Soni

Decomposition Techniques

General Purpose → Recursive
General Purpose → Data

Special Purpose → Exploratory
Special Purpose → Speculative

I. Recursive Decomposition

→ Natural application to Divide-and-Conquer

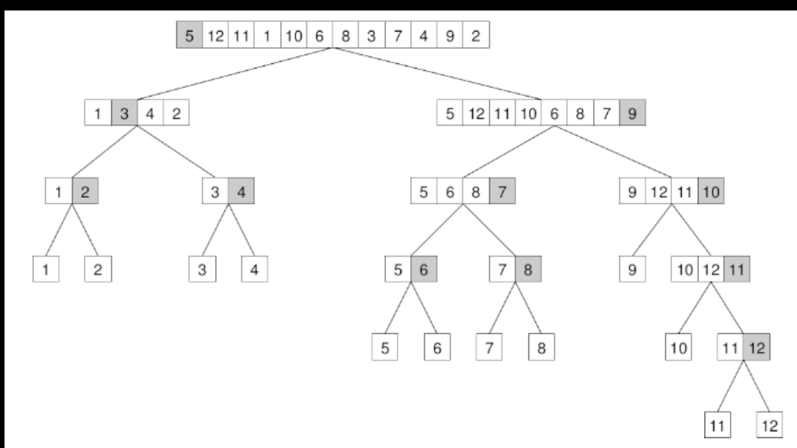
→ Problem decomposed into a set of sub-problems

→ Recursively further divide until desired granularity is reached

→ Solve the base case problem

→ Apply the Conquer technique to Combine the Solutions for the larger case, recursively.

a. Quick Sort



→ After partitioning the first array, there are two partitions

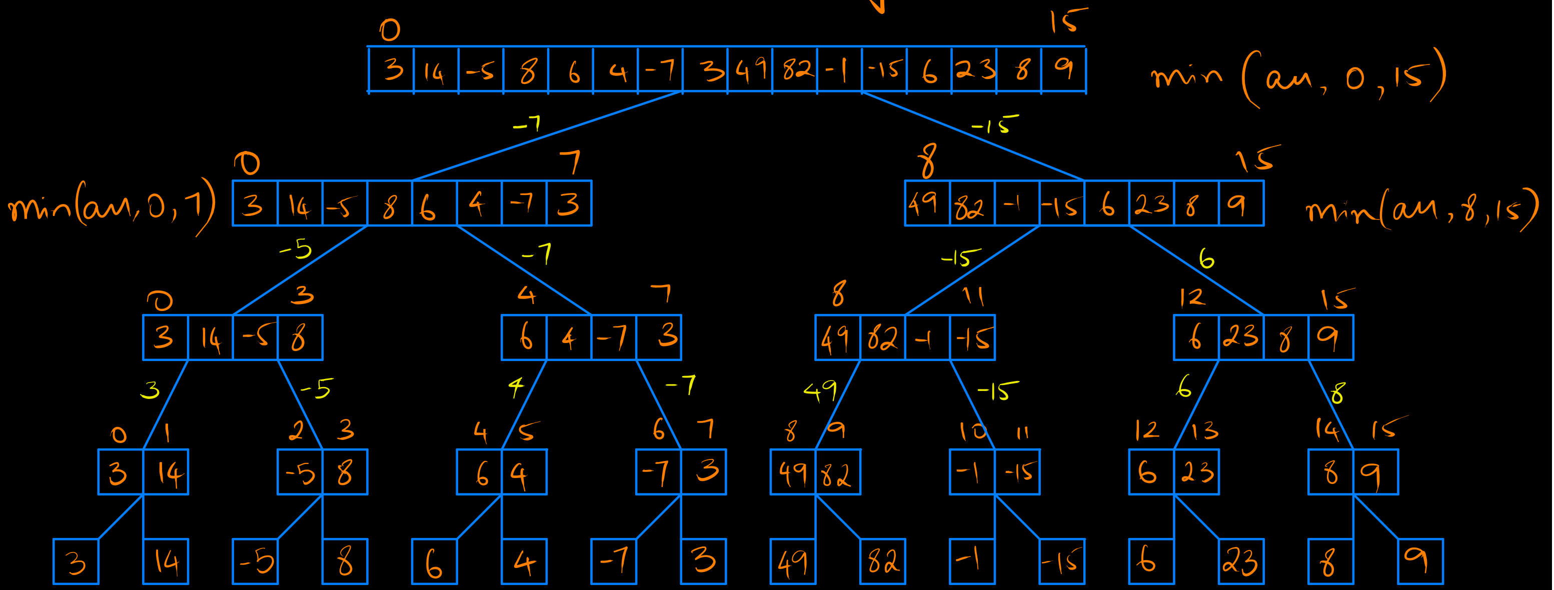
→ Each of these can be partitioned concurrently.

