Lecture 4: Principles of Parallel Algorithm Design (part 4)

Mapping Technique for Load Balancing

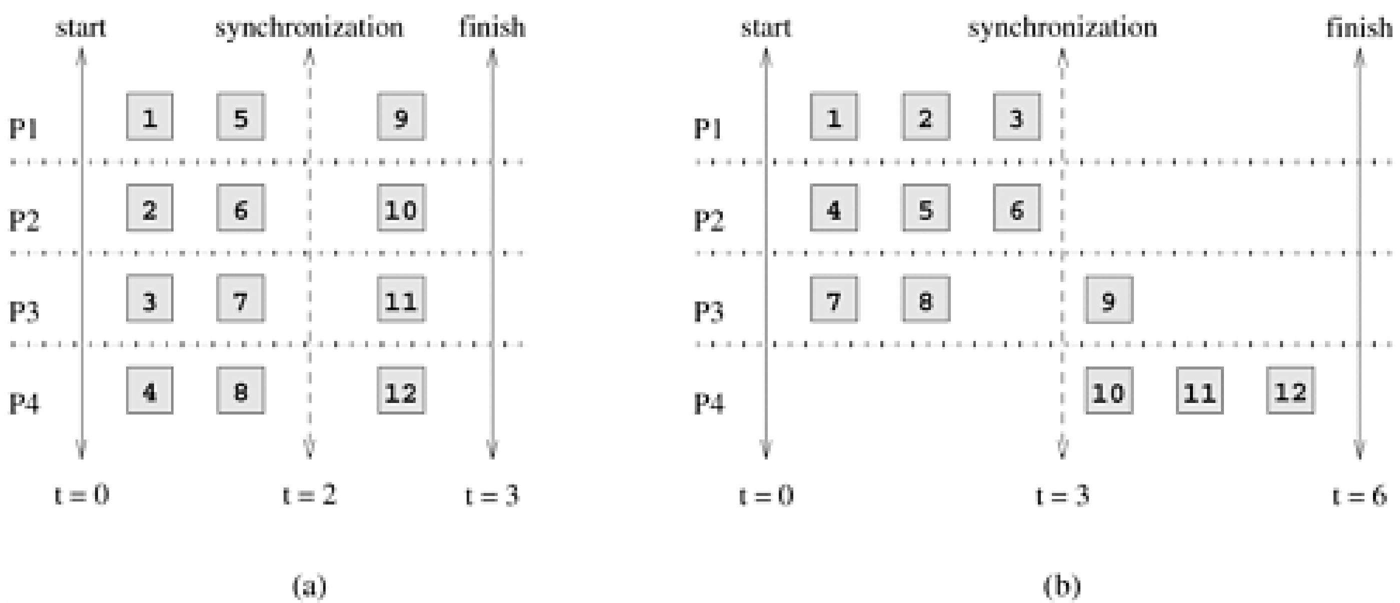
Minimize execution time → Reduce overheads of execution

- Sources of overheads:
  - Inter-process interaction
  - Idling
  - Both interaction and idling are often a function of mapping
- Goals to achieve:
  - To reduce interaction time
  - *To reduce total amount of time some processes being idle (goal of load balancing)*
  - Remark: these two goals often conflict
- Classes of mapping:
  - Static
  - Dynamic

Remark:

1. Loading balancing is **only** a necessary **but not** sufficient condition for reducing idling.
    - Task-dependency graph determines which tasks can execute in parallel and which must wait for some others to finish at a given stage.
2. Good mapping must ensure that computations and interactions among processes at each stage of execution are well balanced.

Figure 3.23. Two mappings of a hypothetical decomposition with a synchronization.



Two mappings of 12-task decomposition in which the last 4 tasks can be started only after the first 8 are finished due to task-dependency.

Schemes for Static Mapping

Static Mapping: It distributes the tasks among processes prior to the execution of the algorithm.

- Mapping Based on Data Partitioning
- Task Graph Partitioning
- Hybrid Strategies

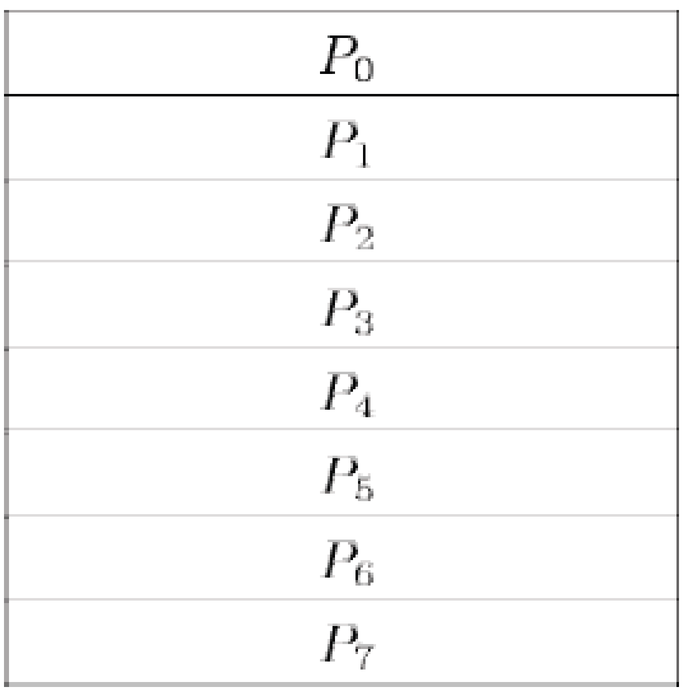
Mapping Based on Data Partitioning

- By owner-computes rule, mapping the relevant data onto processes is equivalent to mapping tasks onto processes
- Array or Matrices
  - Block distributions
  - Cyclic and block cyclic distributions
- Irregular Data
  - Example: data associated with unstructured mesh
  - Graph partitioning

