
Principles of Parallel Algorithm Design: Decomposition and Mapping

Milind Chabbi

Nikhil Hegde

**Department of Computer Science
IIT Dharwad**

Topics for Today

- **Decomposition techniques**
- **Characteristics of tasks and interactions**
- **Mapping techniques for load balancing**

Decomposition Techniques

How should one decompose a task into various subtasks?

- **No single universal recipe**
- **In practice, a variety of techniques are used including**
 - **recursive decomposition**
 - **data decomposition**
 - **exploratory decomposition**
 - **speculative decomposition**

Decomposition Techniques

How should one decompose a task into various subtasks?

- No single universal recipe
- In practice, a variety of techniques are used including
 - recursive decomposition
 - data decomposition
 - exploratory decomposition
 - speculative decomposition

Recursive Decomposition

Suitable for problems solvable using divide-and-conquer

Steps

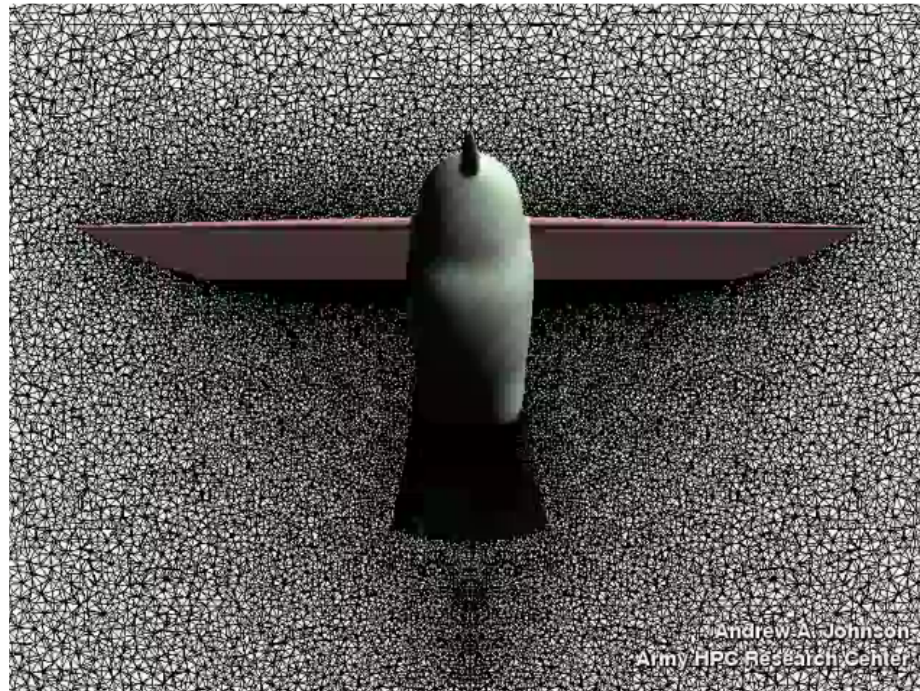
1. **decompose a problem into a set of sub-problems**
2. **recursively decompose each sub-problem**
3. **stop decomposition when minimum desired granularity reached**

Recursive Decomposition

Suitable for problems solvable using divide-and-conquer

Steps

1. **decompose a problem into a set of sub-problems**
2. **recursively decompose each sub-problem**
3. **stop decomposition when minimum desired granularity reached**

