CSE3009 - Parallel and Distributed Computing

Course Type: LTP Credits: 4

Prepared by
Dr Komarasamy G
Associate Professor (Grade-2)
School of Computing Science and Engineering
VIT Bhopal University

Unit-2

Parallel Algorithm and Design - Preliminaries – Decomposition Techniques – Mapping Techniques for Load balancing.

Synchronous Parallel Processing – Introduction, Example-SIMD Architecture and Programming Principles.

Decomposition Techniques

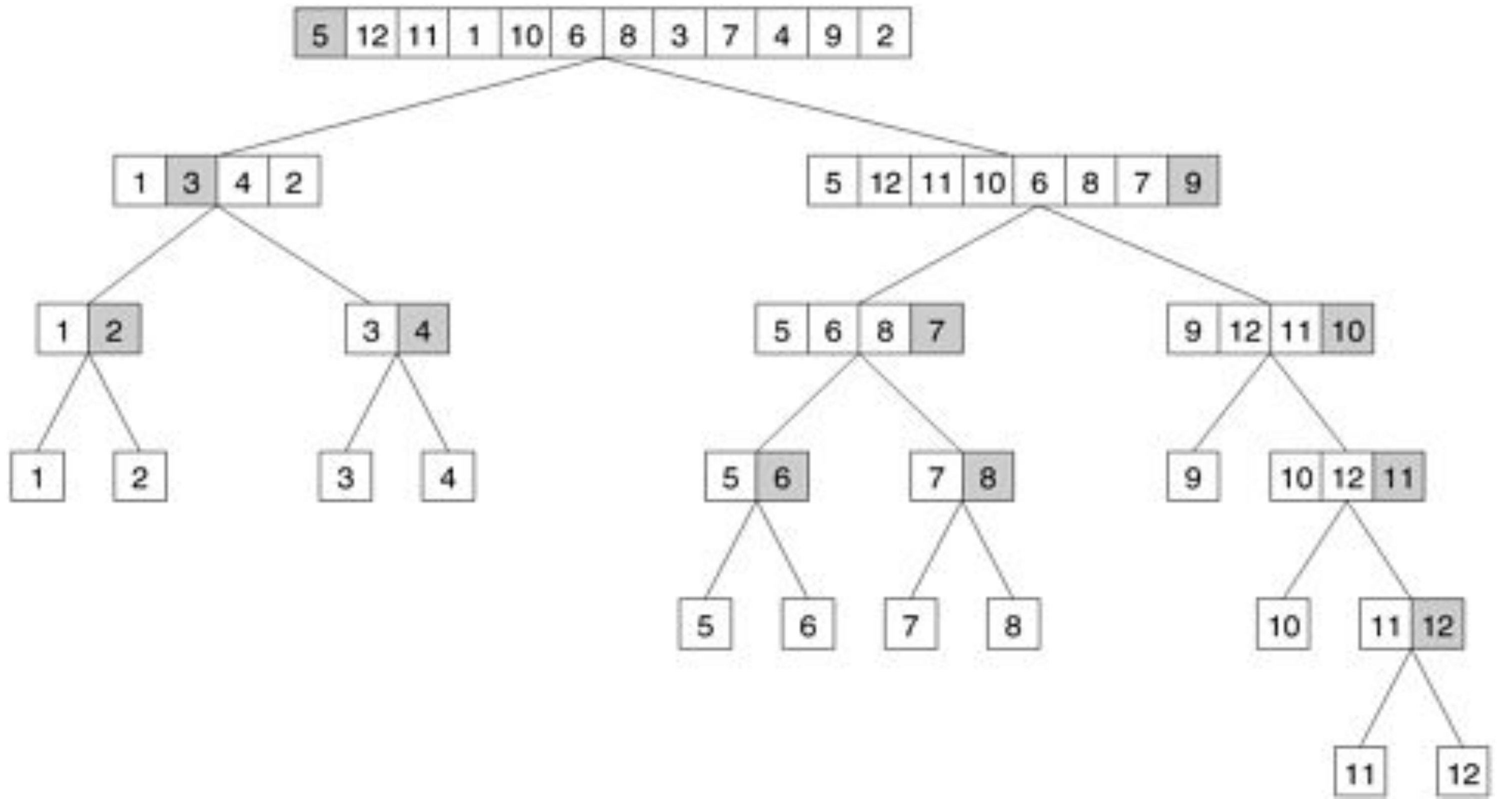
- These techniques are broadly classified as **recursive decomposition, data decomposition, exploratory decomposition, and speculative decomposition.**

