each process will get a block of n/p rows (or columns) of C, whereas in the second case, each process will get a block of size $\frac{n}{\sqrt{p}}$ X $\frac{n}{\sqrt{p}}$.
- In either case, the process will be responsible for computing the entries of the partition of C assigned to it.

- In the case of matrix-matrix multiplication, a one-dimensional distribution will allow us to use up to n processes by assigning a single row of C to each process.
- On the other hand, a two-dimensional distribution will allow us to use up to n^2 processes by assigning a single element of C to each process.

- In addition to allowing a higher degree of concurrency, higher dimensional distributions also sometimes help in reducing the amount of interactions among the different processes for many problems.


A
B
C
X
=
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15

(a)


A
B
C
X
=
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15

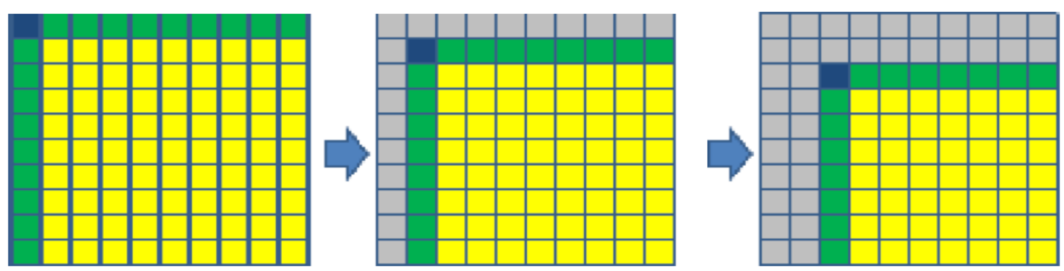
(b)

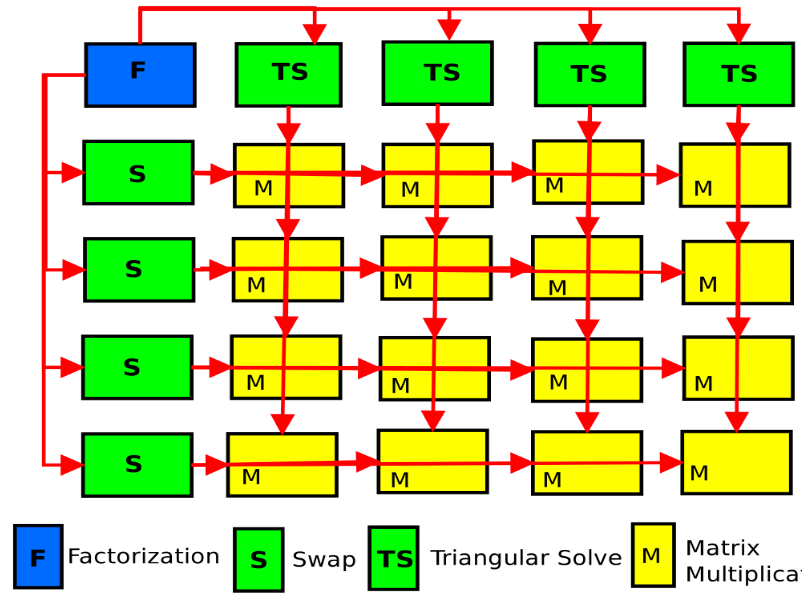
- With a one-dimensional partitioning along the rows, each process needs to access the corresponding n/p rows of matrix A and the entire matrix B, for process P_5.
- Thus the total amount of data that needs to be accessed is $n^2/p + n^2$

- However, with a two-dimensional distribution, each process needs to access $\frac{n}{\sqrt{p}}$ rows of matrix A and $\frac{n}{\sqrt{p}}$ columns of matrix B for process P_5.
- In the two-dimensional case, the total amount of shared data that each process needs to access is $\frac{n^2}{\sqrt{p}}$, which is significantly smaller compared to $O(n^2)$ shared data in the one-dimensional case.