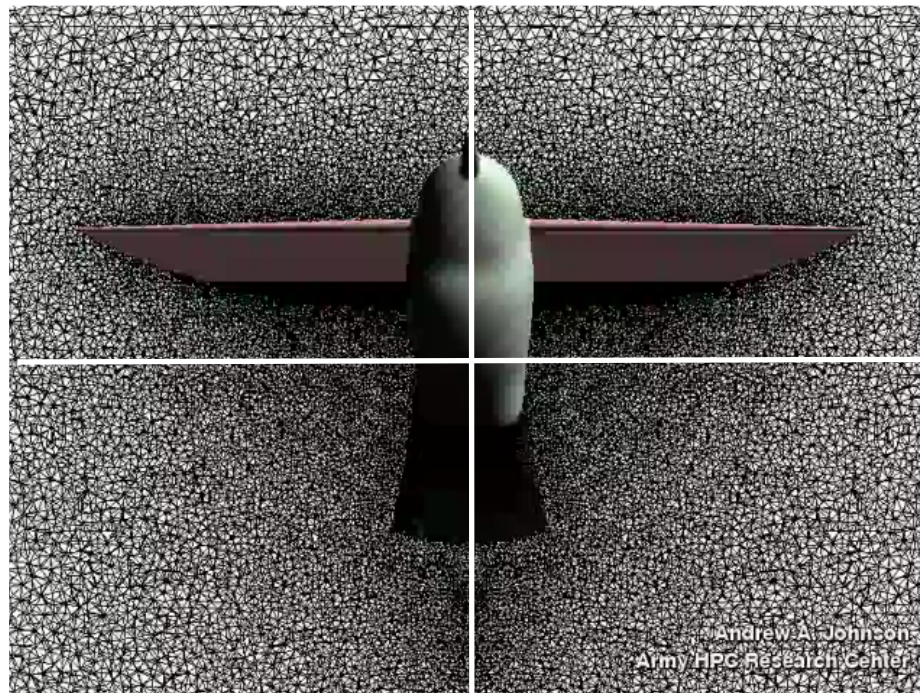


Recursive Decomposition

Suitable for problems solvable using divide-and-conquer

Steps

1. **decompose a problem into a set of sub-problems**
2. **recursively decompose each sub-problem**
3. **stop decomposition when minimum desired granularity reached**

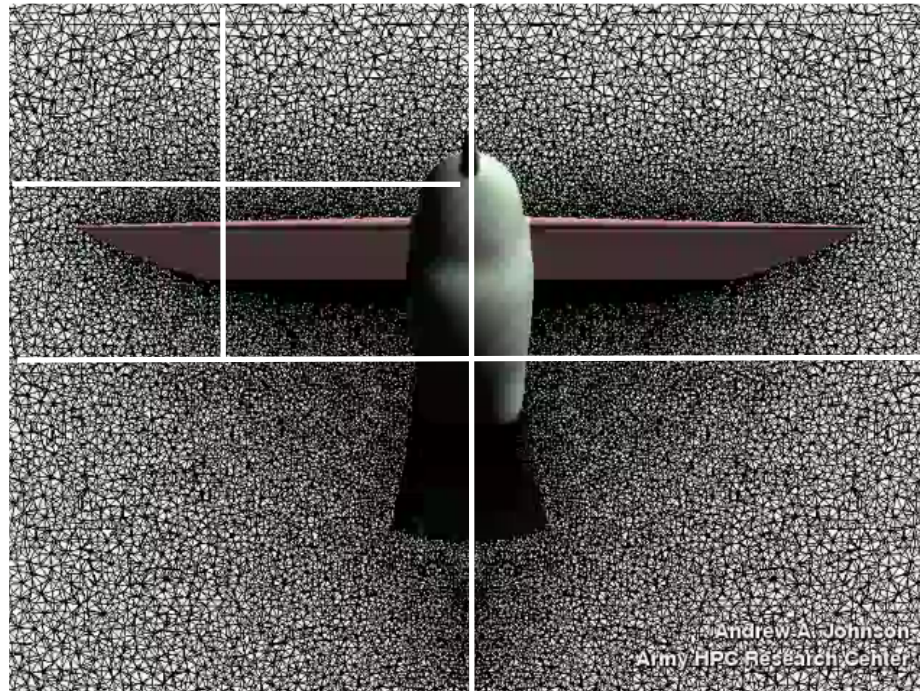


Recursive Decomposition

Suitable for problems solvable using divide-and-conquer

Steps

1. **decompose a problem into a set of sub-problems**
2. **recursively decompose each sub-problem**
3. **stop decomposition when minimum desired granularity reached**