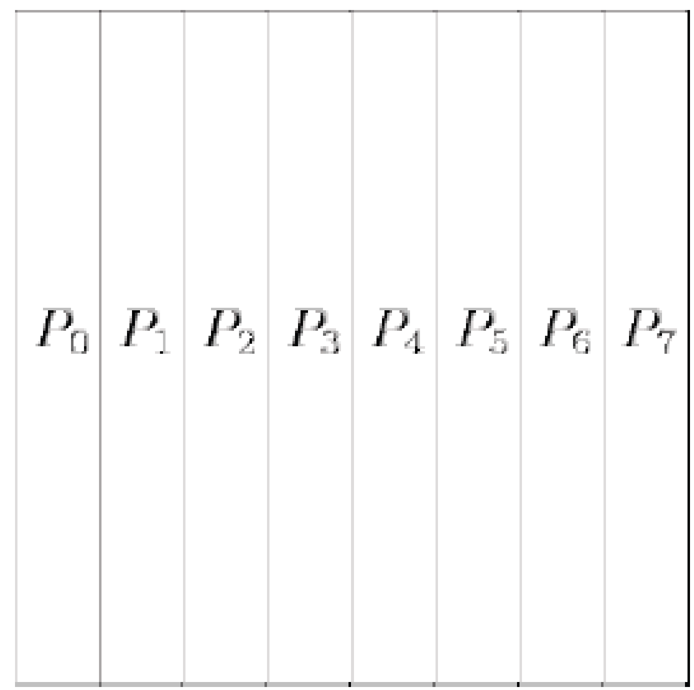
1D Block Distribution

Example. Distribute rows or columns of matrix to different processes

row-wise distribution



column-wise distribution



Multi-D Block Distribution

Example. Distribute blocks of matrix to different processes

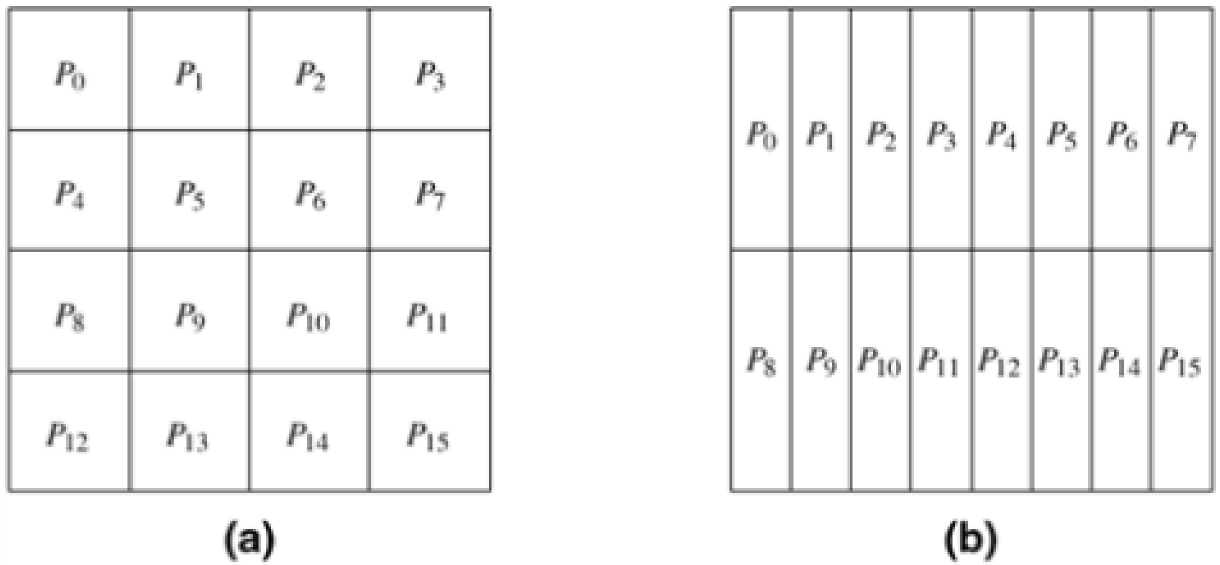


Figure 3.25. Examples of two-dimensional distributions of an array, (a) on a 4 × 4 process grid, and (b) on a 2 × 8 process grid.

Load-Balance for Block Distribution

Example. $n \times n$ dense matrix multiplication $C = A \times B$ using p processes

- Decomposition based on output data.
- Each entry of C use the same amount of computation.
- Either 1D or 2D block distribution can be used:
  - 1D distribution: $\frac{n}{p}$ rows are assigned to a process
  - 2D distribution: $n/\sqrt{p} \times n/\sqrt{p}$ size block is assigned to a process
- Multi-D distribution allows higher degree of concurrency.
- Multi-D distribution can also help to reduce interactions

