Parallel and Distributed Computing

CS3006

Week 4, Lecture 2

Decomposition Techniques

February 14 2024

Original slides by: Dr. Rana Asif Rehman
Additions for Spring 2024 by: Dr. Abdul Qadeer

Agenda

- A Quick Review
- **Decomposition Techniques**
  - Recursive
  - Data-decomposition
  - Exploratory
  - Speculative
  - Hybrid

Quick Review to the Previous Lecture

- Parallel Algorithm Design Life Cycle
- Tasks, Decomposition, and Task-dependency graphs
- Granularity
  - Fine-grained
  - Coarse-grained
- Concurrency
  - Max-degree of concurrency
  - Critical path length
  - Average-degree of concurrency

Decomposition Techniques

- The process of decomposing larger problems into smaller tasks for concurrent executions, is known to as decomposition.
- The techniques that facilitate this decomposition are known to as decomposition techniques.
- **Common techniques:**
  - Recursive
  - Data-decomposition
  - Exploratory decomposition
  - Speculative decomposition
  - Hybrid
- Recursive and data decompositions are relatively general purpose
- Exploratory and speculative are special purpose in nature

Decomposition Techniques

1. Recursive Task Decomposition

- Recursive decomposition is a method for inducing concurrency in the problems that can be solved using divide and conquer strategy
- Divides each problem into a set of independent sub-problems
- Each one of these subproblems is solved by recursively applying a similar division into smaller subproblems followed by a combination of their results
- A natural concurrency exists as different subproblems can be solved concurrently.

Decomposition Techniques

Recursive Decomposition (Example: Quick sort)