Cyclic and Block Cyclic Distributions

- If the amount of work differs for different elements of a matrix, a block distribution can potentially lead to load imbalances.
- A classic example of this phenomenon is LU factorization of a matrix, in which the amount of computation increases from the top left to the bottom right of the matrix.






```
procedure COL_LU (A)
begin
    for k := 1 to n do
        for j := k to n do
            A[j, k]:= A[j, k]/A[k, k];
        endfor;
        for j := k + 1 to n do
            for i := k + 1 to n do
                A[i, j] := A[i, j] - A[i, k] x A[k, j];
            endfor;
        endfor;
  /*
After this iteration, column A[k + 1 : n, k] is logically the kth
column of L and row A[k, k : n] is logically the kth row of U.
  */
```

LU Factorization of a Dense Matrix

A decomposition of LU factorization into 14 tasks

$$\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} L_{1,1} & 0 & 0 \\ L_{2,1} & L_{2,2} & 0 \\ L_{3,1} & L_{3,2} & L_{3,3} \end{pmatrix} \cdot \begin{pmatrix} U_{1,1} & U_{1,2} & U_{1,3} \\ 0 & U_{2,2} & U_{2,3} \\ 0 & 0 & U_{3,3} \end{pmatrix}$$

1: $A_{1,1} \rightarrow L_{1,1}U_{1,1}$

2: $L_{2,1} = A_{2,1}U_{1,1}^{-1}$

3: $L_{3,1} = A_{3,1}U_{1,1}^{-1}$

4: $U_{1,2} = L_{1,1}^{-1}A_{1,2}$

5: $U_{1,3} = L_{1,1}^{-1}A_{1,3}$

6: $A_{2,2} = A_{2,2} - L_{2,1}U_{1,2}$

7: $A_{3,2} = A_{3,2} - L_{3,1}U_{1,2}$

8: $A_{2,3} = A_{2,3} - L_{2,1}U_{1,3}$

9: $A_{3,3} = A_{3,3} - L_{3,1}U_{1,3}$

10: $A_{2,2} \rightarrow L_{2,2}U_{2,2}$

11: $L_{3,2} = A_{3,2}U_{2,2}^{-1}$

12: $U_{2,3} = L_{2,2}^{-1}A_{2,3}$

13: $A_{3,3} = A_{3,3} - L_{3,2}U_{2,3}$

14: $A_{3,3} \rightarrow L_{3,3}U_{3,3}$

Block Distribution for LU

Notice the significant load imbalance

P_0 T_1	P_3 T_4	P_6 T_5
P_1 T_2	P_4 T_6 T_{10}	P_7 T_8 T_{12}
P_2 T_3	P_5 T_7 T_{11}	P_8 T_9 T_{13} T_{14}

Block Cyclic Distributions

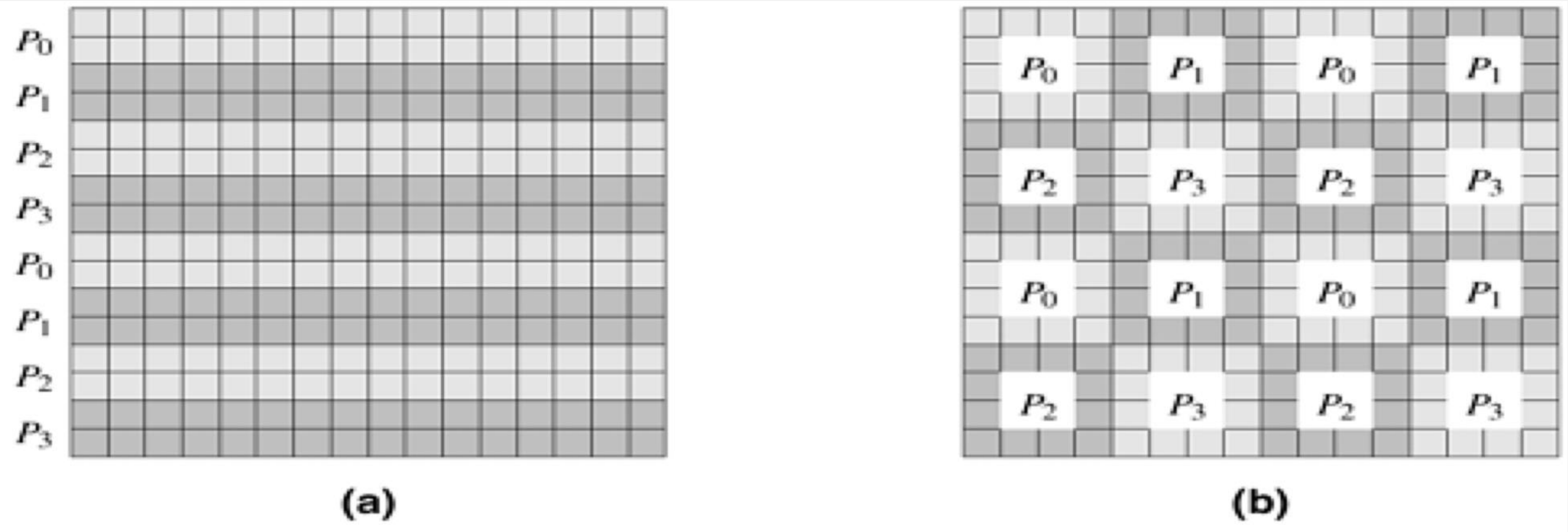
- Variation of the block distribution scheme
  - Partition an array into many more blocks (i.e. tasks) than the number of available processes.
  - Blocks are assigned to processes in a round-robin manner so that each process gets several non-adjacent blocks.
  - N-1 mapping of tasks to processes

