Principles of Parallel Algorithm Design

Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar

To accompany the text "Introduction to Parallel Computing",
Addison Wesley, 2003.

Characteristics of Tasks

Once a problem has been decomposed into independent tasks, the characteristics of these tasks critically impact choice and performance of parallel algorithms. Relevant task characteristics include:

- Task generation.
- Task sizes.
- Size of data associated with tasks.

Task Generation

- Static task generation: Concurrent tasks can be identified a-priori. Typical matrix operations, graph algorithms, image processing applications, and other regularly structured problems fall in this class. These can typically be decomposed using data or recursive decomposition techniques.

- Dynamic task generation: Tasks are generated as we perform computation. A classic example of this is in game playing – each 15 puzzle board is generated from the previous one. These applications are typically decomposed using exploratory or speculative decompositions.

Task Sizes

- Task sizes may be uniform (i.e., all tasks are the same size) or non-uniform.
- Non-uniform task sizes may be such that they can be determined (or estimated) a-priori or not.
- Examples in this class include discrete optimization problems, in which it is difficult to estimate the effective size of a state space.

Size of Data Associated with Tasks

- The size of data associated with a task may be small or large when viewed in the context of the size of the task.

- A small context of a task implies that an algorithm can easily communicate this task to other processes dynamically (e.g., the 15 puzzle).

- A large context ties the task to a process, or alternately, an algorithm may attempt to reconstruct the context at another processes as opposed to communicating the context of the task (e.g., 0/1 integer programming).

Characteristics of Task Interactions

Tasks may communicate with each other in various ways. The associated dichotomy is:

- Static interactions: The tasks and their interactions are known a-priori. These are relatively simpler to code into programs.

- Dynamic interactions: The timing or interacting tasks cannot be determined a-priori. These interactions are harder to code, especitally, as we shall see, using message passing APIs.

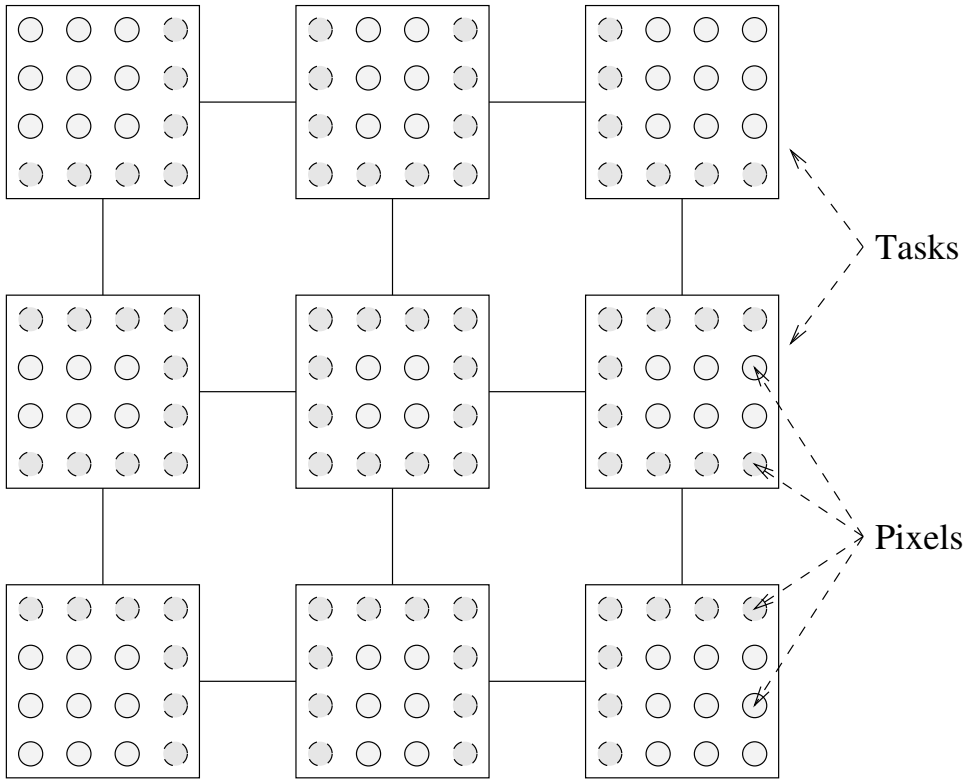
Characteristics of Task Interactions

- Regular interactions: There is a definite pattern (in the graph sense) to the interactions. These patterns can be exploited for efficient implementation.

- Irregular interactions: Interactions lack well-defined topologies.