1. Recursive Decomposition

- Recursive decomposition is a method for inducing concurrency in problems that can be solved using the **divide-and-conquer strategy**.
- Divide and conquer strategy results in natural concurrency, as **different sub problems can be solved concurrently.**
- **Example Quicksort** Consider the problem of sorting a sequence *A of n elements using the commonly used* quicksort algorithm.
- Quicksort is a divide and conquer algorithm that starts by selecting a and **pivot element** *x and then partitions the sequence A into two subsequences* such that all the elements in are smaller than *x and all the elements in* are greater than or equal to *x.*

Decomposition Techniques

- The partitioning step forms the *divide step of the algorithm Each one of the subsequences* is sorted by recursively calling quicksort. Each one of these recursive calls further partitions the sequences. This is illustrated in **Figure for a sequence of 12 numbers.**
- The recursion terminates when each subsequence contains only a single element. The quicksort task-dependency graph based on recursive decomposition for **sorting a sequence of 12 numbers.**



Decomposition Techniques

2. Data Decomposition

- Data decomposition is a powerful and commonly used method for deriving concurrency in algorithms that **operate on large data structures.**
- In this method, the decomposition of computations is done in two steps.

1. In the first step, the data is partitioned on which the **computations has performed.**
2. In the second step, this data partitioning is used to induce a partitioning of the **computations into tasks.**

Decomposition Techniques

3. Partitioning Output Data

- In many computations, each element of the output can be computed independently of **others as a function of the input.**
- In such computations, a partitioning of the output data automatically induces a **decomposition of the problems into tasks**, where each task is assigned the work of computing a portion of the output.

Decomposition Techniques

- **Example, Matrix multiplication**
- Consider the problem of multiplying **two n x n matrices** A and B to yield a matrix C. The four sub matrices of C, roughly of size n/2 **x** n/2 each, are then independently computed by four tasks as the sums of the appropriate products of sub matrices of A and B.
- Figure (a) Partitioning of input and output matrices into 2 x 2 sub matrices. (b) A decomposition of matrix multiplication into four tasks based on the partitioning of the matrices in (a).