→ Maximum Degree of Concurrency : 4

b. Minimum Element

```
min = arr[0];
for(i = 1; i < N; i++)
    if (arr[i] < min)
        arr[i] = min;
```

```
int min(int arr[], int
            start, int end)
{
    if(start == end)
        return(arr[start]);
    int mid = (start + end)/2;
    int m1 = min(arr, start, mid);
    int m2 = min(mid+1, end);
    if (m1 < m2)
        return(m1);
    return(m2);
}
```



→ Minimum of two numbers can be found concurrently

→ Maximum Degree of Concurrency : 8

II. Data Decomposition

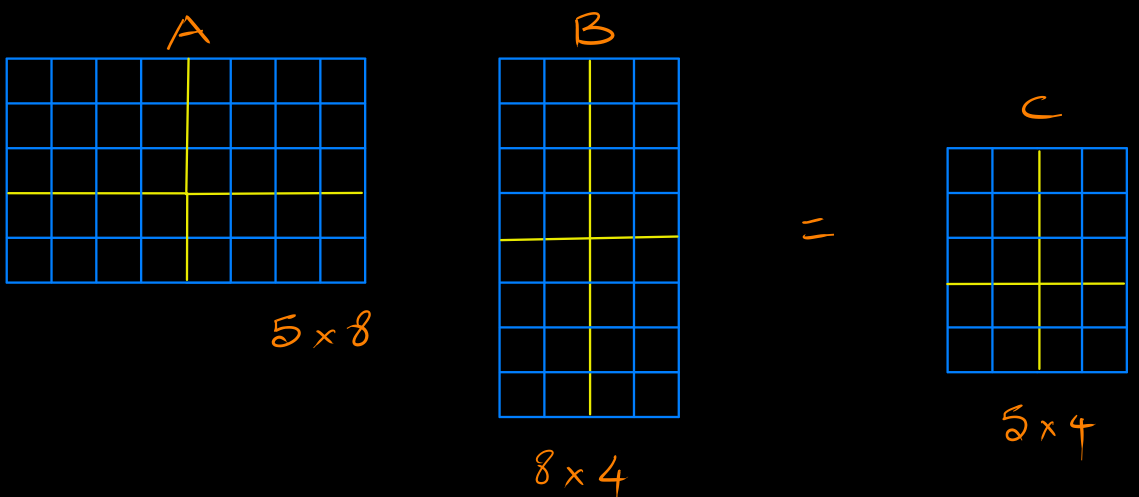
Data on which Computations are performed is Partitioned

Partitioned data induces partitioning of Computation into Tasks

Output Data / Input Data / Both Input & Output Data / Intermediate Data

a. Partitioning Output Data for Matrix Multiplication

Each element of output can be computed independently



$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

$C_{1,1} = A_{1,1} B_{1,1} + A_{1,2} B_{2,1} \rightarrow$ Task 1

$C_{1,2} = A_{1,1} B_{1,2} + A_{1,2} B_{2,2} \rightarrow$ Task 2

$C_{2,1} = A_{2,1} B_{1,1} + A_{2,2} B_{2,1} \rightarrow$ Task 3

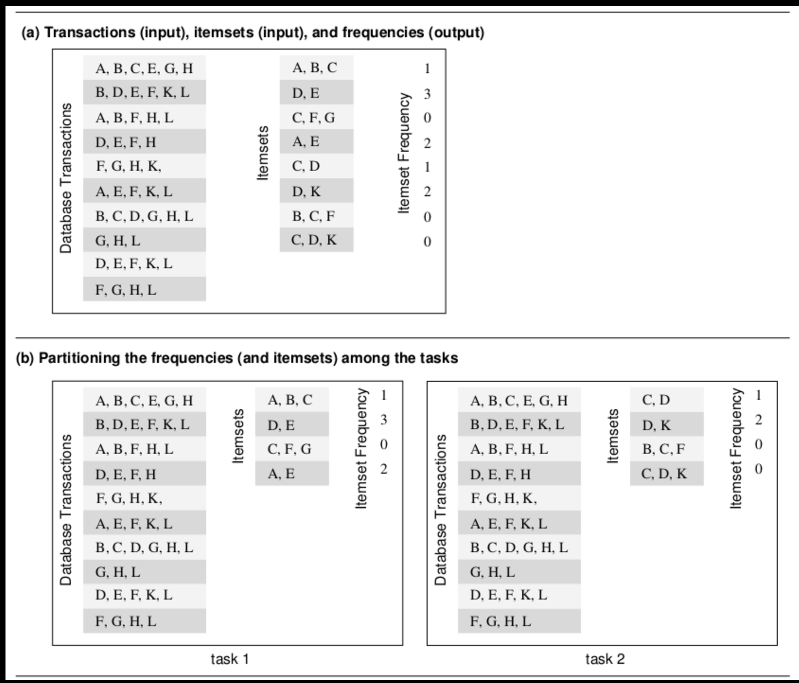
$C_{2,2} = A_{2,1} B_{1,2} + A_{2,2} B_{2,2} \rightarrow$ Task 4