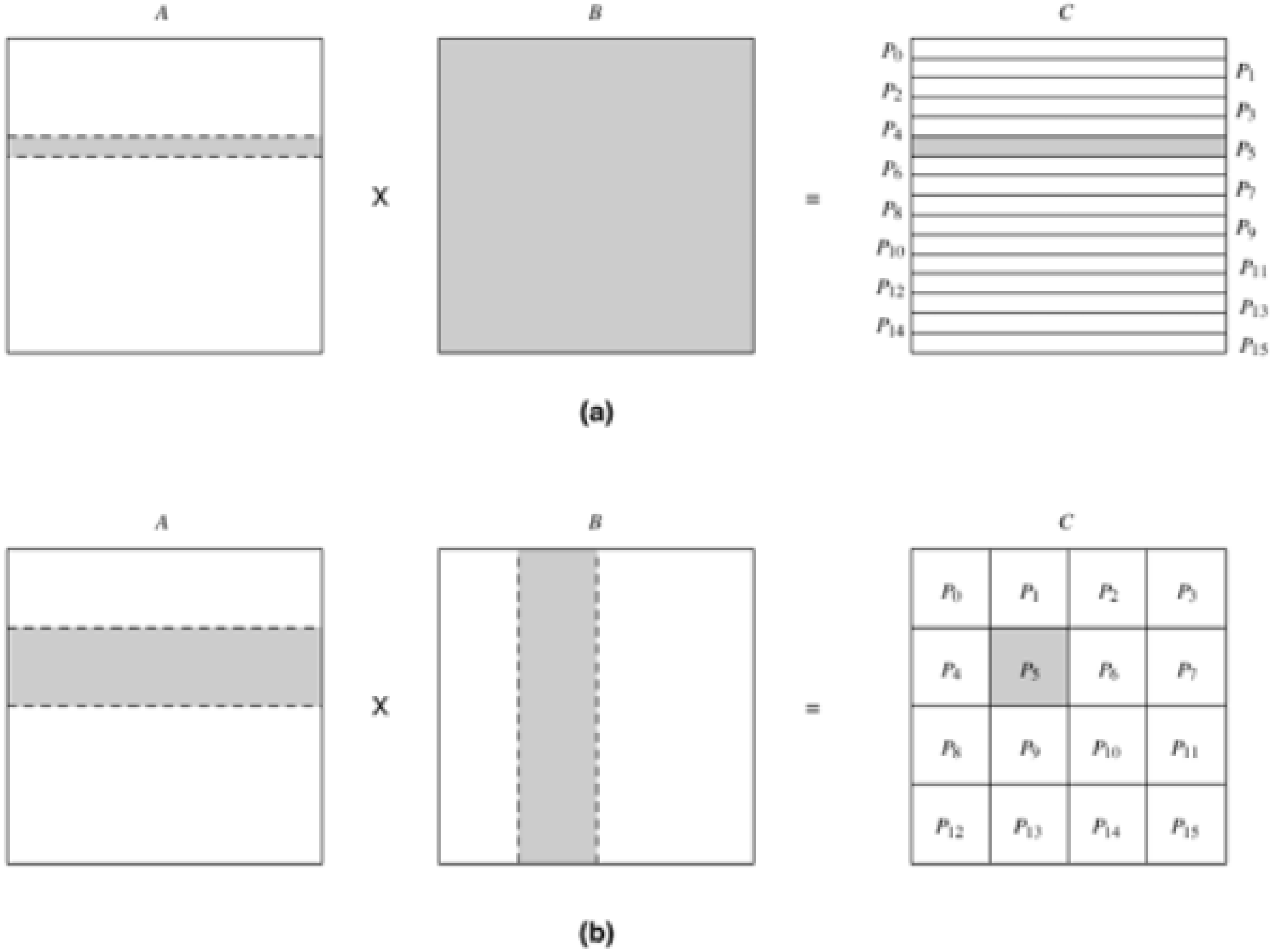


Figure 3.26. Data sharing needed for matrix multiplication with (a) one-dimensional and (b) two-dimensional partitioning of the output matrix. Shaded portions of the input matrices A and B are required by the process that computes the shaded portion of the output matrix C.

Suppose the size of matrix is $n \times n$, and p processes are used.

(a): A process need to access $\frac{n^2}{p} + n^2$ amount of data

(b): A process need to access $O(n^2/\sqrt{p})$ amount of data

Cyclic and Block Cyclic Distributions

- If the amount of work differs for different entries of a matrix, a block distribution can lead to load imbalances.
- Example. Doolittle’s method of LU factorization of dense matrix
  - The amount of computation increases from the top left to the bottom right of the matrix.

Doolittle's method of LU factorization

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = LU = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix}$$

By matrix-matrix multiplication

$u_{1j} = a_{1j},\qquad j = 1,2,\dots,n$ ($1st$ row of U)

$l_{j1} = a_{j1}/u_{11},\qquad j = 1,2,\dots,n$ ($1st$ column of L)

For $i = 2,3,\dots,n-1$ **do**

$u_{ii} = a_{ii} - \sum_{t=1}^{i-1} l_{it}u_{ti}$

$u_{ij} = a_{ij} - \sum_{t=1}^{i-1} l_{it}u_{tj}\qquad$ for $j = i+1,\dots,n$ (ith row of U)

$l_{ji} = \frac{a_{ji} - \sum_{t=1}^{i-1} l_{jt}u_{ti}}{u_{ii}}\qquad$ for $j = i+1,\dots,n$ (ith column of L)

End

$u_{nn} = a_{nn} - \sum_{t=1}^{n-1} l_{nt}u_{tn}$

Serial Column-Based LU

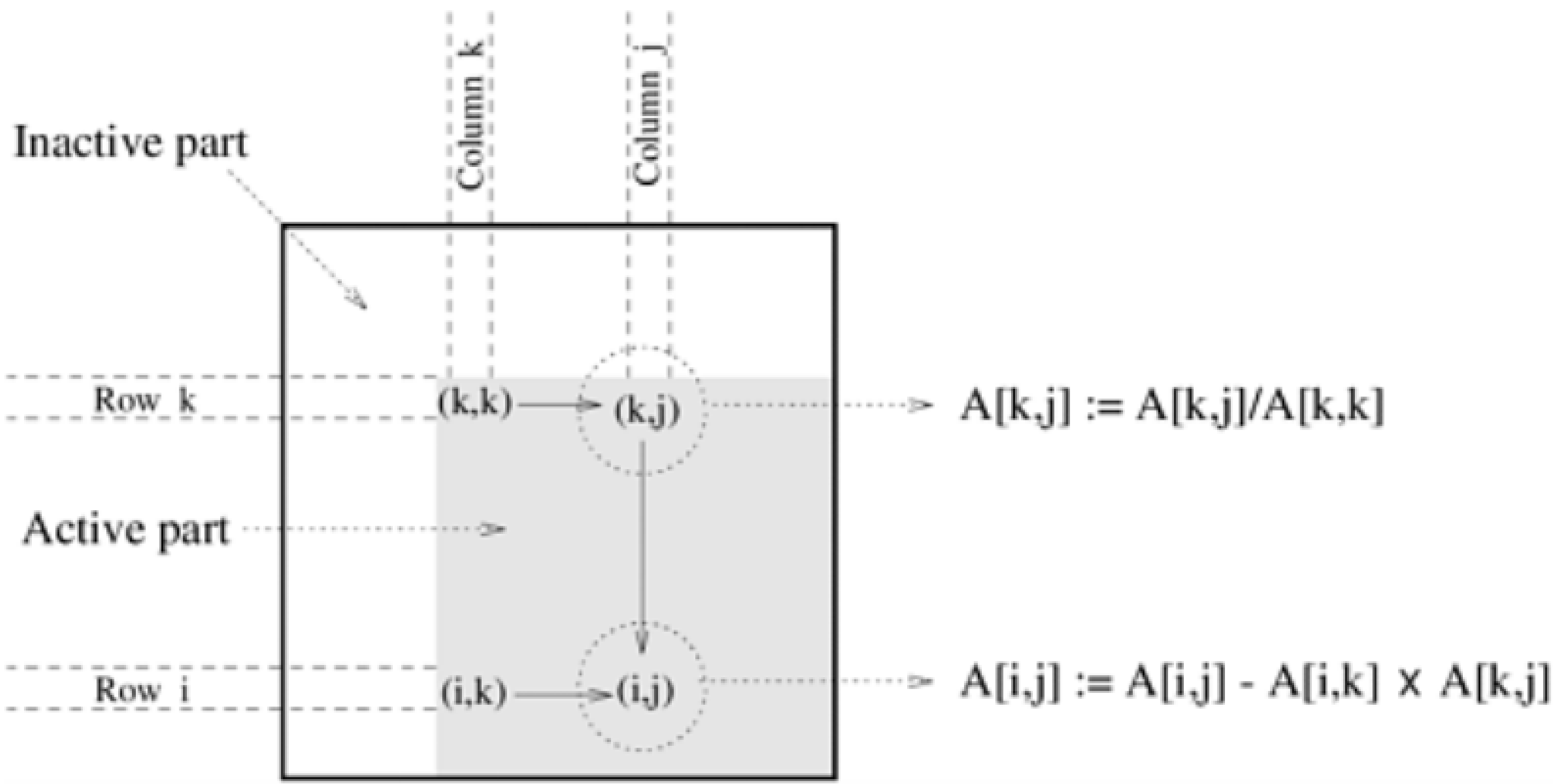
```
procedure COL_LU (A)
begin
   for k := 1 to n do
        for j := k to n do
            A[j, k]:= A[j, k]/A[k, k];
        endfor;
        for j := k + 1 to n do
            for i := k + 1 to n do

                A[i, j] := A[i, j] - A[i, k] × 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.
   */
   endfor;
end COL_LU
```