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

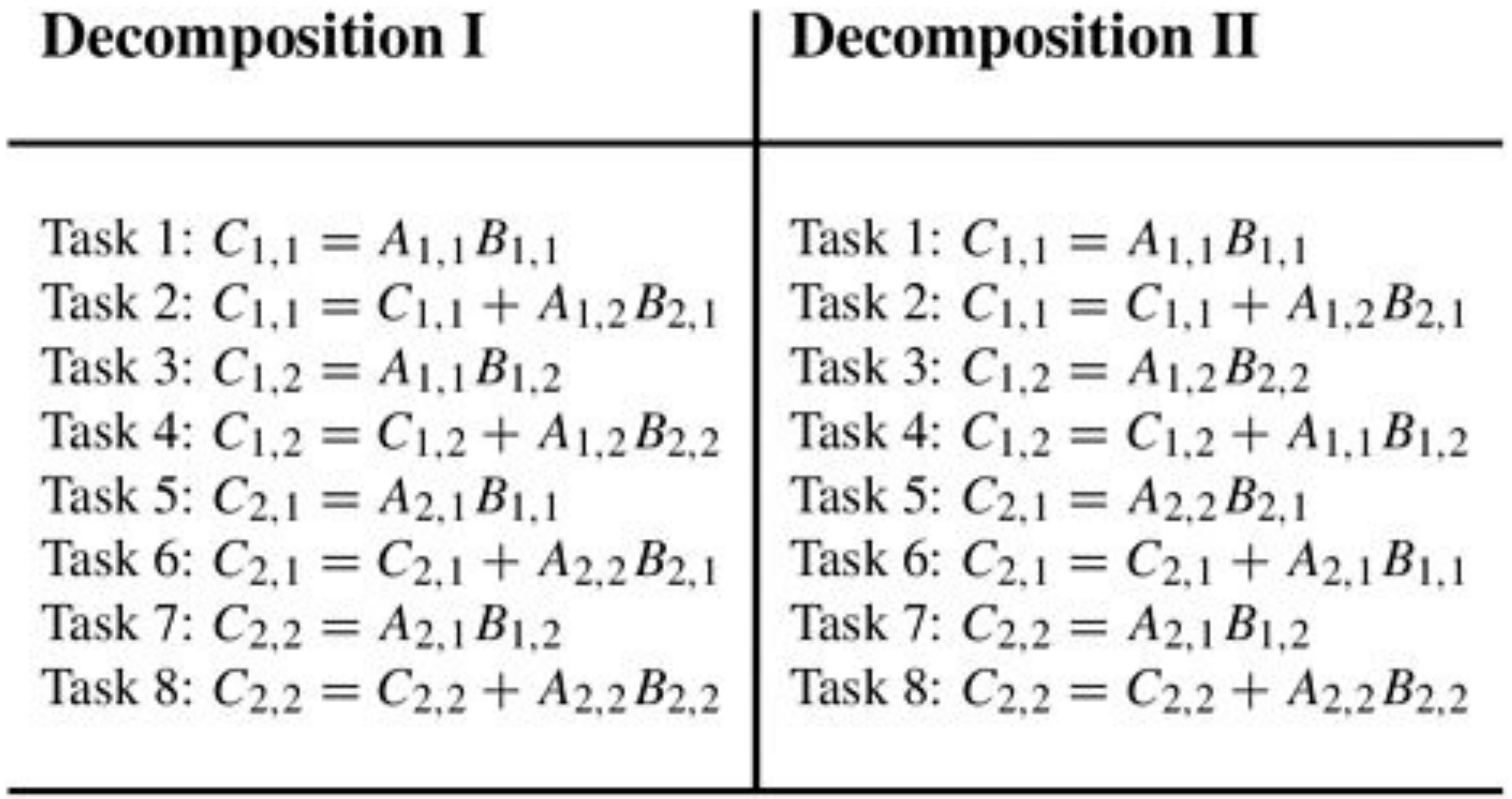
(a)

$$\begin{aligned}
\text{Task 1: } C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\
\text{Task 2: } C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\
\text{Task 3: } C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\
\text{Task 4: } C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2}
\end{aligned}$$

(b)

Decomposition Techniques

- **Two examples of decomposition of matrix multiplication into eight tasks.**



Decomposition I	Decomposition II
Task 1: $C_{1,1} = A_{1,1}B_{1,1}$	Task 1: $C_{1,1} = A_{1,1}B_{1,1}$
Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$	Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$
Task 3: $C_{1,2} = A_{1,1}B_{1,2}$	Task 3: $C_{1,2} = A_{1,2}B_{2,2}$
Task 4: $C_{1,2} = C_{1,2} + A_{1,2}B_{2,2}$	Task 4: $C_{1,2} = C_{1,2} + A_{1,1}B_{1,2}$
Task 5: $C_{2,1} = A_{2,1}B_{1,1}$	Task 5: $C_{2,1} = A_{2,2}B_{2,1}$
Task 6: $C_{2,1} = C_{2,1} + A_{2,2}B_{2,1}$	Task 6: $C_{2,1} = C_{2,1} + A_{2,1}B_{1,1}$
Task 7: $C_{2,2} = A_{2,1}B_{1,2}$	Task 7: $C_{2,2} = A_{2,1}B_{1,2}$
Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$	Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$