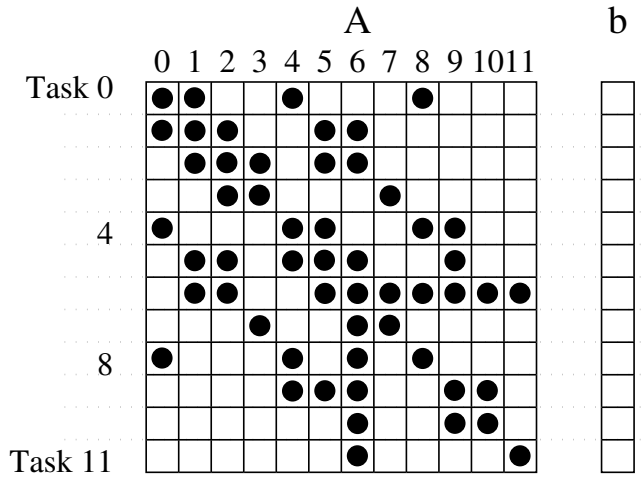
Characteristics of Task Interactions: Example

A simple example of a regular static interaction pattern is in image dithering. The underlying communication pattern is a structured (2-D mesh) one as shown here:

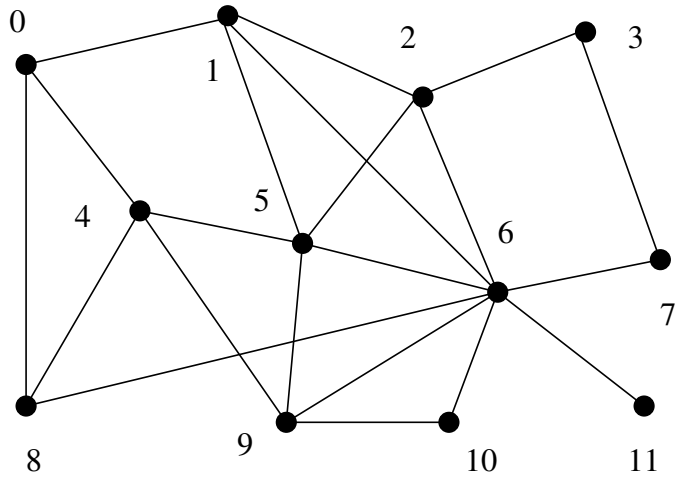


Characteristics of Task Interactions: Example

The multiplication of a sparse matrix with a vector is a good example of a static irregular interaction pattern. Here is an example of a sparse matrix and its associated interaction pattern.



(a)



(b)

Characteristics of Task Interactions

- Interactions may be read-only or read-write.
- In read-only interactions, tasks just read data items associated with other tasks.
- In read-write interactions tasks read, as well as modily data items associated with other tasks.
- In general, read-write interactions are harder to code, since they require additional synchronization primitives.

Characteristics of Task Interactions

- Interactions may be one-way or two-way.
- A one-way interaction can be initiated and accomplished by one of the two interacting tasks.
- A two-way interaction requires participation from both tasks involved in an interaction.
- One way interactions are somewhat harder to code in message passing APIs.

Mapping Techniques

- Once a problem has been decomposed into concurrent tasks, these must be mapped to processes (that can be executed on a parallel platform).

- Mappings must minimize overheads.

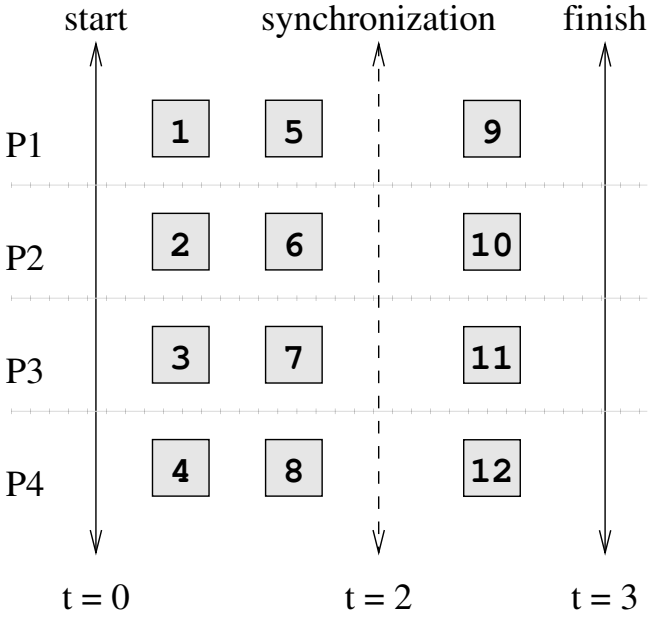
- Primary overheads are communication and idling.

- Minimizing these overheads often represents contradicting objectives.

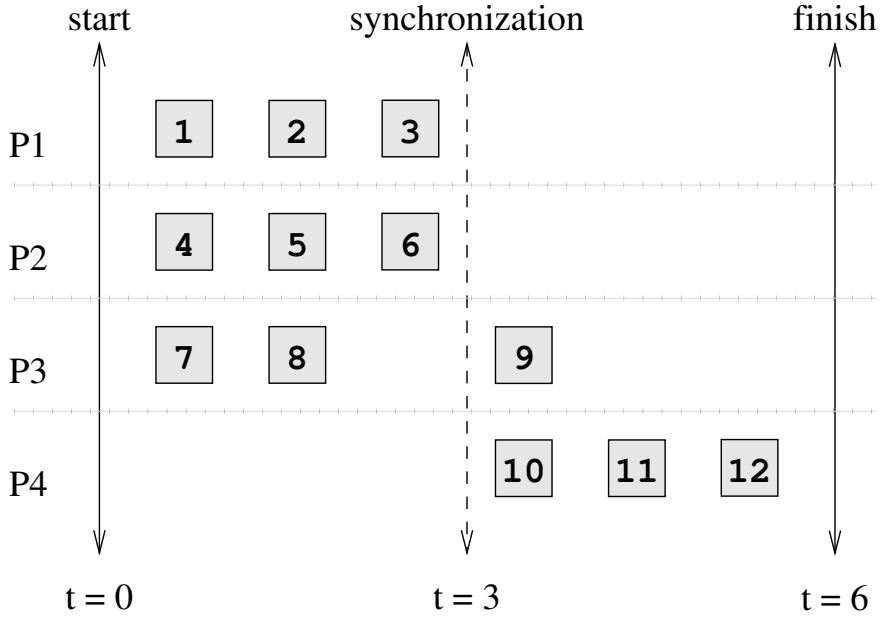
- Assigning all work to one processor trivially minimizes communication at the expense of significant idling.