- Remark: Matrices L and U share space with A

Work used to compute Entries of L and U

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

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



e 3.28. A typical computation in Gaussian elimination and the active part of the coefficient matrix during the kth iteration of the outer loop.

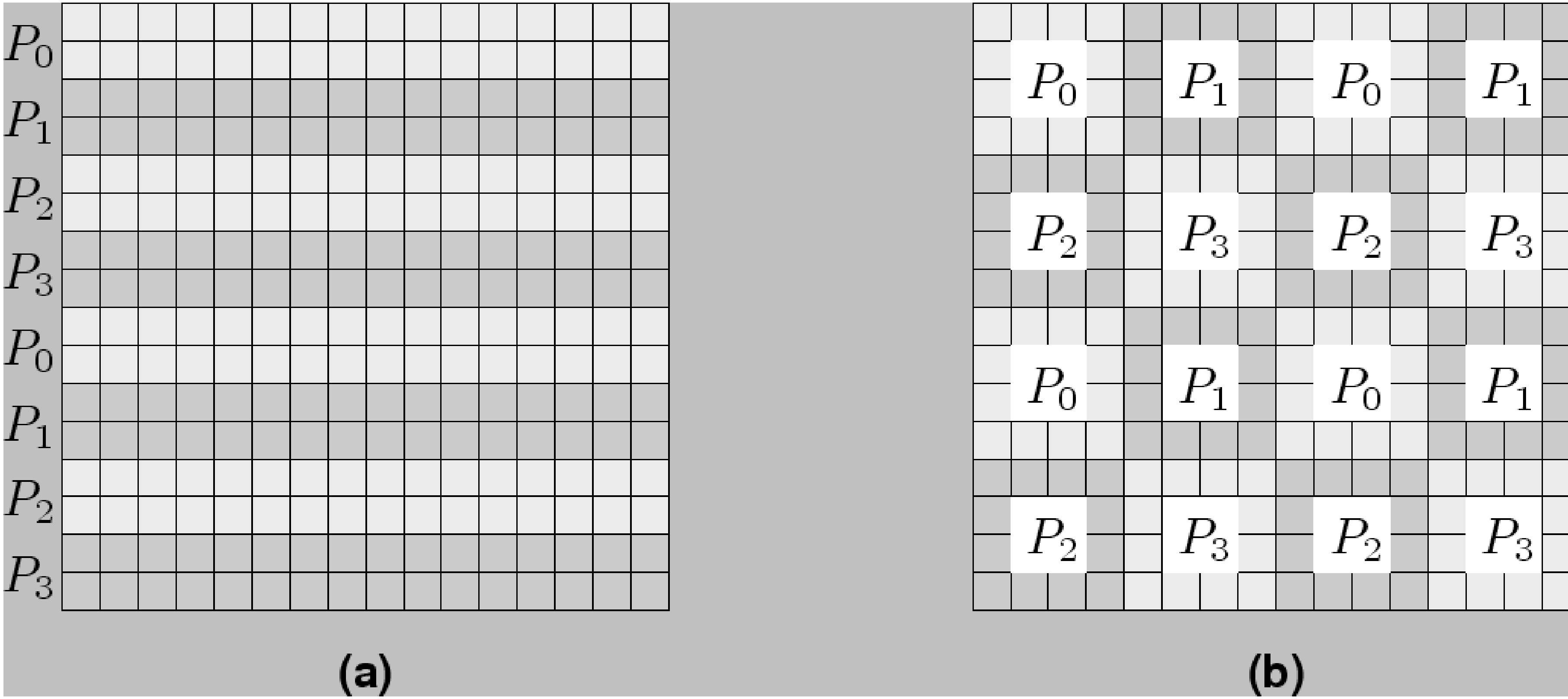
- Block distribution of LU factorization tasks leads to load imbalance.

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 Distribution

- A variation of block distribution that can be used to alleviate the load-imbalance.

- Steps
  1. Partition an array into many more blocks than the number of available processes
  2. Assign blocks to processes in a *round-robin manner* so that each process gets several non-adjacent blocks.



(a) The rows of the array are grouped into blocks each consisting of two rows, resulting in eight blocks of rows. These blocks are distributed to four processes in a *wrap-around* fashion.

(b) The matrix is blocked into 16 blocks each of size 4×4, and it is mapped onto a 2×2 grid of processes in a wraparound fashion.