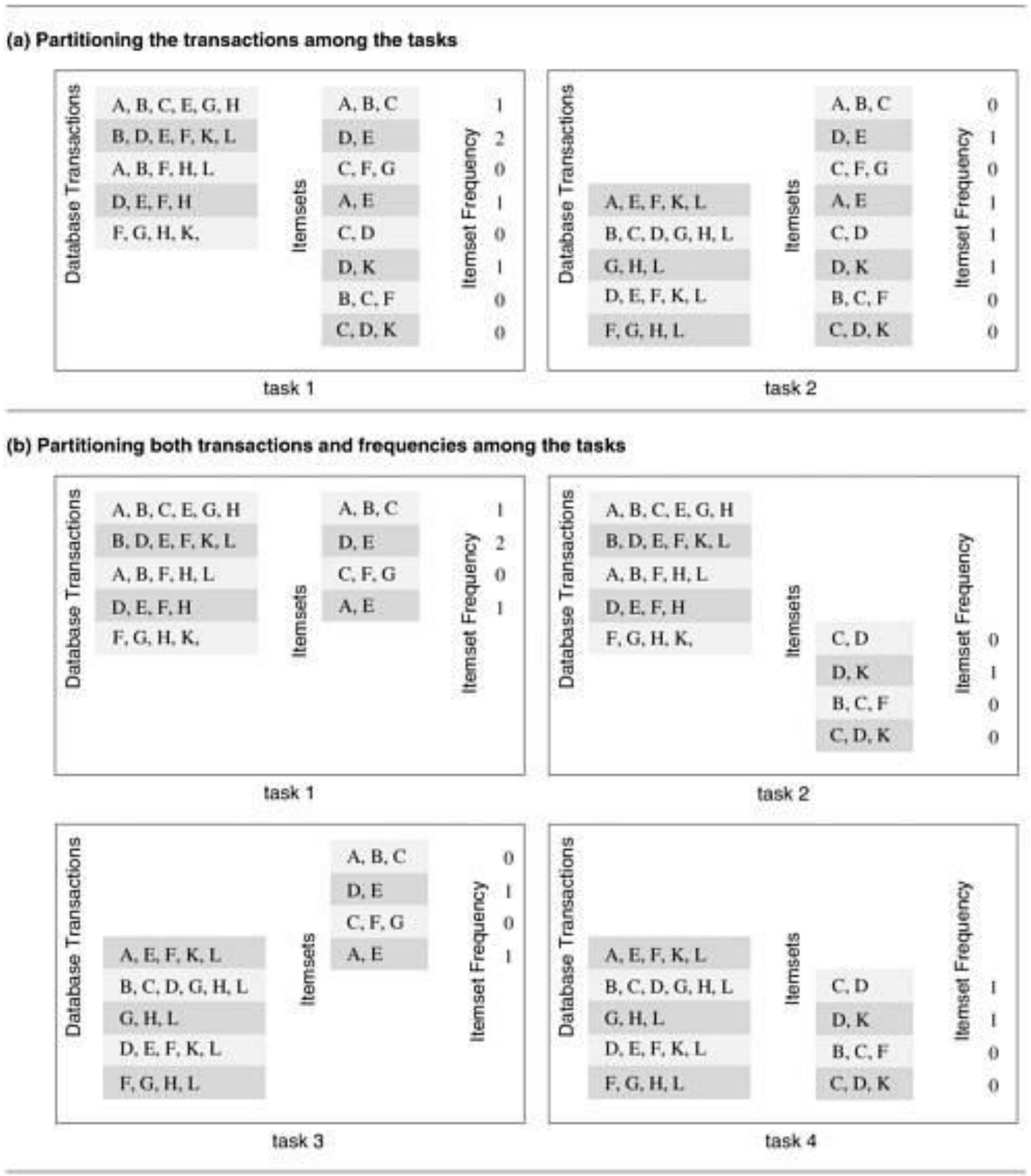
Decomposition Techniques

4. Partitioning Input Data

- Partitioning of output data can be performed only if each output can be naturally computed as a function of the input.
- **A task is created for each partition of the input data and this task performs as much computation as possible using these local data.**
- Figure, some decompositions for computing item set frequencies in a transaction database.

Decomposition Techniques

- **Partitioning Input Data**



(a) Partitioning the transactions among the tasks

task 1

Database Transactions	Itemsets	Itemset Frequency
A, B, C, E, G, H	A, B, C	1
B, D, E, F, K, L	D, E	2
A, B, F, H, L	C, F, G	0
D, E, F, H	A, E	1
F, G, H, K,	C, D	0
	D, K	1
	B, C, F	0
	C, D, K	0

task 2

Database Transactions	Itemsets	Itemset Frequency
	A, B, C	0
	D, E	1
	C, F, G	0
A, E, F, K, L	A, E	1
B, C, D, G, H, L	C, D	1
G, H, L	D, K	1
D, E, F, K, L	B, C, F	0
F, G, H, L	C, D, K	0