Mapping Techniques for Minimum Idling

Mapping must simultaneously minimize idling and load balance. Merely balancing load does not minimize idling.



(a)



(b)

Mapping Techniques for Minimum Idling

Mapping techniques can be static or dynamic.

- Static Mapping: Tasks are mapped to processes a-priori. For this to work, we must have a good estimate of the size of each task. Even in these cases, the problem may be NP complete.

- Dynamic Mapping: Tasks are mapped to processes at runtime. This may be because the tasks are generated at runtime, or that their sizes are not known.

Other factors that determine the choice of techniques include the size of data associated with a task and the nature of underlying domain.

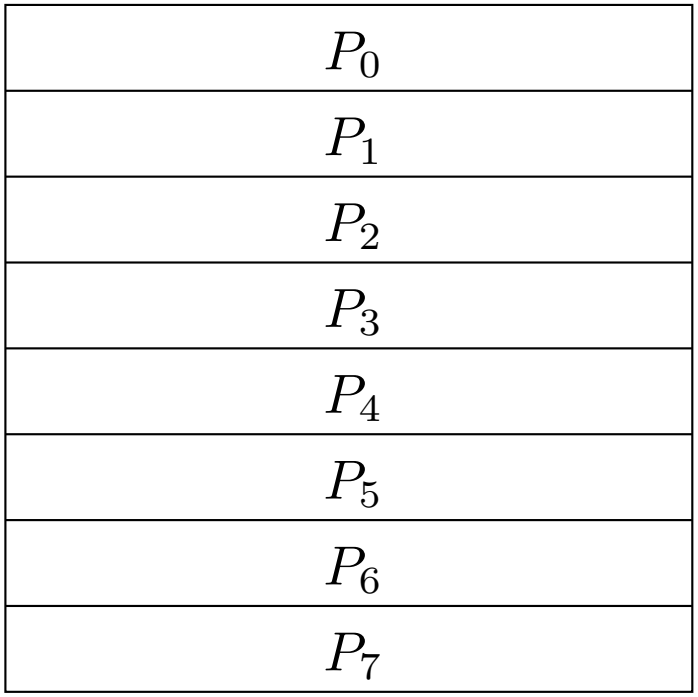
Schemes for Static Mapping

- Mappings based on data partitioning.
- Mappings based on task graph partitioning.
- Hybrid mappings.

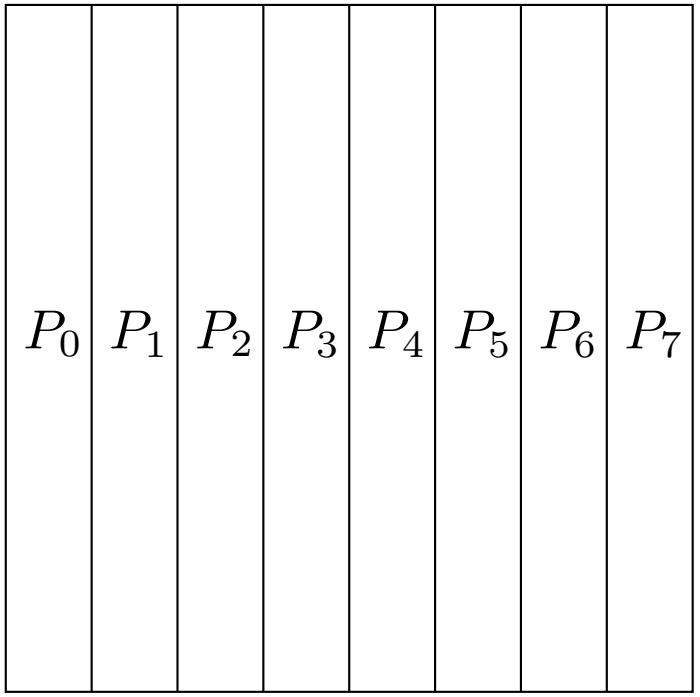
Mappings Based on Data Partitioning

We can combine data partitioning with the "owner-computes" rule to partition the computation into subtasks. The simplest data decomposition schemes for dense matrices are 1-D block distribution schemes.

row-wise distribution



column-wise distribution



Block Array Distribution Schemes

Block distribution schemes can be generalized to higher dimensions as well.