Block-Cyclic Distribution for Gaussian Elimination

- Active submatrix shrinks as elimination progresses
- Assigning blocks in a block-cyclic fashion
  - Each PEs receives blocks from different parts of the matrix
  - In one batch of mapping, the PE doing the most will most likely receive the least in the next batch

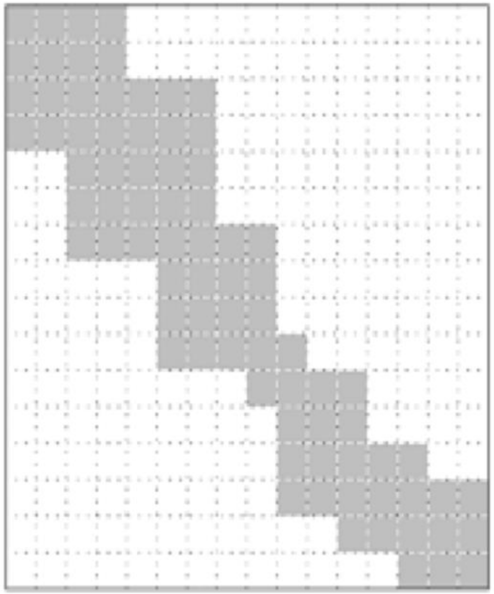
Block-Cyclic Distribution

- A cyclic distribution: a special case with block size = 1
- A block distribution: a special case with block size = ***n/p***
- ***n*** is the dimension of the matrix and ***p*** is the #of processes



Block Partitioning and Random Mapping

- **Sparse matrix computations**



P_0	P_1	P_2	P_3	P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7	P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}	P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}	P_{12}	P_{13}	P_{14}	P_{15}
P_0	P_1	P_2	P_3	P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7	P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}	P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}	P_{12}	P_{13}	P_{14}	P_{15}

(b)