(b) Partitioning both transactions and frequencies among the tasks

task 1

Database Transactions	Itemsets	Itemset Frequency
A, B, C, E, G, H	A, B, C	1
B, D, E, F, K, L	D, E	2
A, B, F, H, L	C, F, G	0
D, E, F, H	A, E	1
F, G, H, K,		

task 2

Database Transactions	Itemsets	Itemset Frequency
A, B, C, E, G, H		
B, D, E, F, K, L		
A, B, F, H, L		
D, E, F, H		
F, G, H, K,	C, D	0
	D, K	1
	B, C, F	0
	C, D, K	0

task 3

Database Transactions	Itemsets	Itemset Frequency
	A, B, C	0
	D, E	1
	C, F, G	0
A, E, F, K, L	A, E	1
B, C, D, G, H, L		
G, H, L		
D, E, F, K, L		
F, G, H, L		

task 4

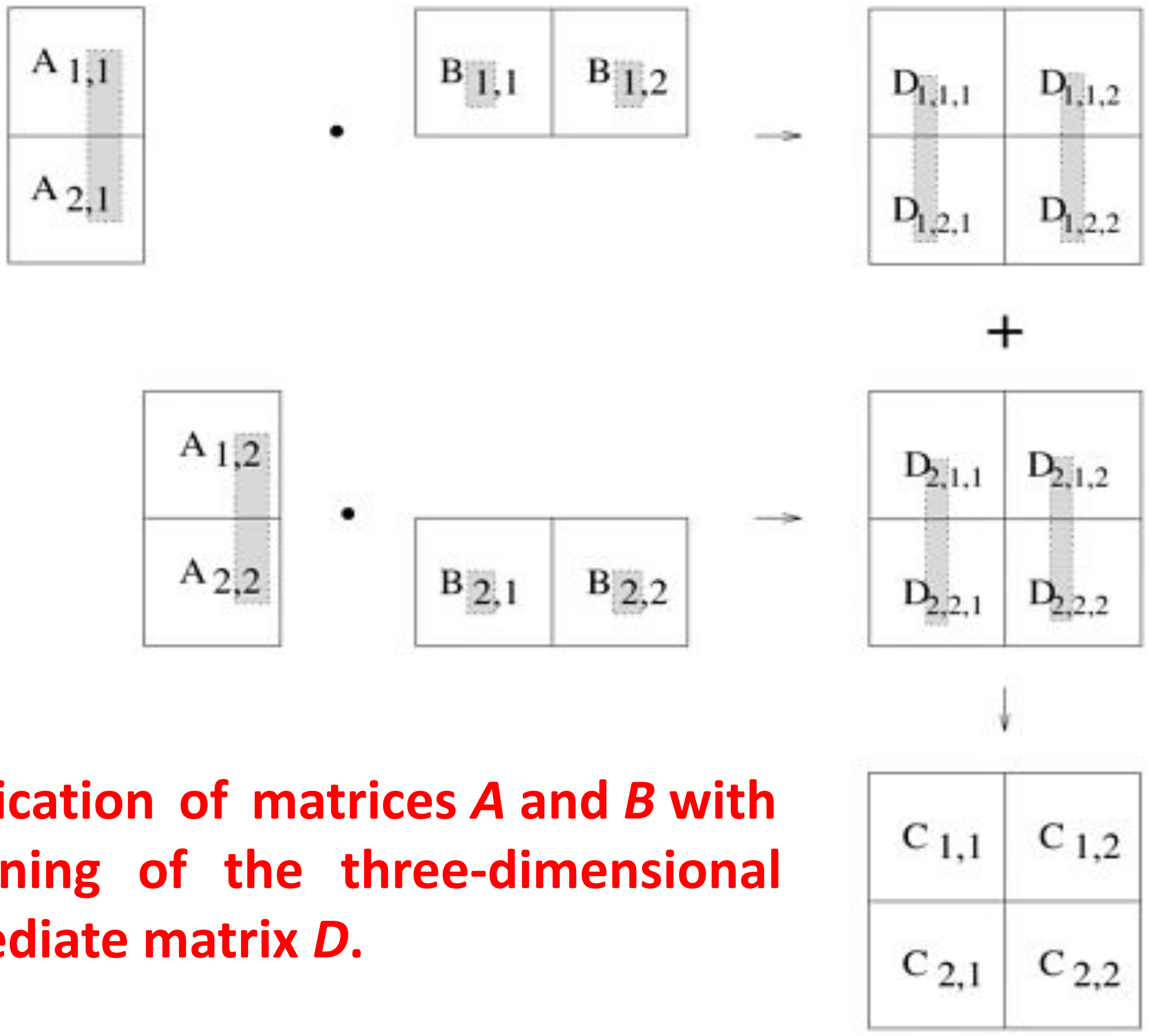
Database Transactions	Itemsets	Itemset Frequency
A, E, F, K, L		
B, C, D, G, H, L	C, D	1
G, H, L	D, K	1
D, E, F, K, L	B, C, F	0
F, G, H, L	C, D, K	0

Some decompositions for computing item set frequencies in a transaction database.