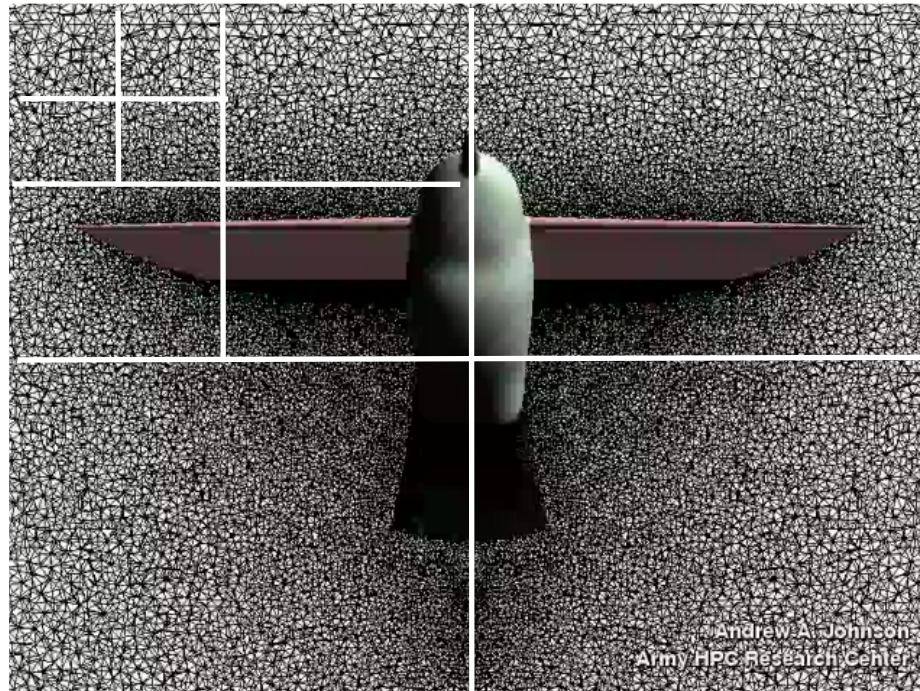


Recursive Decomposition

Suitable for problems solvable using divide-and-conquer

Steps

1. **decompose a problem into a set of sub-problems**
2. **recursively decompose each sub-problem**
3. **stop decomposition when minimum desired granularity reached**