- **Cyclic distribution:** when the block size =1

Randomized Block Distribution

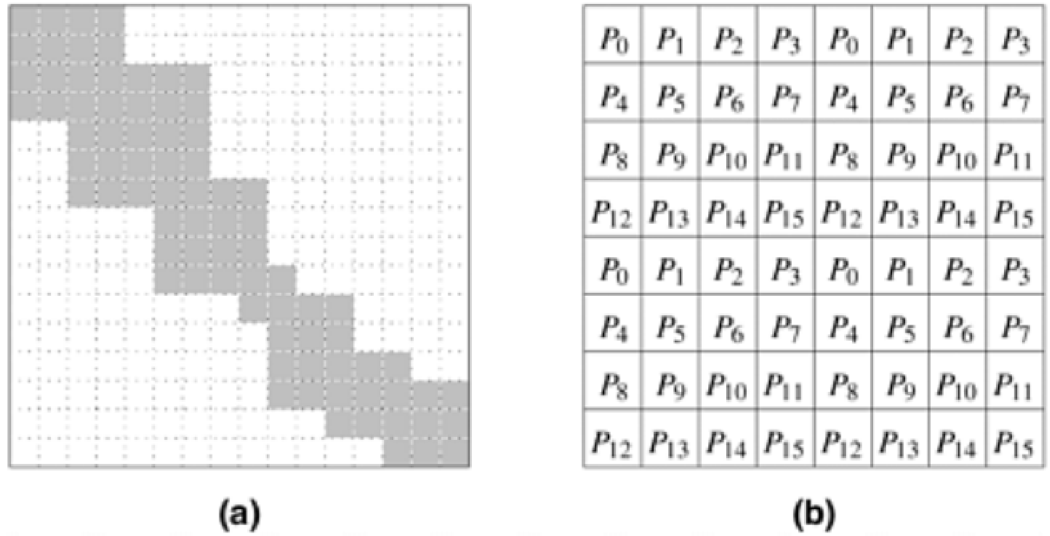


Figure 3.31. Using the block-cyclic distribution shown in (b) to distribute the computations performed in array (a) will lead to load imbalances.

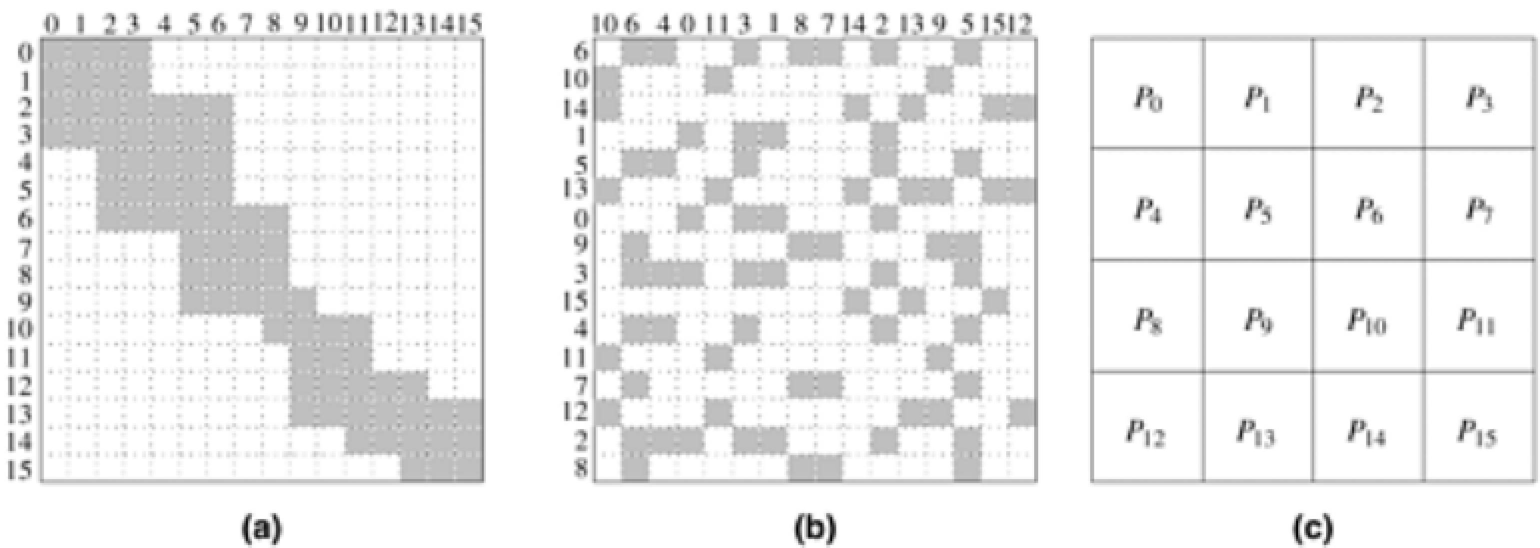
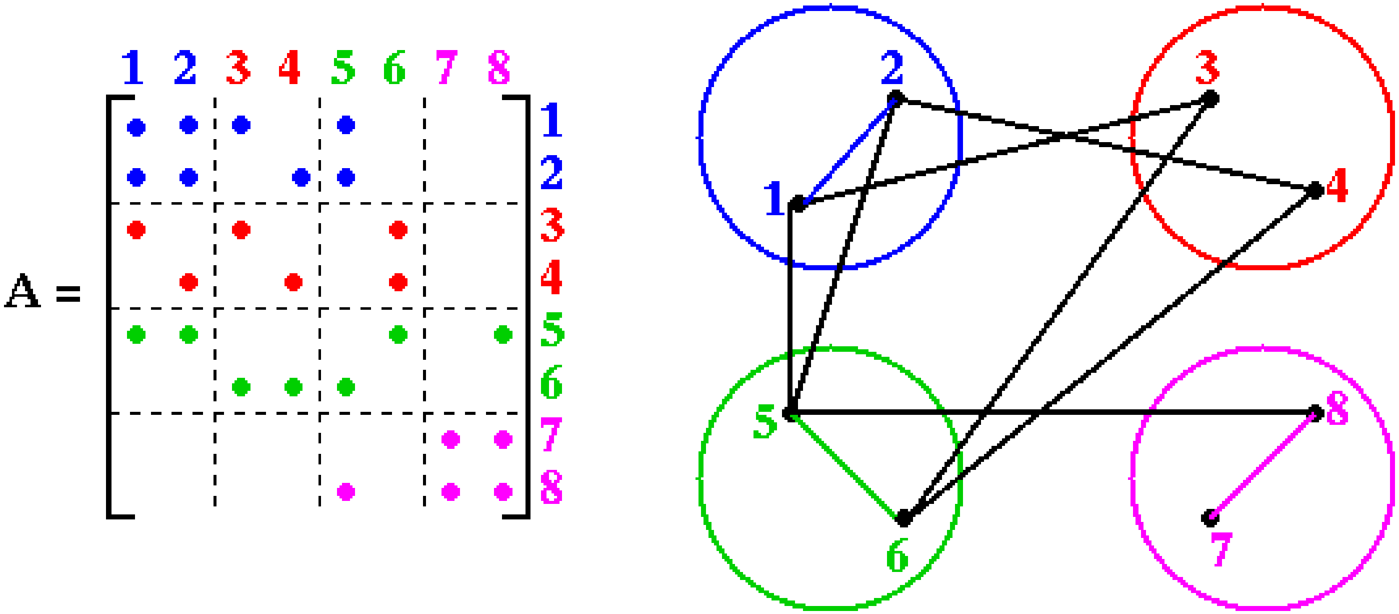