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

(a)

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

(b)

Block Array Distribution Schemes: Examples

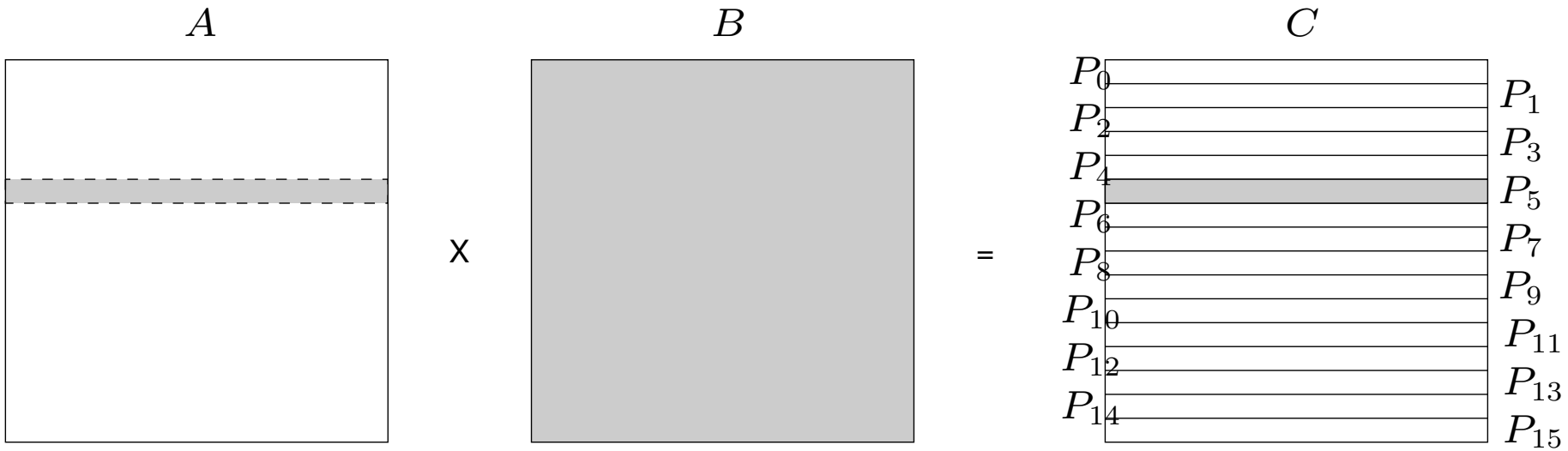
- For multiplying two dense matrices A and B, we can partition the output matrix C using a block decomposition.

- For load balance, we give each task the same number of elements of C. (Note that each element of C corresponds to a single dot product.)

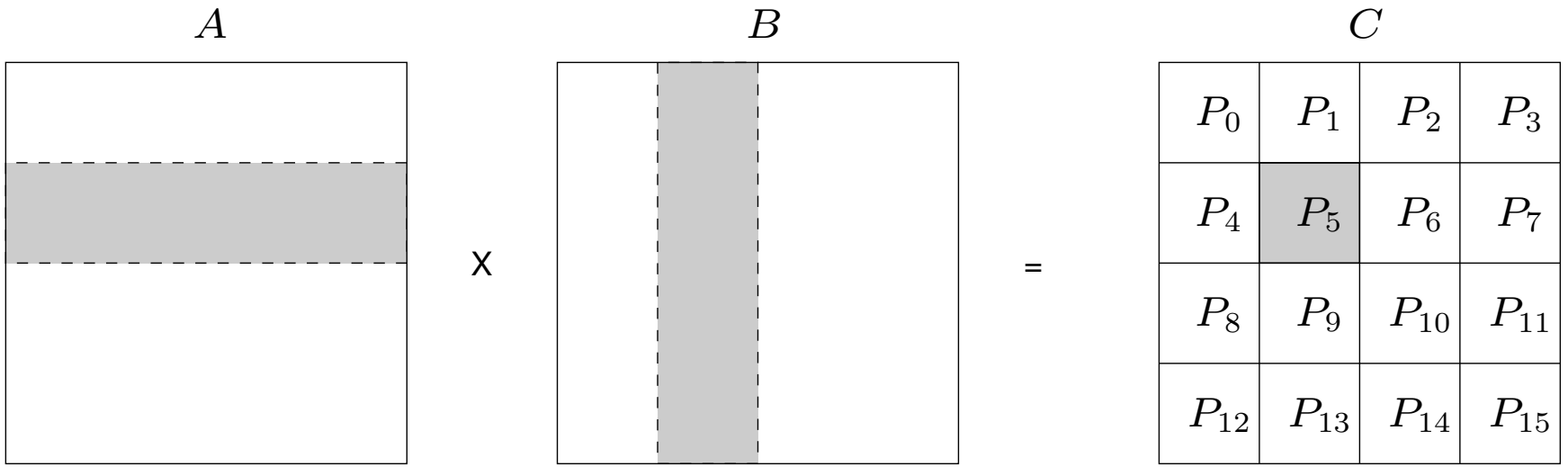
- The choice of precise decomposition (1-D or 2-D) is determined by the associated communication overhead.

- In general, higher dimension decomposition allows the use of larger number of processes.

Data Sharing in Dense Matrix Multiplication



(a)



(b)

Cyclic and Block Cyclic Distributions

- If the amount of computation associated with data items varies, a block decomposition may lead to significant load imbalances.