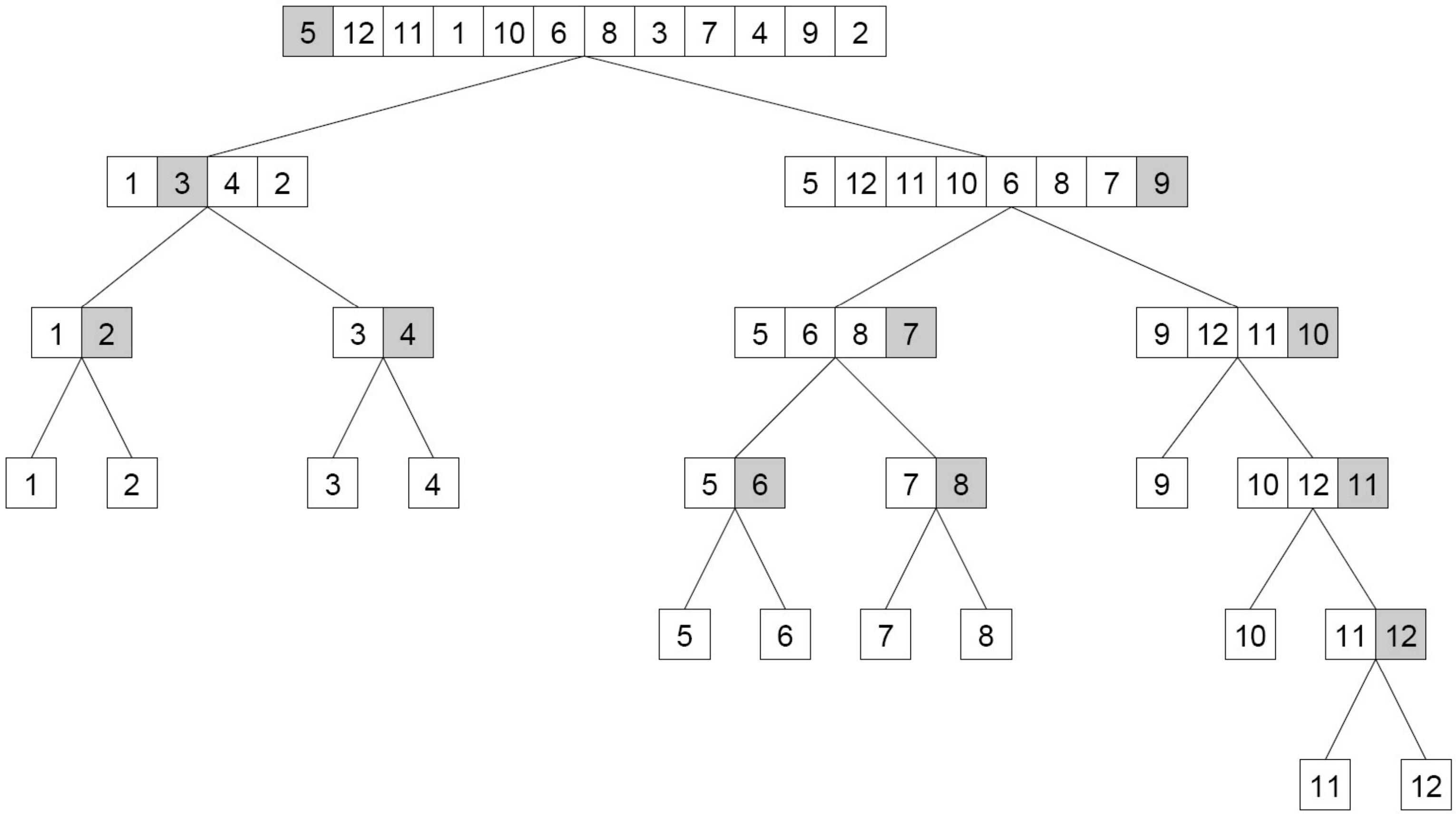


Figure 3.8 The quicksort task-dependency graph based on recursive decomposition for sorting a sequence of 12 numbers.

Decomposition Techniques

Recursive Decomposition (Modifying simple problem to support recursive decomposition)

```
procedure SERIAL_MIN (A, n)
begin
min = A[0];
for i := 1 to n − 1 do
    if (A[i] < min) min := A[i];
endfor;
return min;
end SERIAL_MIN
```

Algorithm 3.1 A serial program for finding the minimum in an array of numbers A of length n.

```
procedure RECURSIVE_MIN (A, n)
begin
if (n = 1) then
    min := A[0];
else
    lmin := RECURSIVE_MIN (A, n/2);
    rmin := RECURSIVE_MIN (&(A[n/2]), n − n/2);
    if (lmin < rmin) then
        min := lmin;
    else
        min := rmin;
    endelse;
endelse;
return min;
end RECURSIVE_MIN
```

Algorithm 3.2 A recursive program for finding the minimum in an array of numbers A of length n.

Decomposition Techniques

Recursive Decomposition (Modifying simple problem to support recursive decomposition**)**

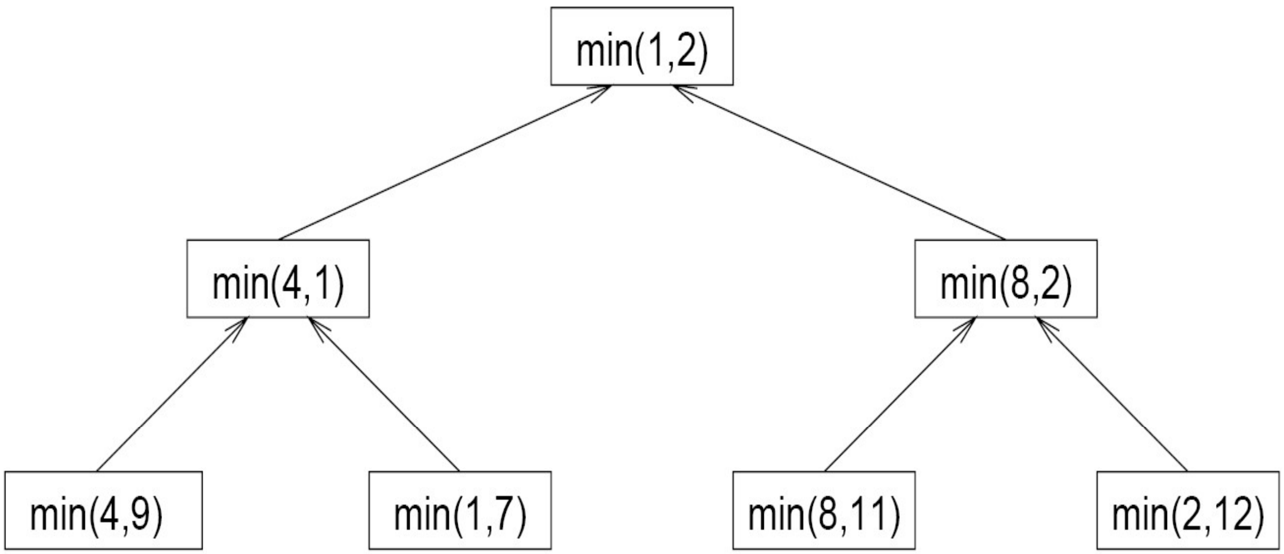


Figure 3.9 The task-dependency graph for finding the minimum number in the sequence {4, 9, 1, 7, 8, 11, 2, 12}. Each node in the tree represents the task of finding the minimum of a pair of numbers.

Decomposition Techniques

2. Data Decomposition

- Powerful and commonly used method
- Two step procedure:
  1. Partition data on which computation is to be performed
  2. This data partitioning is used to induce a partitioning of the computations into tasks.
- **Partitioning output data**
  - Used where each element of the output can be computed independently of others as a function of the input.
  - Partitioning of the output data automatically induces a decomposition of the problems into tasks
  - each task is assigned the work of computing a portion of the output

Decomposition Techniques

Data Decomposition (Partitioning Output Data)

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

(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)

Figure 3.10 (a) Partitioning of input and output matrices into 2×2 submatrices. (b) A decomposition of matrix multiplication into four tasks based on the partitioning of the matrices in (a).