Decomposition Techniques

- Partitioning both Input and Output Data In some cases, in which it is possible to partition the output data, partitioning of input data can offer additional concurrency.
- Partitioning intermediate data Partitioning intermediate data can sometimes lead to higher concurrency than partitioning input or output data.
- Figure, **Multiplication of matrices A and B with partitioning of the three-dimensional intermediate matrix D.**

Decomposition Techniques



Multiplication of matrices *A* and *B* with partitioning of the three-dimensional intermediate matrix *D*.

Decomposition Techniques

Figure A decomposition of matrix multiplication based on partitioning the intermediate three-dimensional matrix.

Stage I

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \left(\begin{array}{c} \begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{pmatrix} \\ \begin{pmatrix} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,2} & D_{2,2,2} \end{pmatrix} \end{array} \right)$$

Stage II

$$\begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{pmatrix} + \begin{pmatrix} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,2} & D_{2,2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

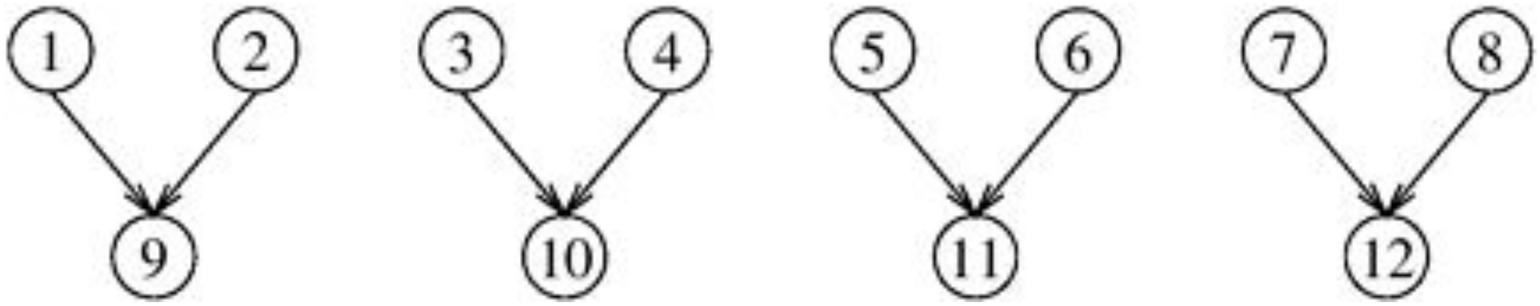
A decomposition induced by a partitioning of D

Task 01:	$D_{1,1,1} = A_{1,1} B_{1,1}$
Task 02:	$D_{2,1,1} = A_{1,2} B_{2,1}$
Task 03:	$D_{1,1,2} = A_{1,1} B_{1,2}$
Task 04:	$D_{2,1,2} = A_{1,2} B_{2,2}$
Task 05:	$D_{1,2,1} = A_{2,1} B_{1,1}$
Task 06:	$D_{2,2,1} = A_{2,2} B_{2,1}$
Task 07:	$D_{1,2,2} = A_{2,1} B_{1,2}$
Task 08:	$D_{2,2,2} = A_{2,2} B_{2,2}$
Task 09:	$C_{1,1} = D_{1,1,1} + D_{2,1,1}$
Task 10:	$C_{1,2} = D_{1,1,2} + D_{2,1,2}$
Task 11:	$C_{2,1} = D_{1,2,1} + D_{2,2,1}$
Task 12:	$C_{2,2} = D_{1,2,2} + D_{2,2,2}$

A decomposition of matrix multiplication based on partitioning the intermediate three-dimensional matrix.