- A simple example of this is in LU decomposition (or Gaussian Elimination) of dense matrices.

LU Factorization of a Dense Matrix

A decomposition of LU factorization into 14 tasks – notice the significant load imbalance.

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

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 Cyclic Distributions

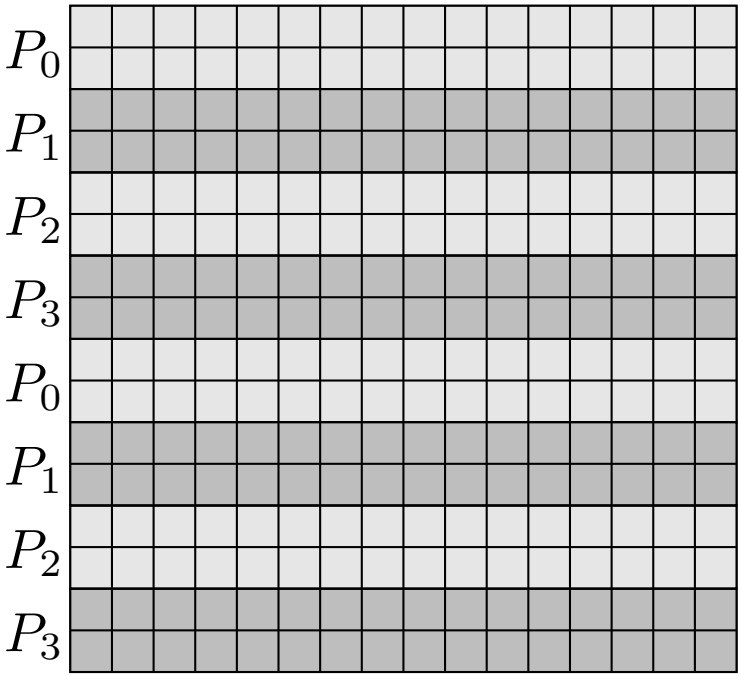
- Variation of the block distribution scheme that can be used to alleviate the load-imbalance and idling problems.

- Partition an array into many more blocks than the number of available processes.

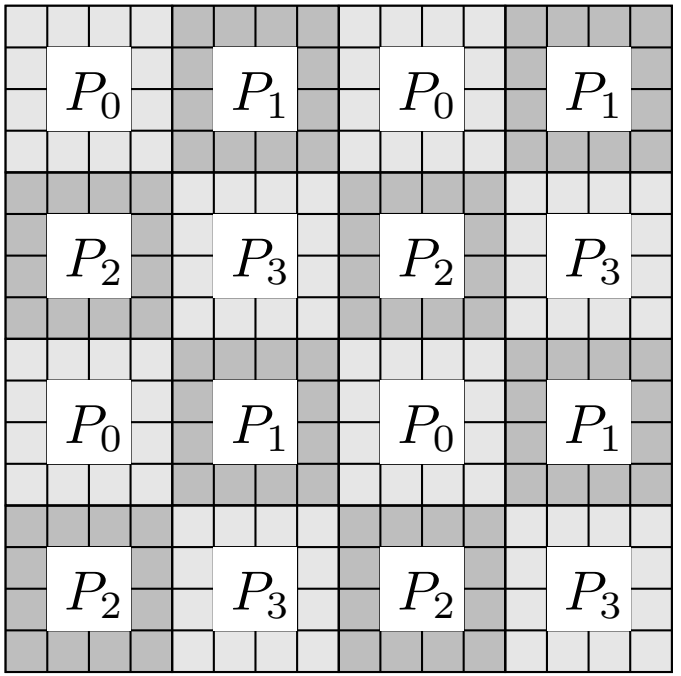
- Blocks are assigned to processes in a round-robin manner so that each process gets several non-adjacent blocks.

Block-Cyclic Distribution

- A cyclic distribution is a special case in which block size is one.
- A block distribution is a special case in which block size is n/p, where n is the dimension of the matrix and p is the number of processes.



(a)

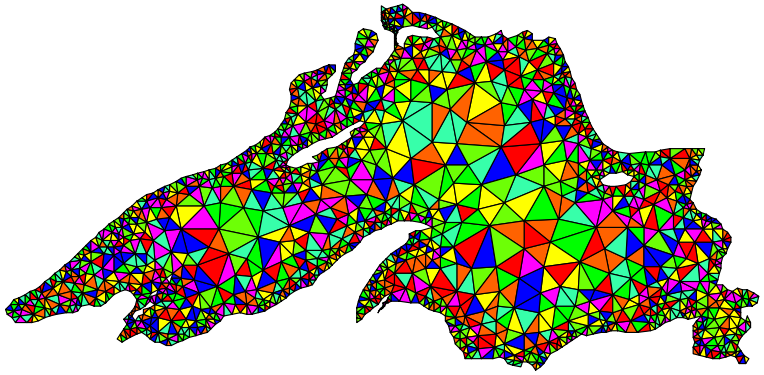


(b)