Decomposition Techniques

Data Decomposition (Partitioning Output Data)

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: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$

Figure 3.11 Two examples of decomposition of matrix multiplication into eight tasks.

Decomposition Techniques

(Partitioning Output Data)

(a) Transactions (input), itemsets (input), and frequencies (output)

Database Transactions
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,
A, E, F, K, L
B, C, D, G, H, L
G, H, L
D, E, F, K, L
F, G, H, L

Itemsets	Itemset Frequency
A, B, C	1
D, E	3
C, F, G	0
A, E	2
C, D	1
D, K	2
B, C, F	0
C, D, K	0

(b) Partitioning the frequencies (and itemsets) among the tasks

task 1

Database Transactions
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,
A, E, F, K, L
B, C, D, G, H, L
G, H, L
D, E, F, K, L
F, G, H, L

Itemsets	Itemset Frequency
A, B, C	1
D, E	3
C, F, G	0
A, E	2

task 2

Database Transactions
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,
A, E, F, K, L
B, C, D, G, H, L
G, H, L
D, E, F, K, L
F, G, H, L

Itemsets	Itemset Frequency
C, D	1
D, K	2
B, C, F	0
C, D, K	0

Figure 3.12 Computing itemset frequencies in a transaction database.

Decomposition Techniques

Data Decomposition

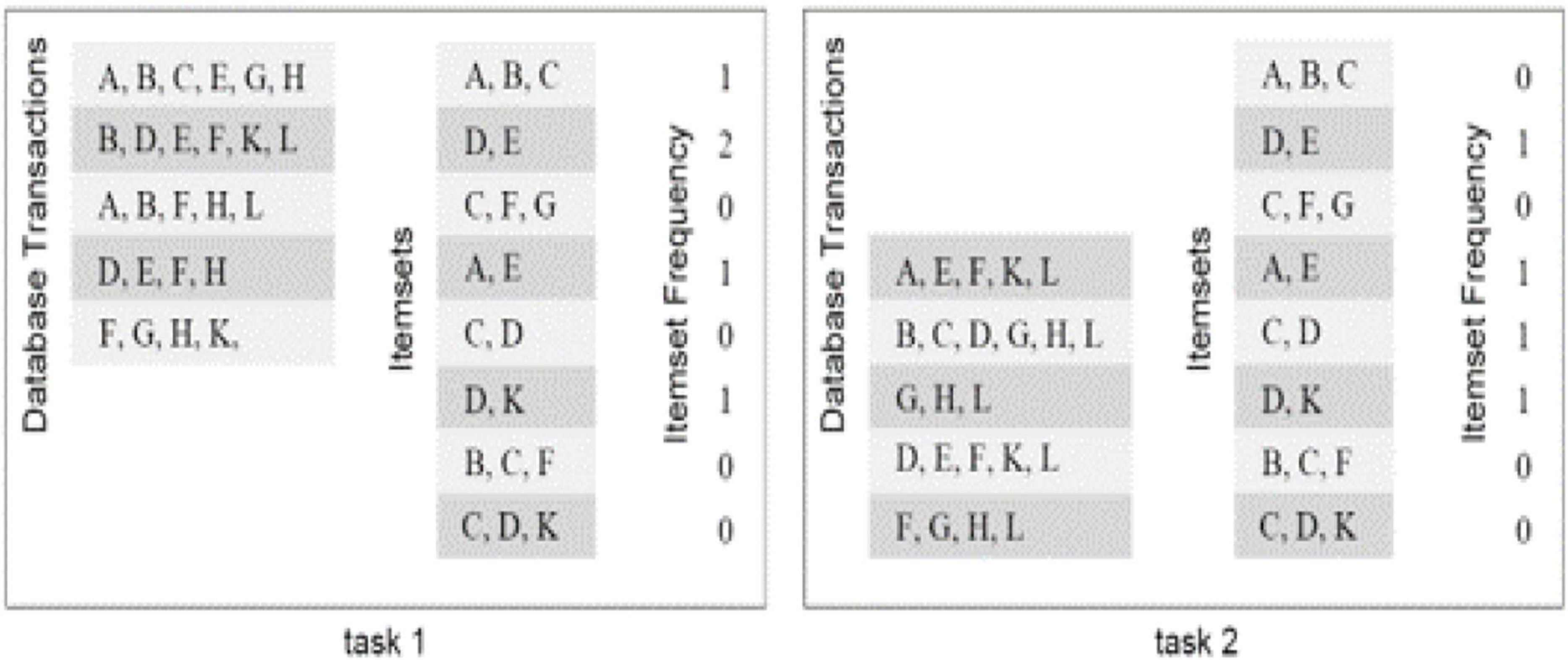
Partitioning input data

- In many algorithms, it is not possible or desirable to partition the output data.
  - The output may be a single unknown value. Such as in case of finding sum, minimum, maximum or frequencies of a number.
- It is sometimes possible to partition the input data, and then use this partitioning to induce concurrency
- A task is created for each partition of the input data and this task performs as much computation as possible using these local data
- Then local solutions are combined to generate a global solution