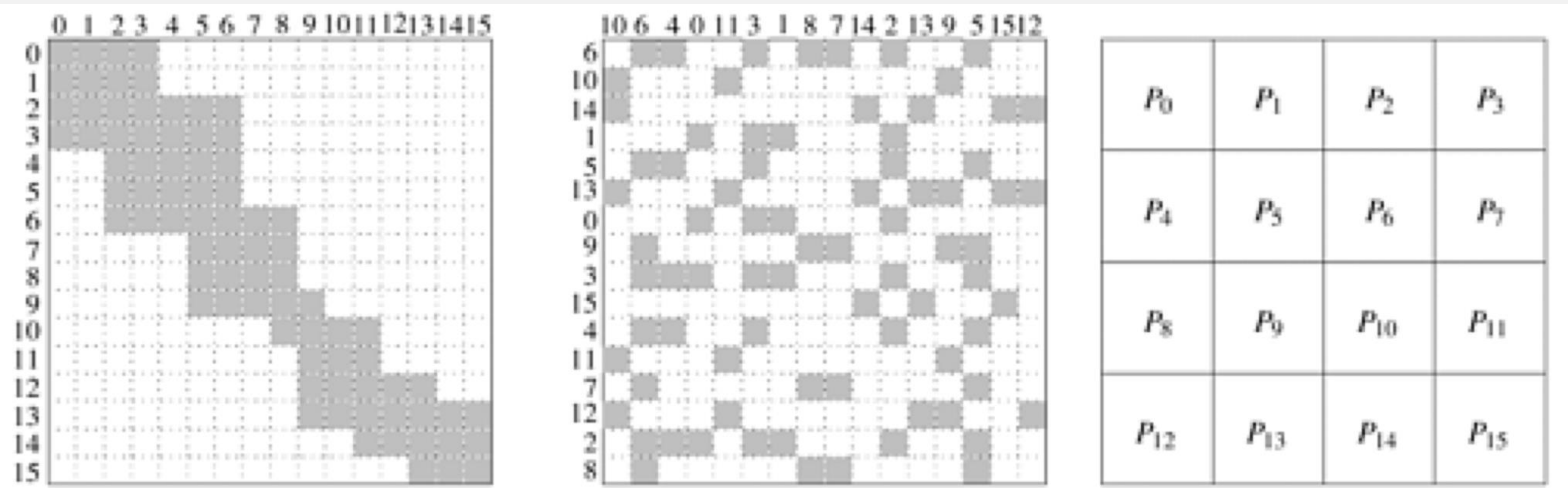
Load imbalance using block-cyclic partitioning/mapping
– more non-zero blocks to diagonal processes *P*0, *P*5, *P*10, and *P*15 than others
– P12 gets nothing

Block Partitioning and Random Mapping

$$V = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$$

$$\text{random}(V) = [8, 2, 6, 0, 3, 7, 11, 1, 9, 5, 4, 10]$$

$$\text{mapping} = \underbrace{8\ \ 2\ \ 6}_{P_0}\ \underbrace{0\ \ 3\ \ 7}_{P_1}\ \underbrace{11\ \ 1\ \ 9}_{P_2}\ \underbrace{5\ \ 4\ \ 10}_{P_3}$$



(a) (b) (c)

Graph Partitioning Based Data Decomposition

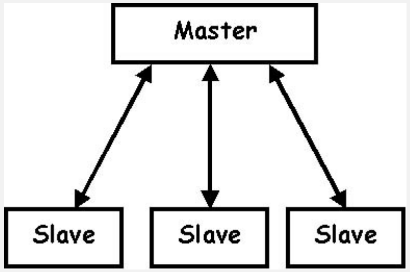
- Array-based partitioning and static mapping
  - Regular domain, i.e. rectangular, mostly dense matrix
  - Structured and regular interaction patterns
  - Quite effective in balancing the computations and minimizing the interactions
- Irregular domain
  - Sparse matrix-related
  - Numerical simulations of physical phenomena
    - Car, water/blood flow, geographic
- Partition the irregular domain so as to
  - Assign equal number of nodes to each process
  - Minimizing edge count of the partition.

Schemes for Dynamic Mapping

- Also referred to as dynamic load balancing
- Load balancing is the primary motivation for dynamic mapping.
- Dynamic mapping schemes can be
  - Centralized
  - Distributed

Centralized Dynamic Mapping

- Processes are designated as **masters** or **slaves**
  - Workers (slave is politically incorrect)
- General strategies
  - Master has pool of tasks and as central dispatcher
  - When one runs out of work, it requests from master for more work.
- Challenge
  - When process # increases, master may become the bottleneck.
- Approach
  - Chunk scheduling: a process picks up multiple tasks at once
  - Chunk size:
    - **Large chunk sizes may lead to significant load imbalances as well**
    - **Schemes to gradually decrease chunk size as the computation progresses.**



Distributed Dynamic Mapping

- All processes are created equal
  - Each can send or receive work from others
    - Alleviates the bottleneck in centralized schemes.
- Four critical design questions:
  - how are sending and receiving processes paired together
  - who initiates work transfer
  - how much work is transferred
  - when is a transfer triggered?

Reference

- http://parallelcomp.uw.hu/ch03lev1sec4.html
- https://passlab.github.io/CSCE569/notes/lecture10_design02.pdf