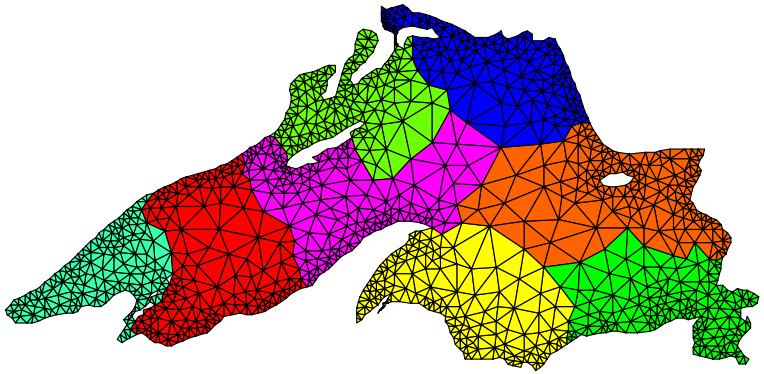
Graph Partitioning Dased Data Decomposition

- In case of sparse matrices, block decompositions are more complex.
- Consider the problem of multiplying a sparse matrix with a vector.
- The graph of the matrix is a useful indicator of the work (number of nodes) and communication (the degree of each node).
- In this case, we would like to partition the graph so as to assign equal number of nodes to each process, while minimizing edge count of the graph partition.

Partitioning the Graph of Lake Superior



Random Partitioning



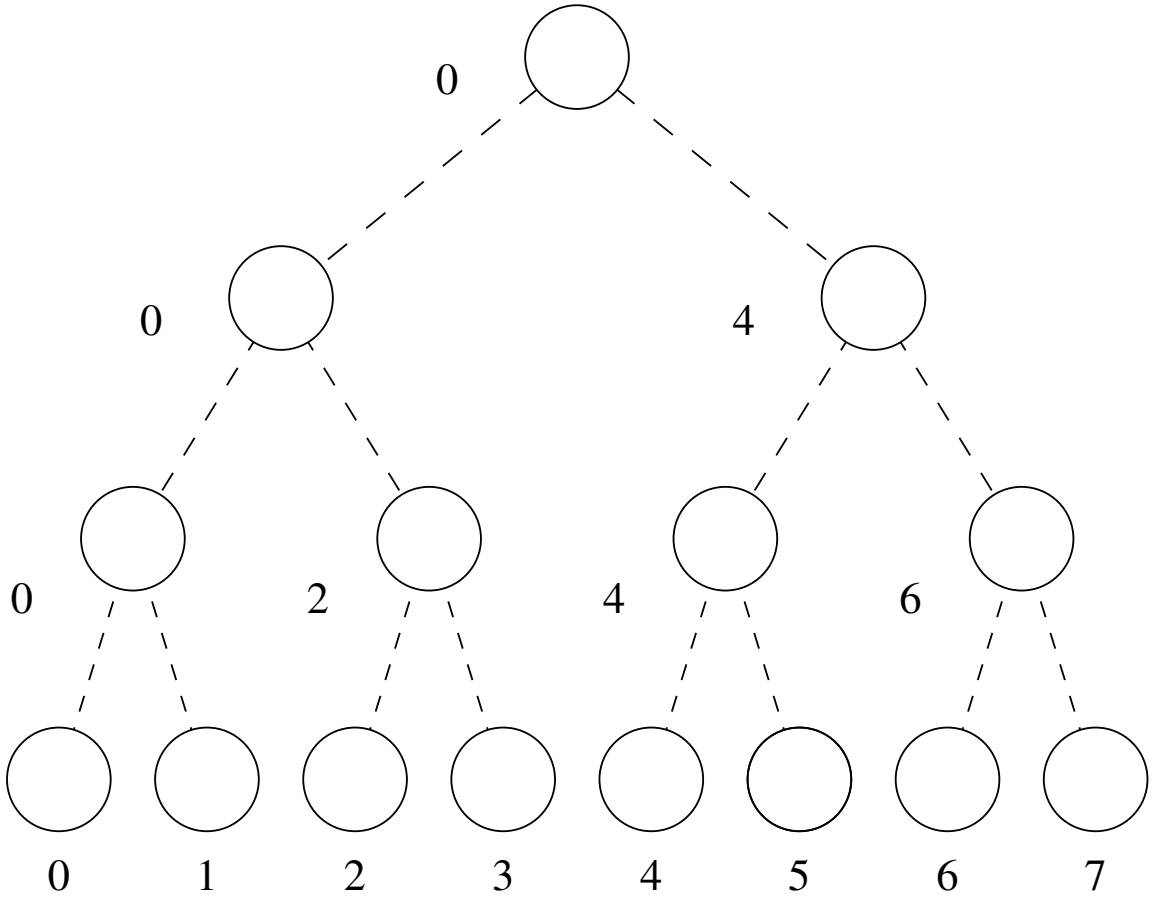
Partitioning for minimum edge-cut.

Mappings Based on Task Paritioning

- Partitioning a given task-dependency graph across processes.
- Determining an optimal mapping for a general task-dependency graph is an NP-complete problem.
- Excellent heuristics exist for structured graphs.