Decomposition Techniques

Partitioning input data

- (a) Partitioning the transactions among the tasks



Decomposition Techniques

Data Decomposition

Partitioning both input and output data

- Consider the problems where output data-partitioning is possible
- Here, partitioning the input also, can offer additional concurrency
- The next example shows 4-way decomposition of the previous example based on both input-output partitioning.

Decomposition Techniques

Partitioning both input and output data

(b) Partitioning both transactions and frequencies among the tasks

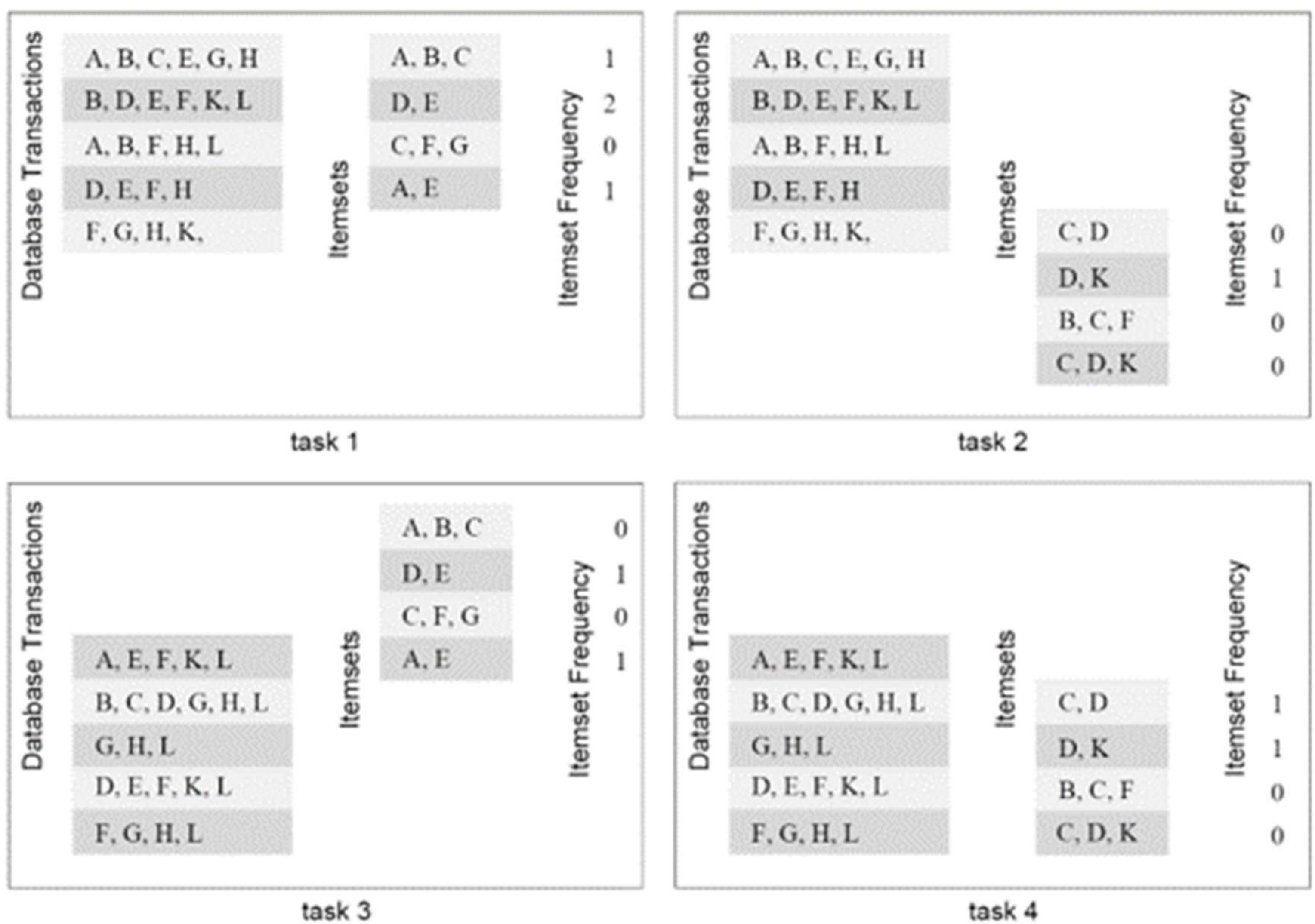


Figure 3.13 Some decompositions for computing itemset frequencies in a transaction database.