Figure 3.33. Using a two-dimensional random block distribution shown in (b) to distribute the computations performed in array (a), as shown in (c).

Graph Partitioning

Sparse-matrix vector multiplication



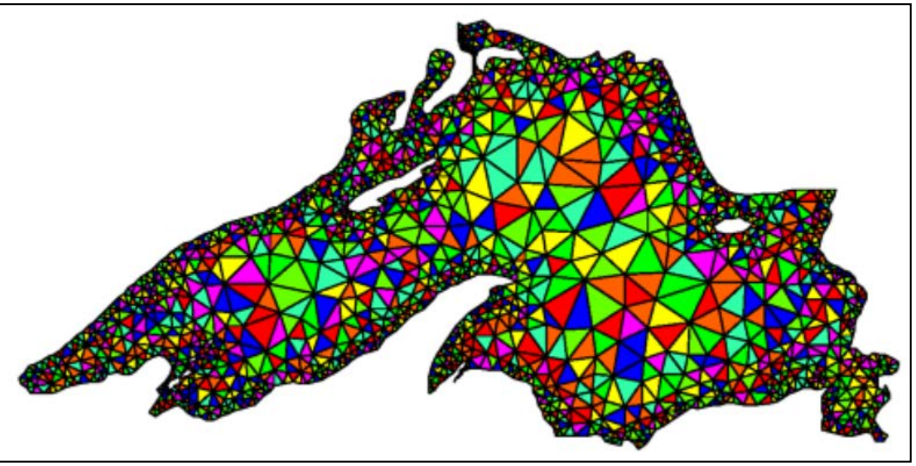
Work: nodes
Interaction/communication: edges

Partition the graph:

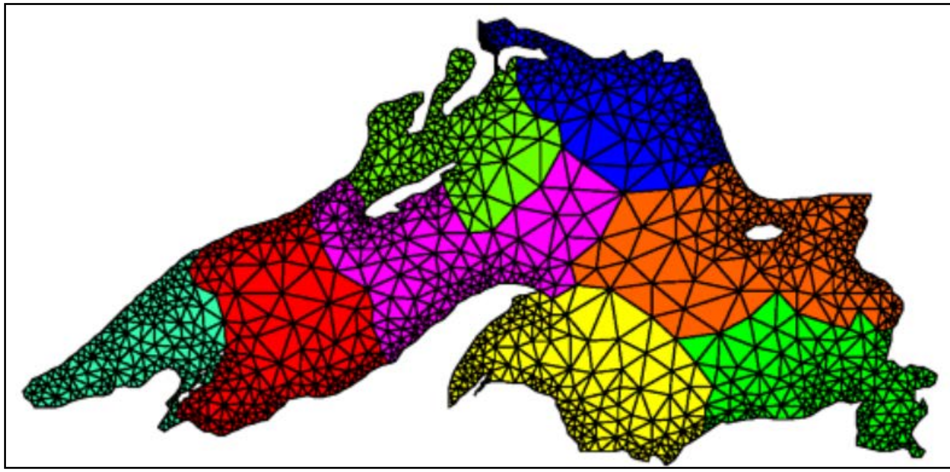
- Assign roughly same number of nodes to each process
- Minimize edge count of graph partition

Finite element simulation of water contaminant in a lake.

- Goal of partitioning: balance work & minimize communication



Random Partitioning



Partitioning for Minimizing Edge-Count

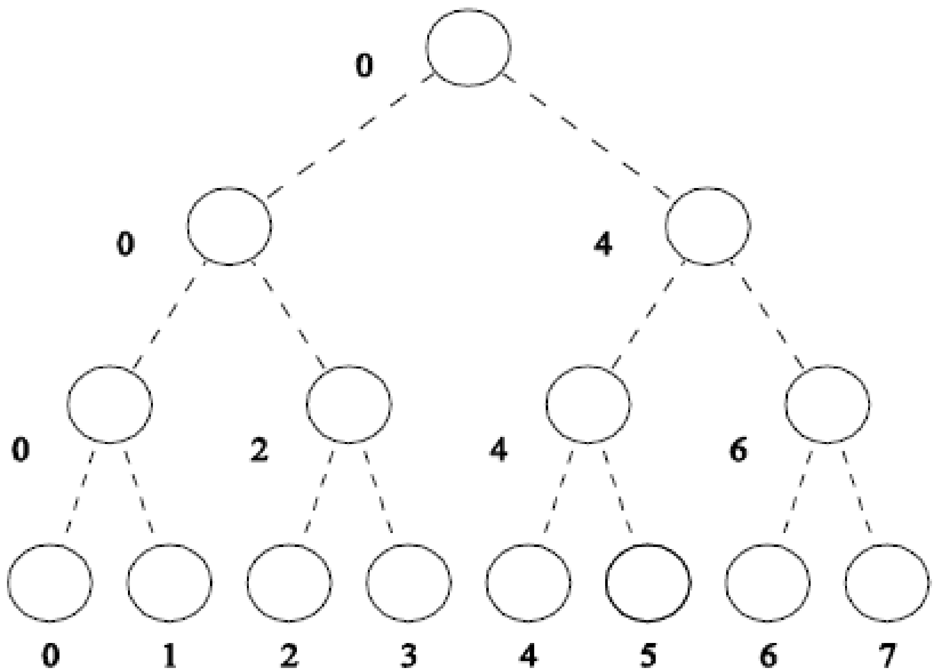
- Assign equal number of nodes (or cells) to each process
  - Random partitioning may lead to high interaction overhead due to data sharing
- Minimize edge count of the graph partition
  - Each process should get roughly the same number of elements and the number of edges that cross partition boundaries should be minimized as well.