$\Leftarrow$ Max Degree of Concurrency? 4

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

$\boxed{4}$

b. Partitioning Output Data for Computing Itemset frequency



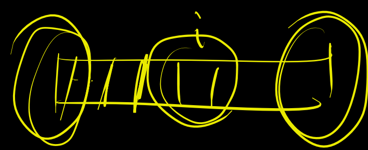
(inducing Partitioning of input data)

Possible only when each output can be computed as a function of the input.

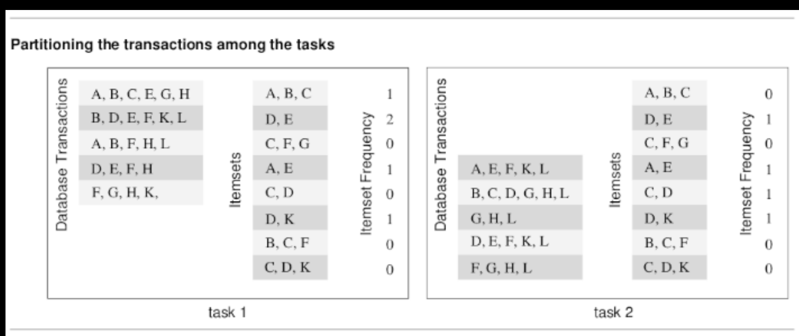
Minimum, Maximum, Sum of N numbers?

Sorting N numbers?

i^{th}

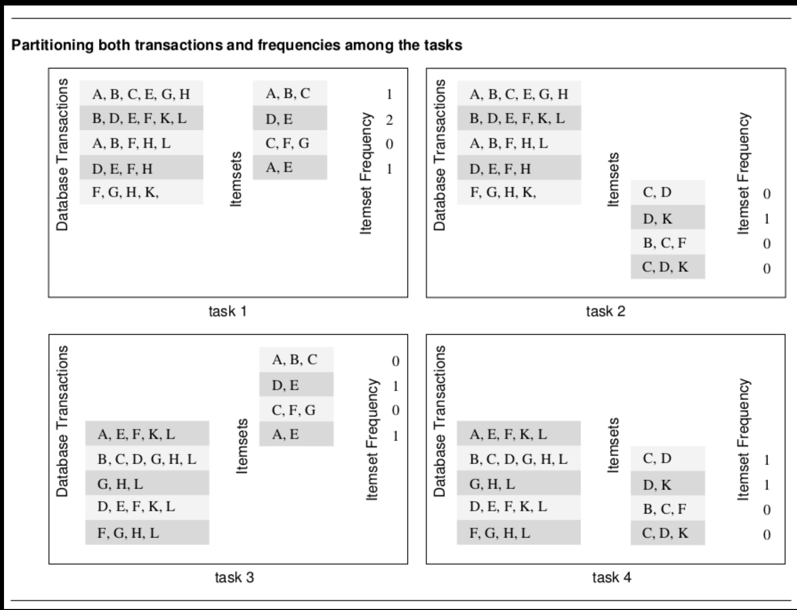


c. Partitioning Input data for Computing Itemset Frequency



Owner-Computes Rule: Partitioning based on input or output data, where a task computes its part based on the partition assigned to it.