Decomposition Techniques

Partitioning both intermediate data

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: $C_{2,2} = D_{1,2,2} + D_{2,2,2}$

Figure 3.15 A decomposition of matrix multiplication based on partitioning the intermediate three-dimensional matrix.

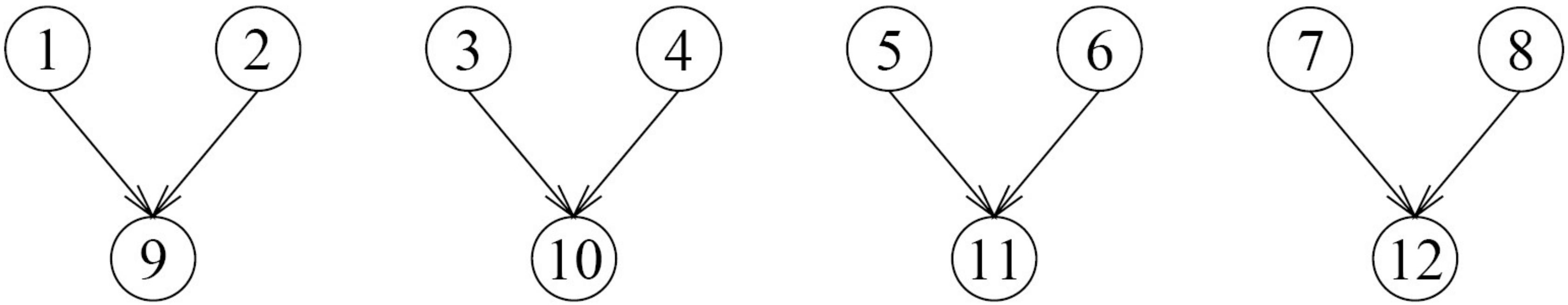


Figure 3.16 The task-dependency graph of the decomposition shown in Figure 3.15.

Estimating Number of Operations

- O(n^3/8) operations for multiplying two n/2 X n/2 matrices
  - Above is by each of the 8 tasks
- O(n^2/4) for addition of two n/2 X n/2 matrices
  - Above is by each of the 4 tasks
- Note we traded more space to hold intermediate matrices to achieve better concurrency
  - A typical space-speed tradeoff!

Decomposition Techniques

Owner Compute Rule

- Task decomposition based on data-partitioning is widely known as owner compute rule.
- Two types of partitioning hence, two definitions:

1. If we assign partitions of the input data to tasks:
   - The rule means that a task performs all the computations that can be done using these data
2. If we assign partition of output data to the tasks:
   - The rule means that a task computes all the data in the partition assigned to it (portion of the output).

Decomposition Techniques

3. Exploratory Decomposition

- Specially used to decompose the problems having underlying computation like search-space exploration.
- Steps:
  1. Partition the search space into smaller parts
  2. Search each one of these parts concurrently, until the desired solutions are found.

Decomposition Techniques

3. Exploratory Decomposition

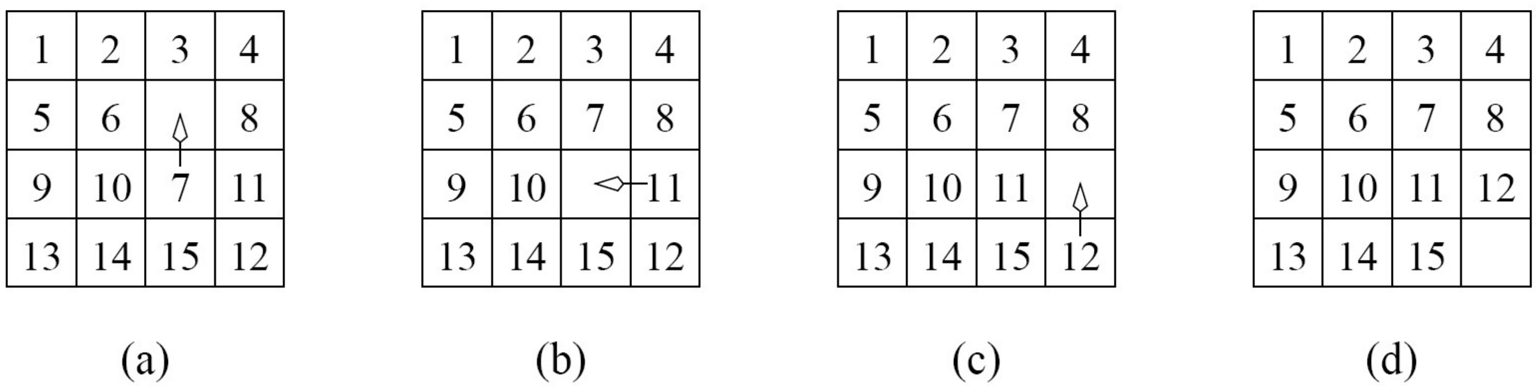


Figure 3.17 A 15-puzzle problem instance showing the initial configuration (a), the final configuration (d), and a sequence of moves leading from the initial to the final configuration.