Decomposition Techniques

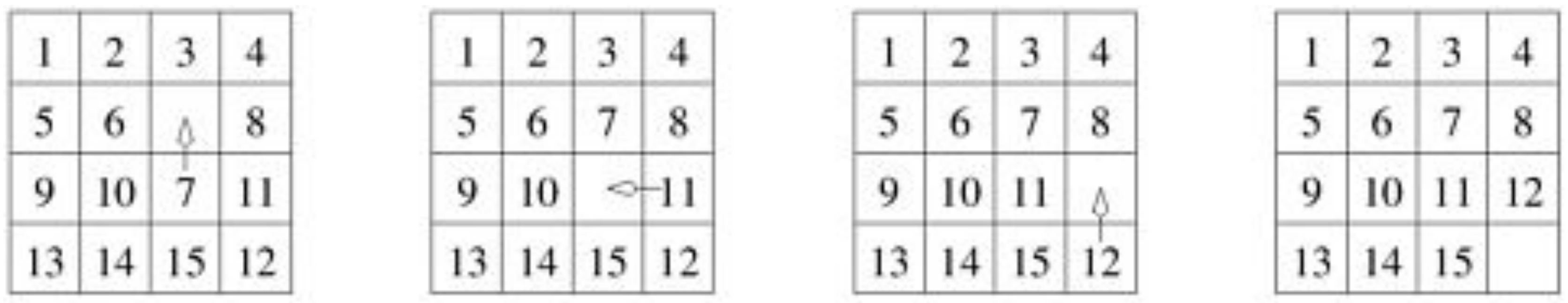
The task-dependency graph of the decomposition shown in the Figure.



Decomposition Techniques

5. Exploratory Decomposition

- Exploratory decomposition is used to decompose problems whose underlying computations correspond to a search of a space for solutions.
- **In exploratory decomposition, the search space is partitioned into smaller parts, and search each one of these parts concurrently, until the desired solutions are found.**
- Figure, **15-puzzle problem** instance showing the initial configuration (a), the final configuration (d), and a sequence of moves leading from the initial to the final configuration.



(a) (b) (c) (d)

Decomposition Techniques

6. Speculative Decomposition

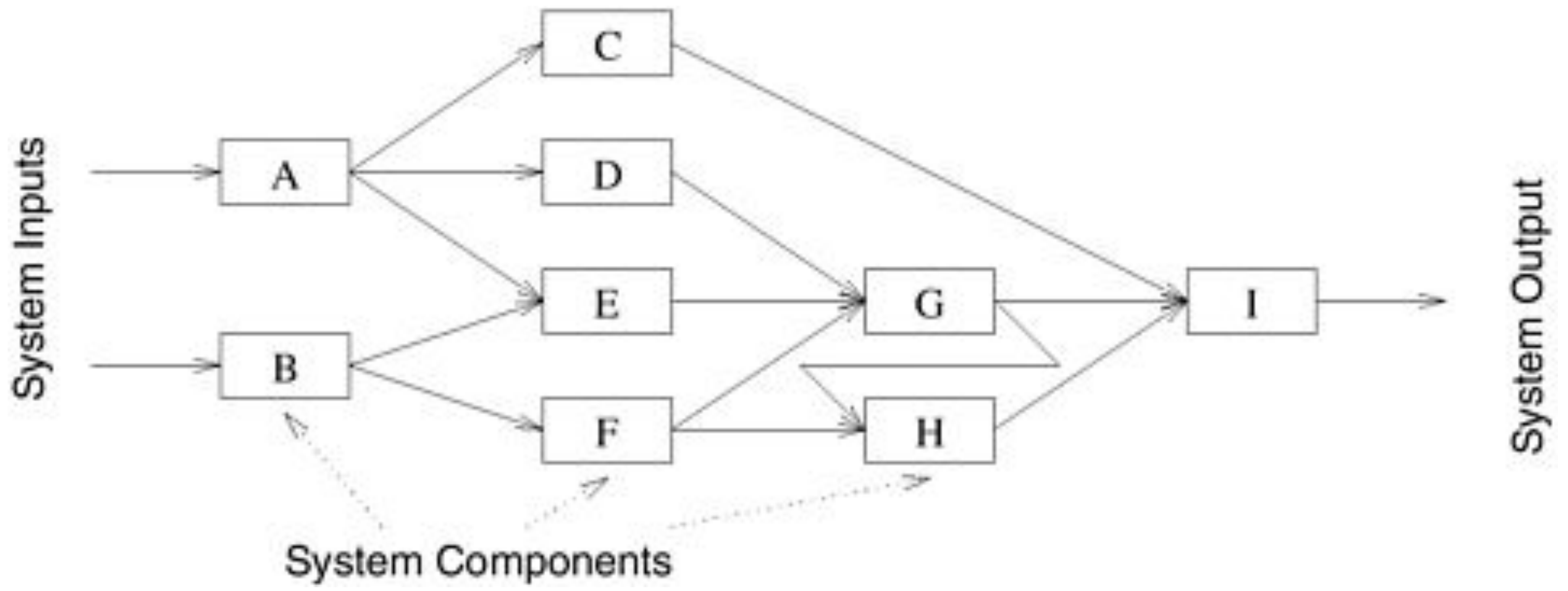
- Speculative decomposition is used when a **program may take one of many possible computationally significant branches depending on the output of other computations that precede it.**
- In this situation, while one task is performing the computation whose output is **used in deciding the next computation**, other tasks can concurrently start the computations of the next stage.
- The speedup due to speculative decomposition can add up if **there are multiple speculative stages.**
- An example of an application in which speculative decomposition is useful is discrete event simulation.