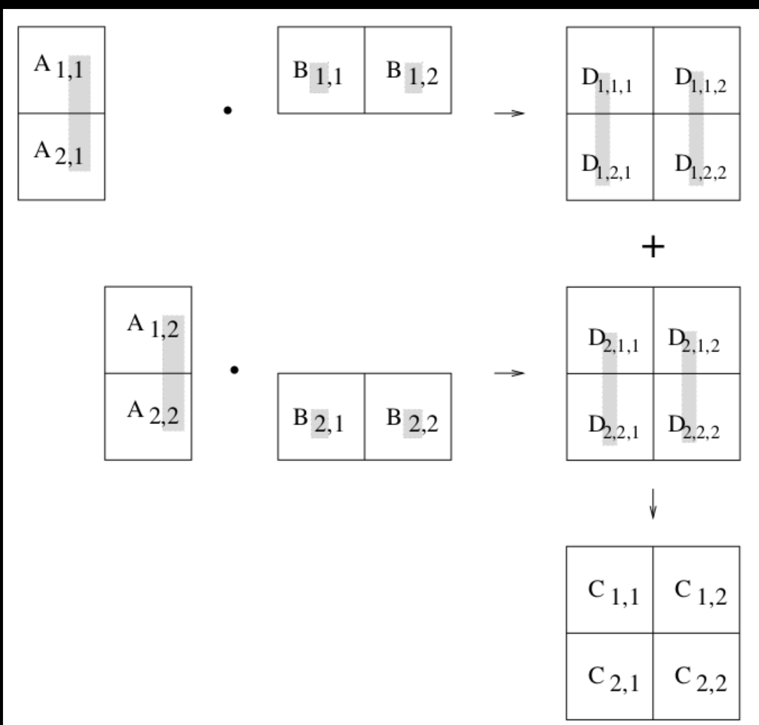
d. Partitioning both Input & Output data for Computing Itemset frequency



e. Partitioning intermediate data for Matrix Multiplication

Multi-Stage Computations (Output of one is input to the next)

Partitioning Output or Input of an intermediate stage leads to a decomposition.



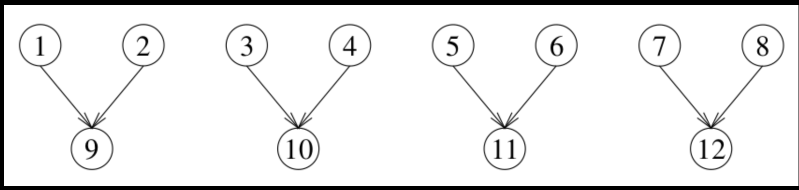
Stage I

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

Stage II

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

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



Maximum Degree of Concurrency? $\boxed{8}$

Tasks 1 - 8 : $O\left(\frac{n^3}{8}\right)$ Tasks 9 - 12 : $O\left(\frac{n^2}{4}\right)$

→ Better degree of Concurrency

→ Needed re-structuring of the algorithm

→ Cost of extra aggregate memory

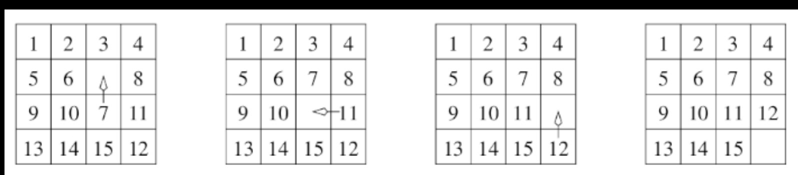
III. Exploratory Decomposition

Used to decompose problems whose underlying computations correspond to a search of a space for solutions.