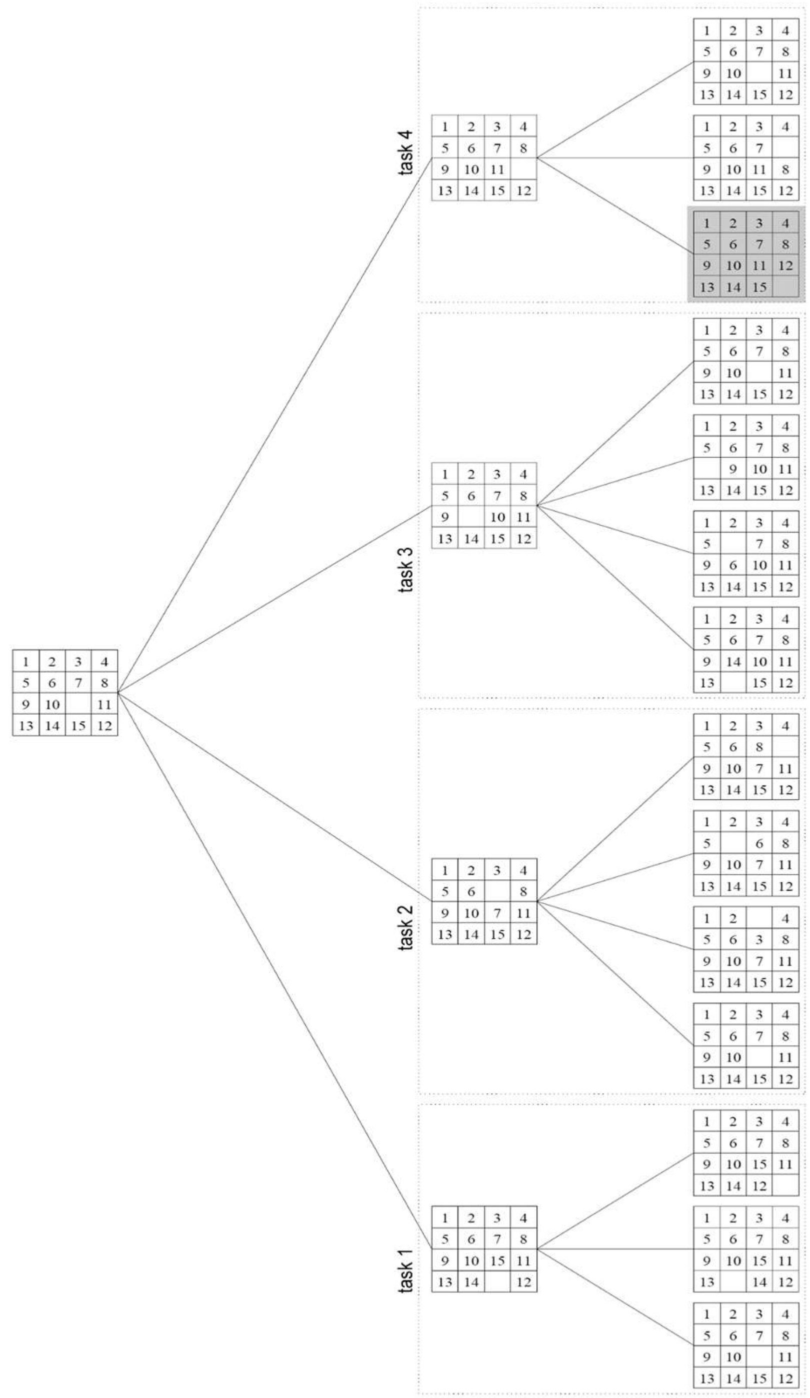


Figure 3.18 The states generated by an instance of the 15-puzzle problem.