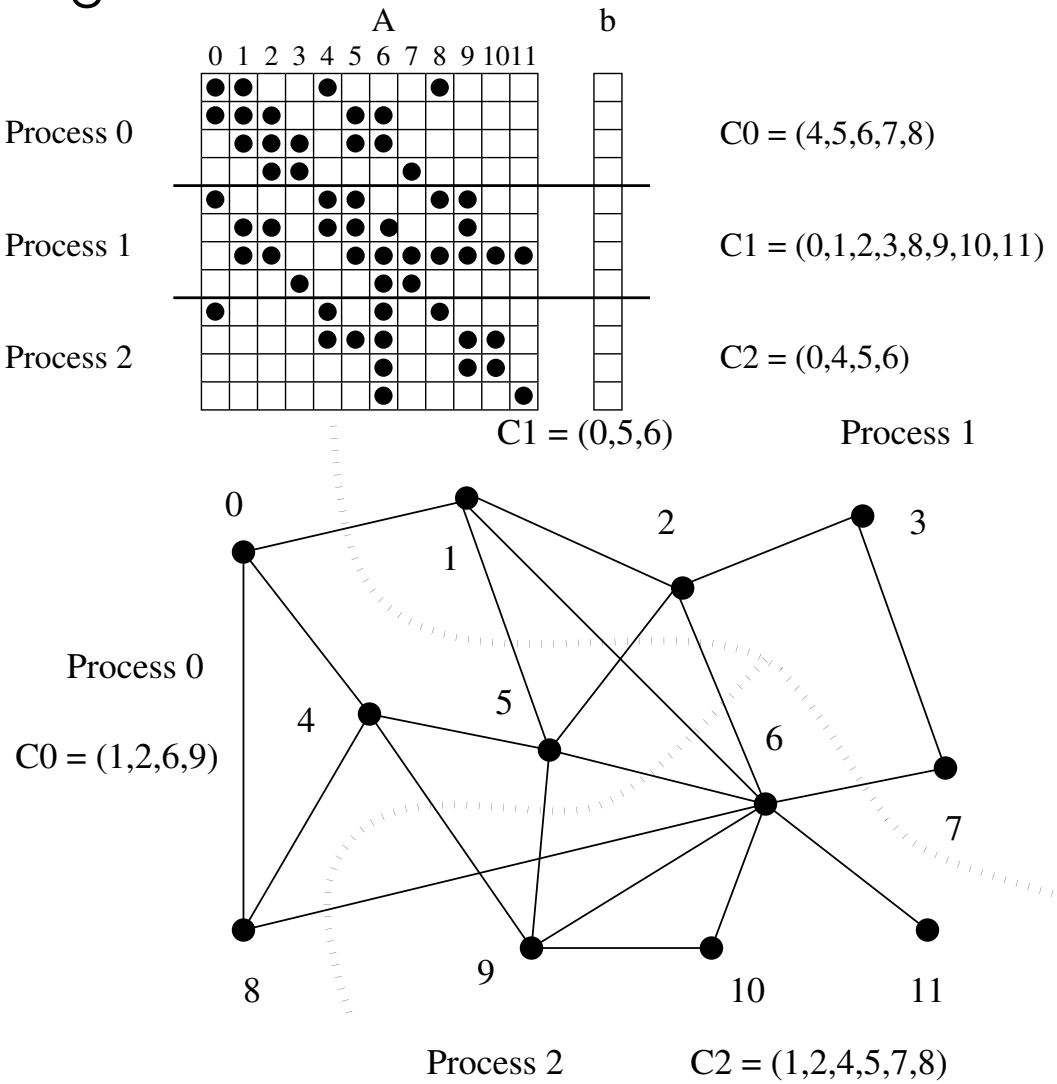
Task Paritioning: Mapping a Binary Tree Dependency Graph

Example illustrates the dependency graph of one view of quick-sort and how it can be assigned to processes in a hypercube.



Task Paritioning: Mapping a Sparse Graph

Sparse graph for computing a sparse matrix-vector product and its mapping.