Decomposition Techniques

- Example **Parallel discrete event simulation**
- Consider the simulation of a system that is represented as a **network or a directed graph.**
- The nodes of this network represent components. Each component has an input buffer of jobs. The initial state of each component or node is idle.
- An idle component picks up a job from its input queue, if there is one, processes that job in some finite amount of time, and puts it in the input buffer of the components which are connected to it by outgoing edges.
- A component has to wait if the input buffer of one of its outgoing neighbors if full, until that neighbor picks up a job to create space in the buffer. There is a finite number of input job types.

Decomposition Techniques

- Example **Parallel discrete event simulation**

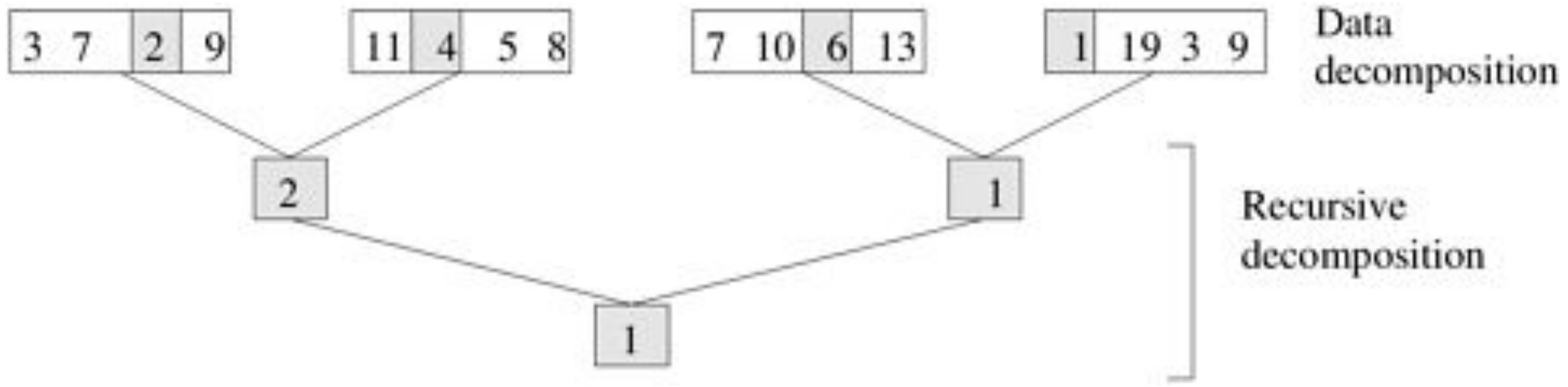


A simple network for discrete event simulation.

Decomposition Techniques

7. Hybrid Decompositions

- A computation is structured into **multiple stages** and it is sometimes necessary to apply different types of decomposition in different stages.
- Figure, Hybrid decomposition for finding the minimum of an array of size 16 using four tasks.



Hybrid decomposition for finding the minimum of an array of size 16 using four tasks.

More details: http://users.atw.hu/parallelcomp/ch03lev1sec2.html