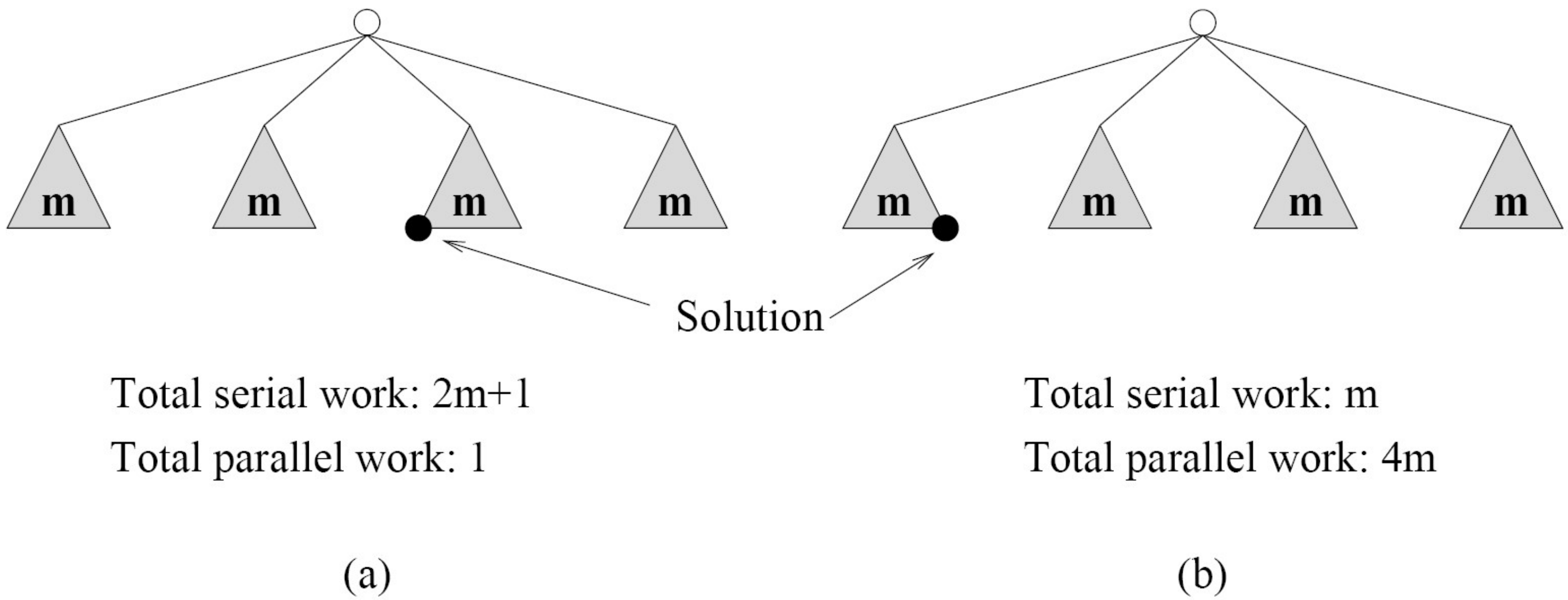


Figure 3.19 An illustration of anomalous speedups resulting from exploratory decomposition.

Decomposition Techniques

4. Speculative Decomposition

- Usually used in the problems where different input values or output of previous stage causes many computationally intensive branches.
- Speculation is something like Gamble or Risk or preliminary guess.
- Steps:
  - Speculate(guess) the output of previous stage
  - Start performing computations in the next stage before even the completion of the previous stage.
  - After availability of the output of previous stage, if speculation was correct than most of the computation for next step would have already been done.

Decomposition Techniques

4. Speculative Decomposition

- Switch (the programming construct) Example Algorithm:

1: Calculate expression for the switch condition → task 0

2: Case 0: Multiply vector **b** with matrix **A** → task 1

3: Case 1: Multiply vector **c** with matrix **A** → task 2

4: Case 2: Multiply vector **d** with matrix **A** → task 3

5: display result → task 4

Decomposition Techniques

4. Speculative Decomposition

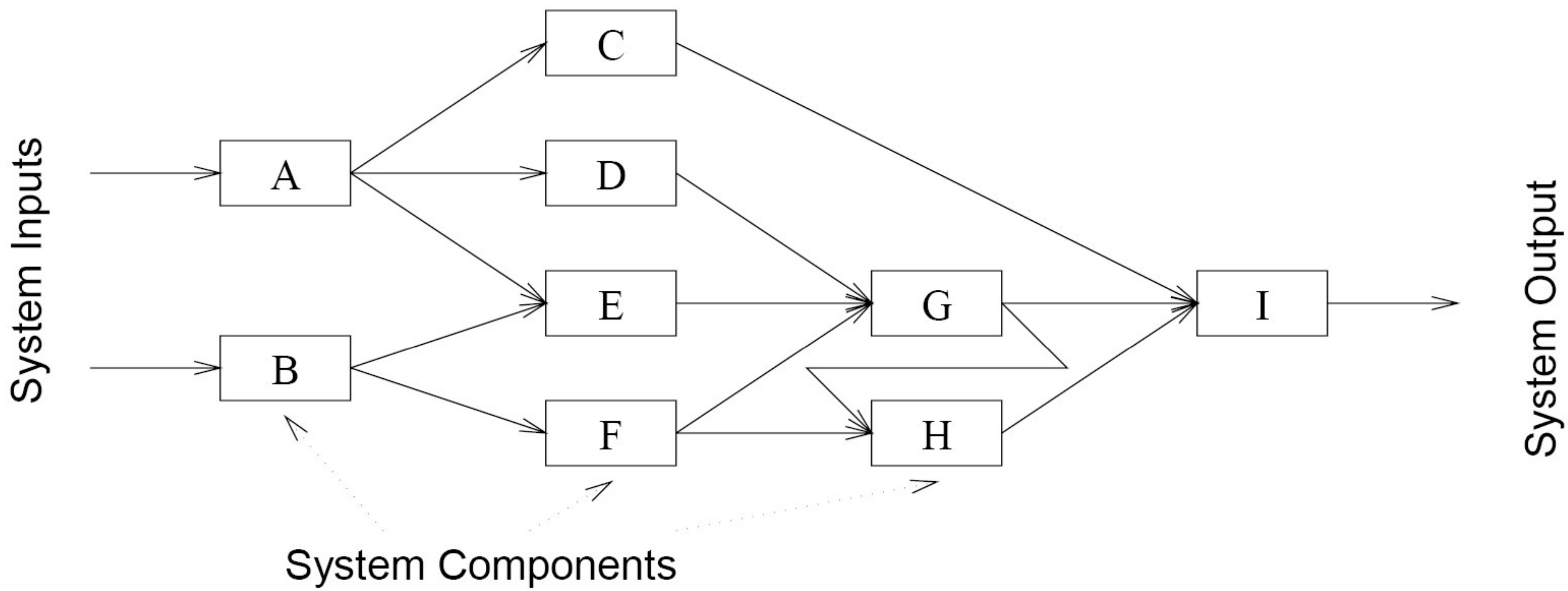


Figure 3.20 A simple network for discrete event simulation.

Decomposition Techniques

5. Hybrid Decomposition

- Decomposition technique are not exclusive
  - We often need to combine them together

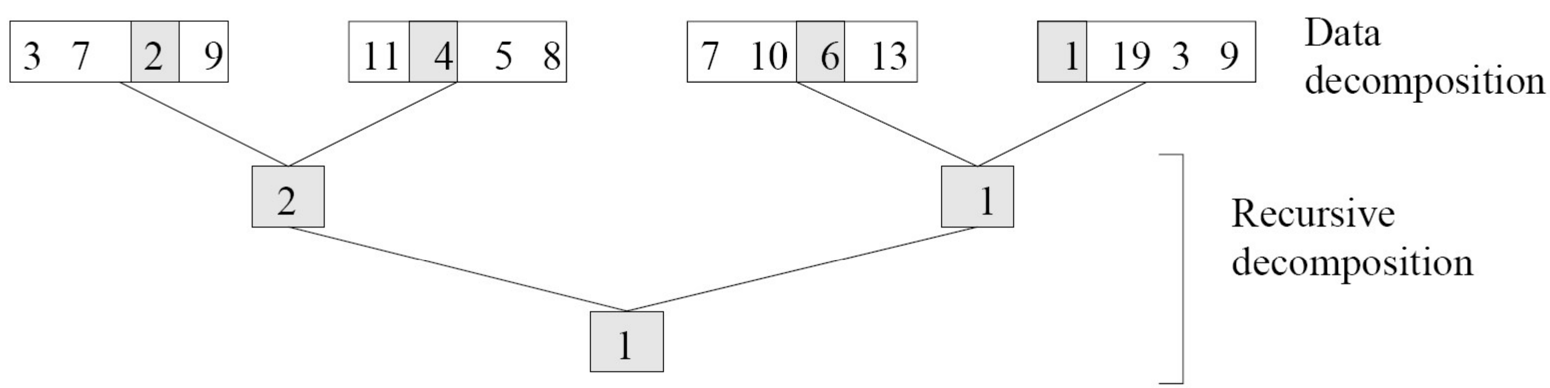


Figure 3.21 Hybrid decomposition for finding the minimum of an array of size 16 using four tasks.

Parallel Execution Style: Fork-Exec Idiom

Fork-Exec Idiom

- A versatile way to make multiple executing elements
- Usually address space is not shared among multiple processes (but more on that later!)
- Let's discuss it at a blog that I wrote a while ago
  - https://www.educative.io/blog/fork-exec-linux
  - We can execute example code there as well
- Few kinds of processing can be done using this model
  - But using a variant of fork (called clone) we can make threads sharing the address space
  - Suitable for shared-memory model to execute tasks

Questions

References

1. Kumar, V., Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to parallel computing* (Vol. 110). Redwood City, CA: Benjamin/Cummings.
2. Quinn, M. J. Parallel Programming in C with MPI and OpenMP,(2003).