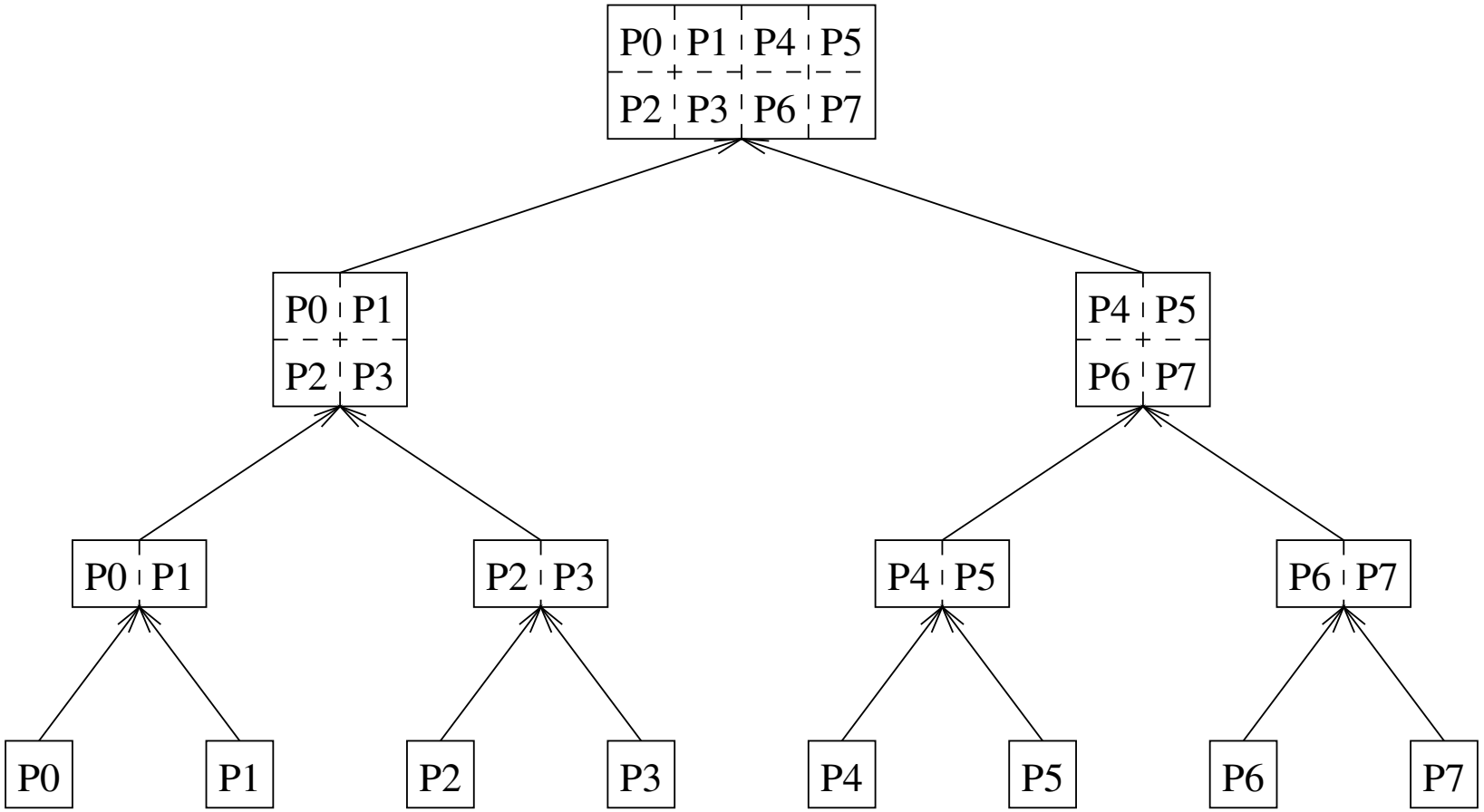


Hierarchical Mappings

- Sometimes a single mapping technique is inadequate.
- For example, the task mapping of the binary tree (quicksort) cannot use a large number of processors.
- For this reason, task mapping can be used at the top level and data partitioning within each level.

Hierarchical Mapping: Example

An example of task partitioning at top level with data partitioning at the lower level.



Schemes for Dynamic Mapping

- Dynamic mapping is sometimes also referred to as dynamic load balancing, since load balancing is the primary motivation for dynamic mapping.

- Dynamic mapping schemes can be centralized or distributed.

Centralized Dynamic Mapping

- Processes are designated as masters or slaves.
- When a process runs out of work, it requests the master for more work.
- When the number of processes increases, the master may become the bottleneck.
- To alleviate this, a process may pick up a number of tasks (a chunk) at one time. This is called Chunk scheduling.
- Selecting large chunk sizes may lead to significant load imbalances as well.
- A number of schemes have been used to gradually decrease chunk size as the computation progresses.

Distributed Dynamic Mapping

- Each process can send or receive work from other processes.
- This alleviates the bottleneck in centralized schemes.
- There are four critical questions: how are sensing and receiving processes paired together, who initiates work transfer, how much work is transferred, and when is a transfer triggered?
- Answers to these questions are generally application specific. We will look at some of these techniques later in this class.

Minimizing Interaction Overheads

- Maximize data locality: Where possible, reuse intermediate data. Restructure computation so that data can be reused in smaller time windows.

- Minimize volume of data exchange: There is a cost associated with each word that is communicated. For this reason, we must minimize the volume of data communicated.

- Minimize frequency of interactions: There is a startup cost associated with each interaction. Therefore, try to merge multiple interactions to one, where possible.

- Minimize contention and hot-spots: Use decentralized techniques, replicate data where necessary.

Minimizing Interaction Overheads (continued)

- Overlapping computations with interactions: Use non-blocking communications, multithreading, and prefetching to hide latencies.
- Replicating data or computations.
- Using group communications instead of point-to-point primitives.
- Overlap interactions with other interactions.

Parallel Algorithm Models

An algorithm model is a way of structuring a parallel algorithm by selecting a decomposition and mapping technique and applying the appropriate strategy to minimize interactions.

- Data Parallel Model: Tasks are statically (or semi-statically) mapped to processes and each task performs similar operations on different data.

- Task Graph Model: Starting from a task dependency graph, the interrelationships among the tasks are utilized to promote locality or to reduce interaction costs.

Parallel Algorithm Models (continued)

- Master-Slave Model: One or more processes generate work and allocate it to worker processes. This allocation may be static or dynamic.

- Pipeline / Producer-Comsumer Model: A stream of data is passed through a succession of processes, each of which perform some task on it.

- Hybrid Models: A hybrid model may be composed either of multiple models applied hierarchically or multiple models applied sequentially to different phases of a parallel algorithm.