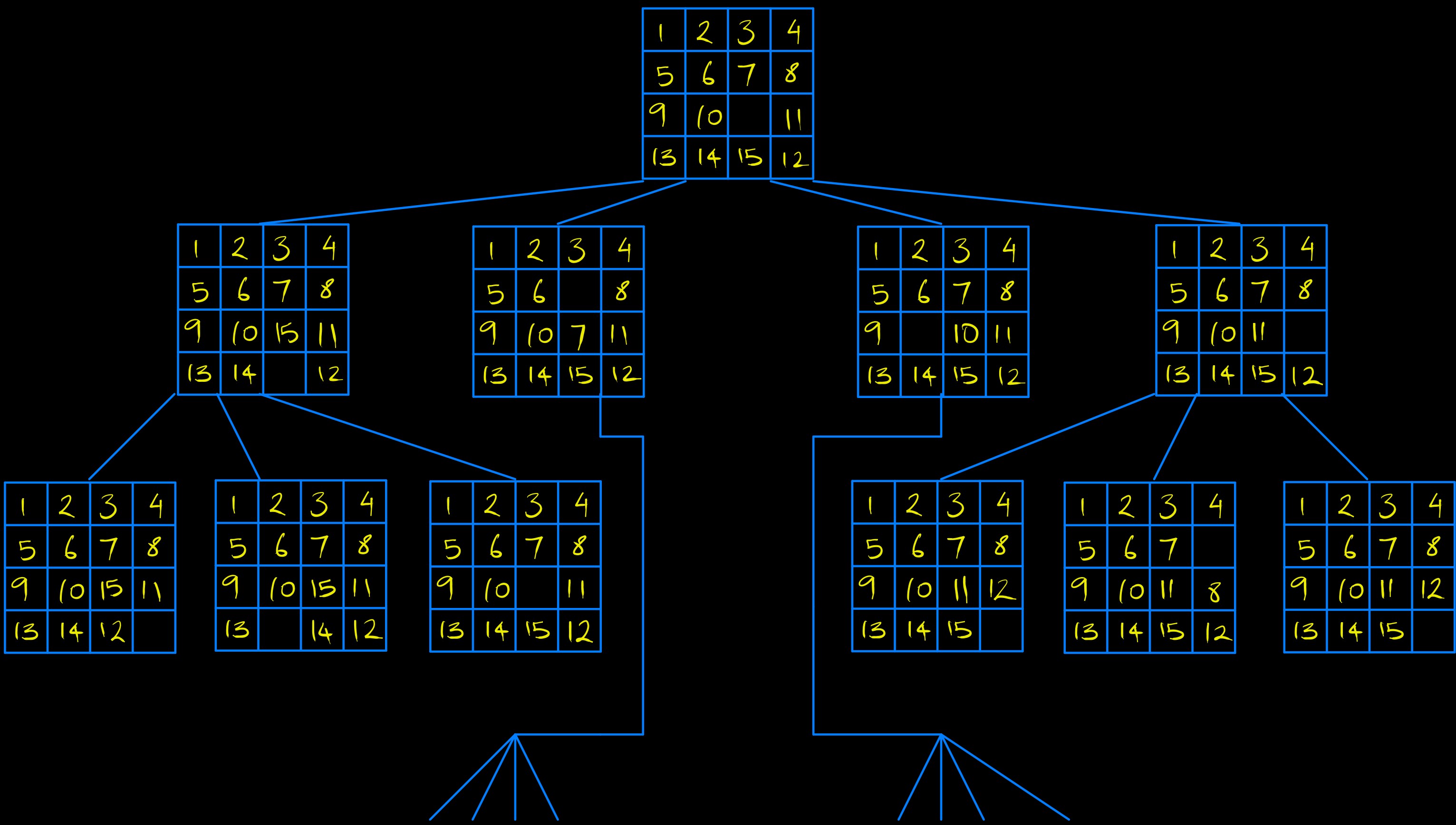
Partition the search space into smaller parts, and search each one of those parts concurrently, until desired solutions are found.

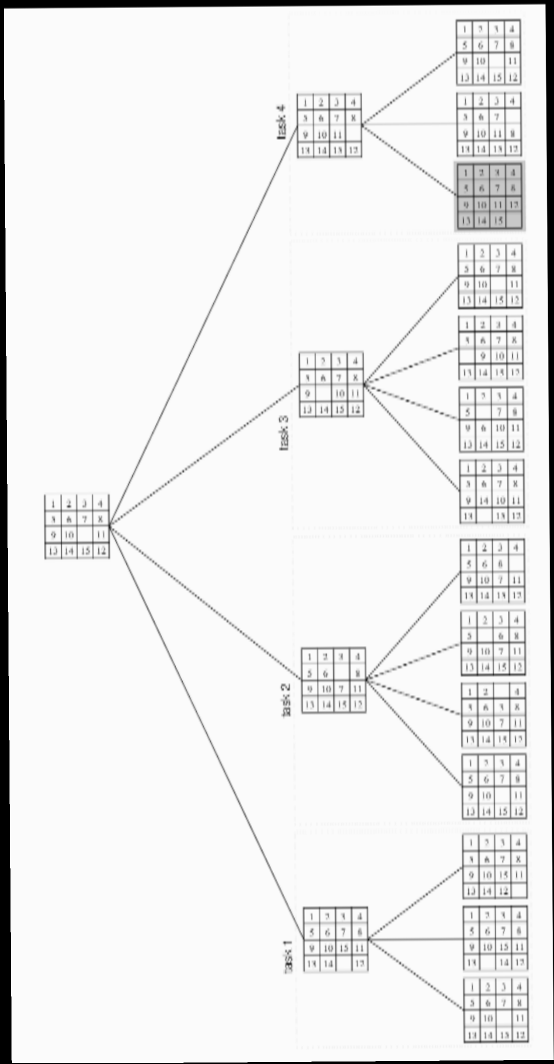
Discrete Optimization problems (Integer Programming)
Theorem proving
Game playing