Mappings Based on Task Partitioning

- Mapping based on task partitioning can be used when computation is naturally expressed in the form of a *static task-dependency graph* with known sizes.
- Finding optimal mapping minimizing idle time and minimizing interaction time is NP-complete
- Heuristic solutions exist for many structured graphs

Mapping a Binary Tree Task-Dependency Graph

- Finding minimum using hypercube network.
  - Hypercube: node numbers that differ in 1 bit are adjacent.

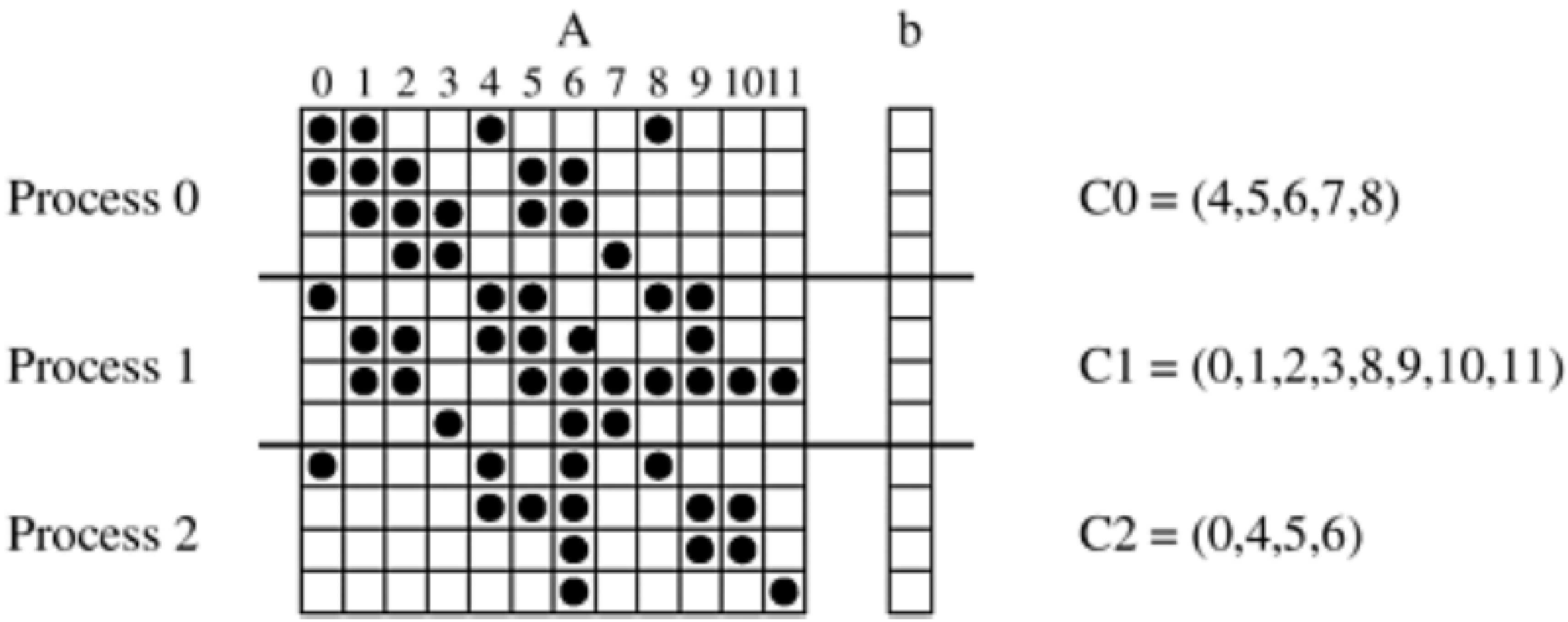


- Mapping the tree graph onto 8 processes
- Mapping minimizes the interaction overhead by mapping inter-dependent tasks onto the same process (i.e., process 0) and others on processes only one communication link away from each other
- Idling exists. This is inherent in the graph

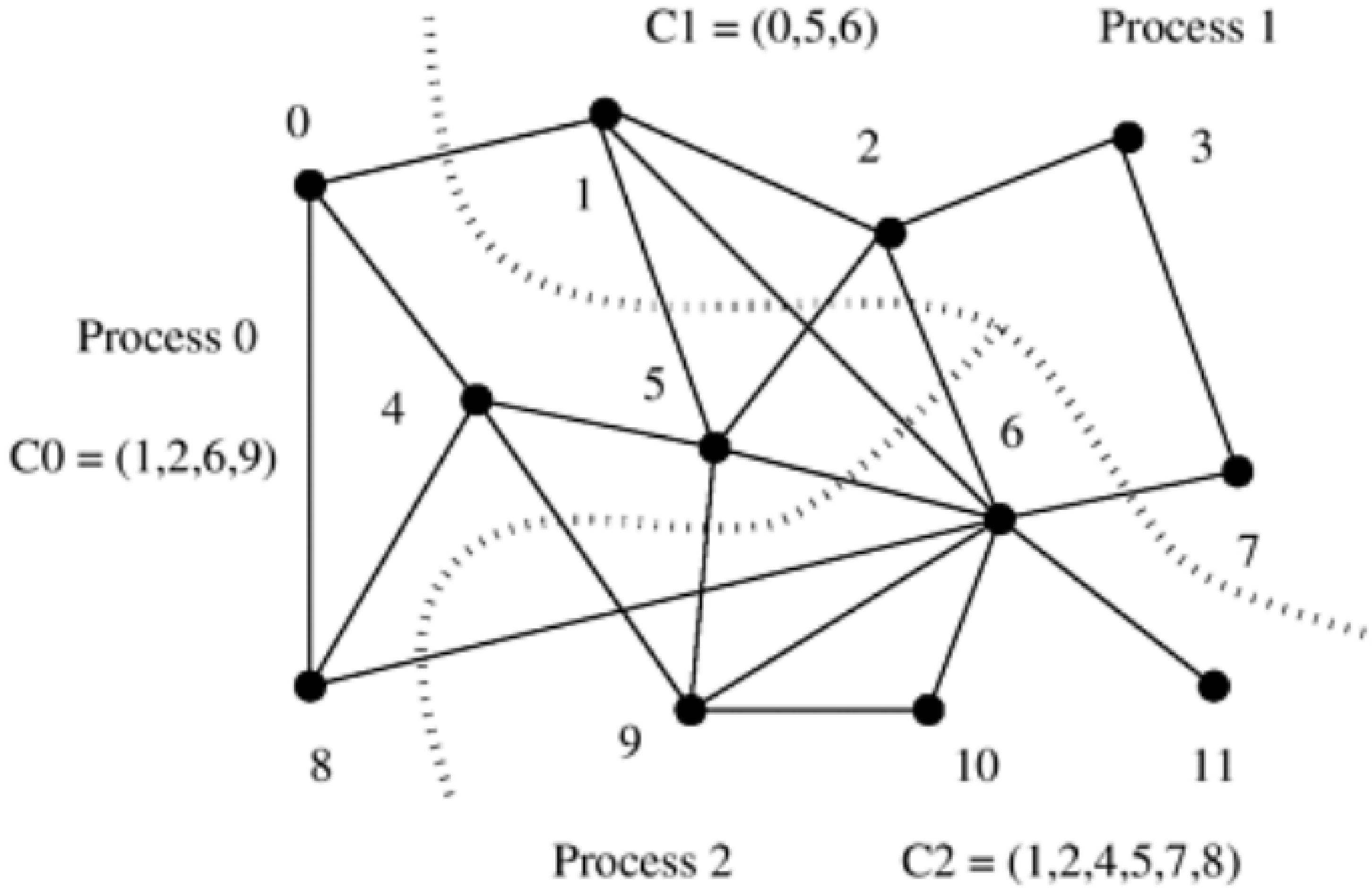
Mapping a Sparse Graph

Example. Sparse matrix-vector multiplication using 3 processes

- Arrow distribution



- Partitioning task-interaction graph to reduce interaction overhead



Schemes for Dynamic Mapping

- When static mapping results in highly imbalanced distribution of work among processes or when task-dependency graph is dynamic, use dynamic mapping
- Primary goal is to balance load – dynamic load balancing
  - Example: Dynamic load balancing for AMR
- Types
  - Centralized
  - Distributed

Centralized Dynamic Mapping

- Processes
  - Master: mange a group of available tasks
  - Slave: depend on master to obtain work
- Idea
  - When a slave process has no work, it takes a portion of available work from master
  - When a new task is generated, it is added to the pool of tasks in the master process
- Potential problem
  - When many processes are used, master process may become bottleneck
- Solution
  - Chunk scheduling: every time a process runs out of work it gets a group of tasks.

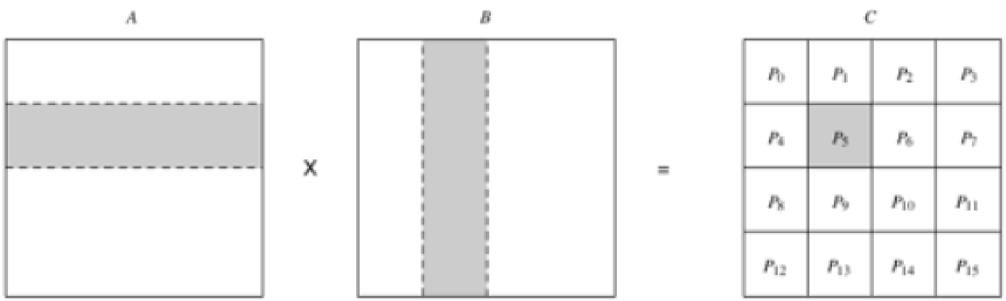
Distributed Dynamic Mapping

- All processes are peers. Tasks are distributed among processes which exchange tasks at run time to balance work
- Each process can send or receive work from other processes
  - How are sending and receiving processes paired together
  - Is the work transfer initiated by the sender or the receiver?
  - How much work is transferred?
  - When is the work transfer performed?

Techniques to Minimize Interaction Overheads

- Maximize data locality
  - Maximize the reuse of recently accessed data
  - Minimize volume of data-exchange
    - Use high dimensional distribution. Example: 2D block distribution for matrix multiplication
  - Minimize frequency of interactions
    - Reconstruct algorithm such that shared data are accessed and used in large pieces.
    - Combine messages between the same source-destination pair

- Minimize contention and hot spots
  - Competition occur when multi-tasks try to access the same resources concurrently: multiple processes sending message to the same process; multiple simultaneous accesses to the same memory block



- Using $C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,k} B_{k,j}$ causes contention. For example, $C_{0,0}$, $C_{0,1}, C_{0,\sqrt{p}-1}$ attempt to read $A_{0,0}$, at the same time.
- A contention-free manner is to use:

$$C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,(i+j+k)\%\sqrt{p}} B_{(i+j+k)\%\sqrt{p},j}$$

All tasks $P_{*,j}$ that work on the same row of C access block $A_{i,(i+j+k)\%\sqrt{p}}$, which is different for each task.

- Overlap computations with interactions
  - Use non-blocking communication
- Replicate data or computations
  - Some parallel algorithm may have read-only access to shared data structure. If local memory is available, replicate a copy of shared data on each process if possible, so that there is only initial interaction during replication.
- Use collective interaction operations
- Overlap interactions with other interactions

Parallel Algorithm Models

- Data parallel
  - Each task performs similar operations on different data
  - Typically statically map tasks to processes
- Task graph
  - Use task dependency graph to promote locality or reduce interactions
- Master-slave
  - One or more master processes generating tasks
  - Allocate tasks to slave processes
  - Allocation may be static or dynamic
- Pipeline/producer-consumer
  - Pass a stream of data through a sequence of processes
  - Each performs some operation on it
- Hybrid
  - Apply multiple models hierarchically, or apply multiple models in sequence to different phases