15-Puzzle Problem :



2, 3 or 4 Successors to be explored at each configuration
Task of finding a path from initial configuration to final is now to find a path to final from one of the explored nodes


1 2 3 4
5 6 7 8
9 10 11
13 14 15 12
1 2 3 4
5 6 7 8
9 10 15 11
13 14 12
1 2 3 4
5 6 8
9 10 7 11
13 14 15 12
1 2 3 4
5 6 7 8
9 10 11
13 14 15 12
1 2 3 4
5 6 7 8
9 10 11
13 14 15 12
1 2 3 4
5 6 7 8
9 10 15 11
13 14 12
1 2 3 4
5 6 7 8
9 10 15 11
13 14 12
1 2 3 4
5 6 7 8
9 10 11
13 14 15 12
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15
1 2 3 4
5 6 7
9 10 11 8
13 14 15 12
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15



→ Generate first few levels serially until search tree has sufficient no. of leaf nodes

→ Assign each leaf node to a task to explore further until at least one of them finds a solution

→ Task that finds the solution to inform others to terminate their searches.

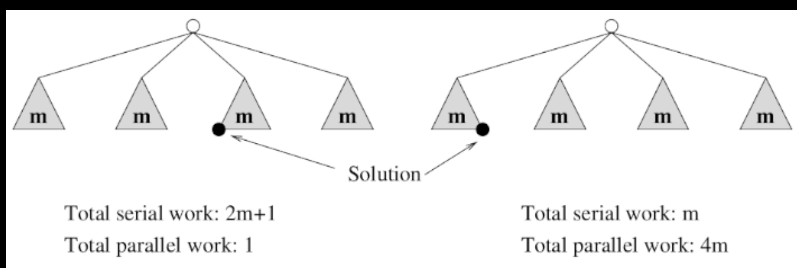
Looks similar to Data decomposition, but different.

Tasks are performed in their entirety.

Unfinished Tasks can be terminated.

Tasks perform useful computations towards the solution of the problem.

Work performed by the parallel formulation can be smaller or greater than the serial algorithm.



Speedup = 2m+1 Speedup = 1 (None)

Sai

IV. Speculative Decomposition

Used when a program may take one of the many possible branches depending on a condition.

One Task ⇒ Execute Condition

Other Tasks ⇒ Concurrently execute one or more branch computations

e.g. Switch Statement : Condition → One Task
Multiple cases → Other Tasks

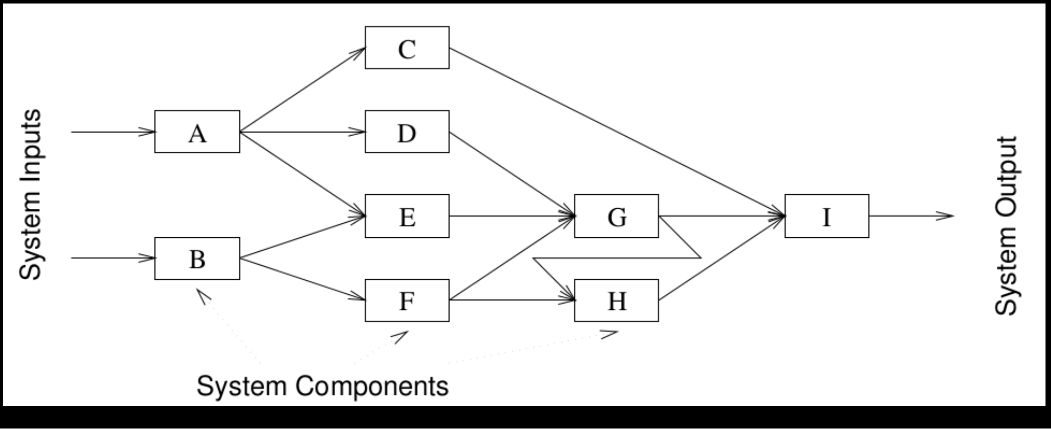
After Condition is executed, the Correct Case is used, while others are discarded.

Faster than serial execution

But to reduce wasteful computations, perform computations of most likely outcomes

If that branch is not taken, discard it (roll back) and perform the right one

e.g. Discrete Event Simulation



V. Hybrid Decompositions

Apply different kinds of decompositions at different stages of the Computation