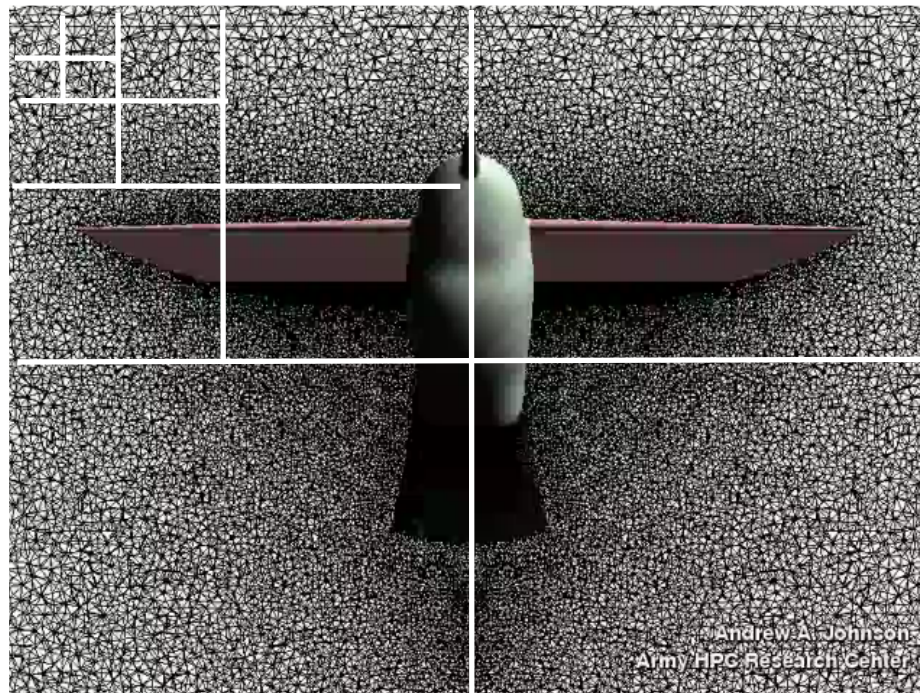


Recursive Decomposition

Suitable for problems solvable using divide-and-conquer

Steps

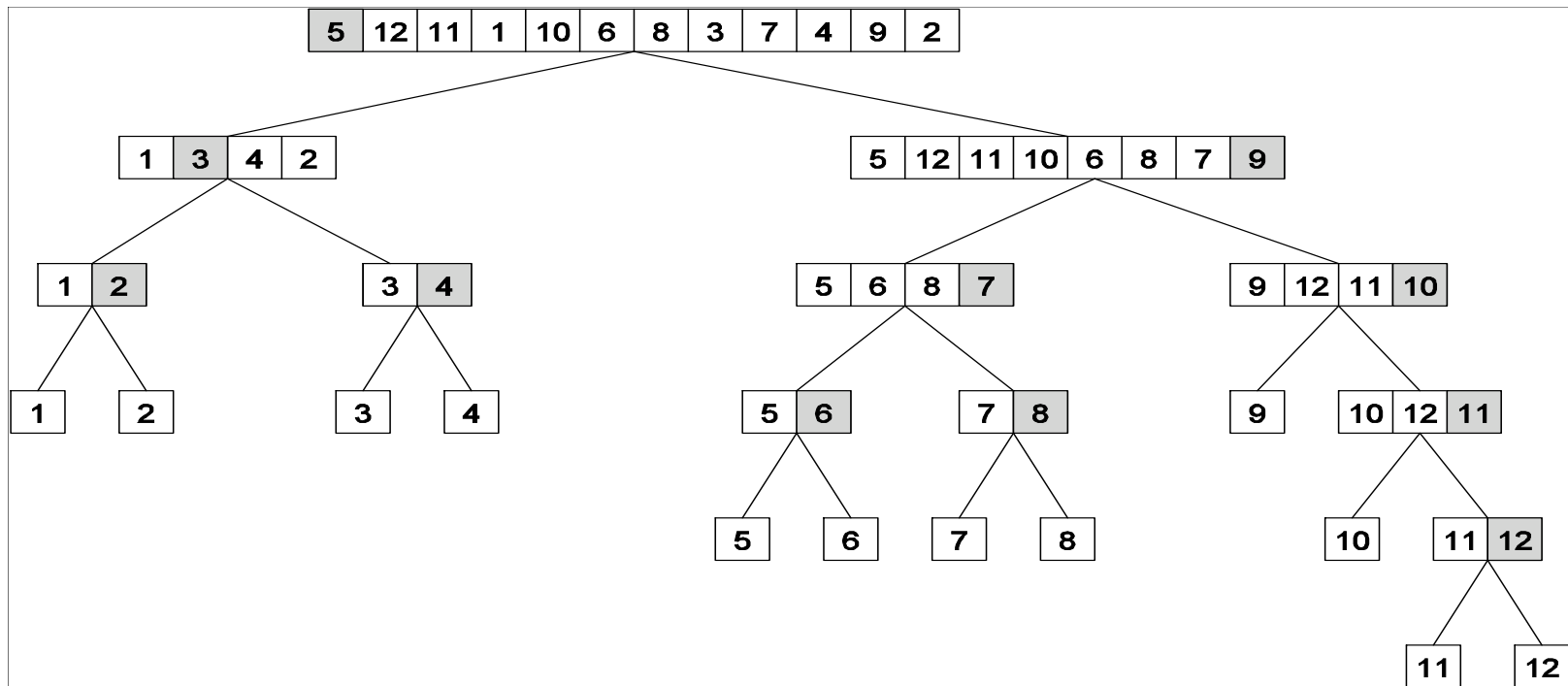
1. **decompose a problem into a set of sub-problems**
2. **recursively decompose each sub-problem**
3. **stop decomposition when minimum desired granularity reached**



Recursive Decomposition for Quicksort

Sort a vector v :

1. **Select a pivot**
2. **Partition v around pivot into v_{left} and v_{right}**
3. **In parallel, sort v_{left} and sort v_{right}**



Recursive Decomposition for Min

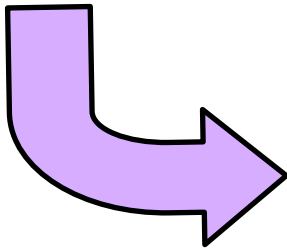
Finding the minimum in a vector using divide-and-conquer

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

Recursive Decomposition for Min

Finding the minimum in a vector using divide-and-conquer

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

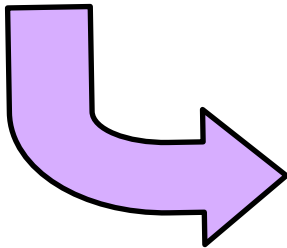


```
procedure RECURSIVE_MIN (A, n)
begin
  if ( n = 1 ) then
    min := A[0];
  else
    lmin := spawn RECURSIVE_MIN(&A[0], n/2 );
    rmin := spawn RECURSIVE_MIN(&A[n/2], n-n/2);
    if (lmin < rmin) then
      min := lmin;
    else
      min := rmin;
  return min;
```

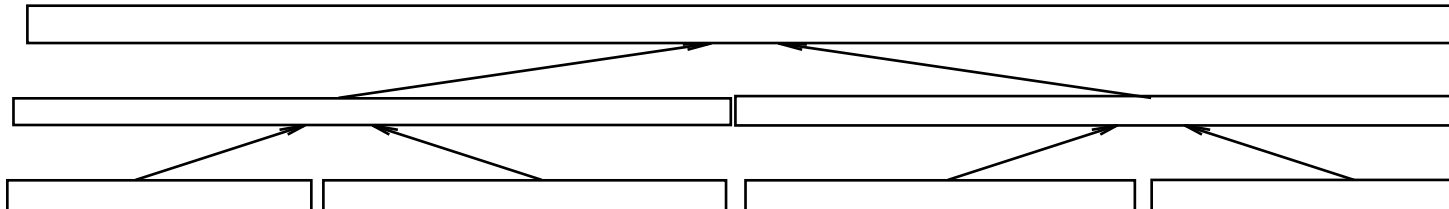
Recursive Decomposition for Min

Finding the minimum in a vector using divide-and-conquer

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



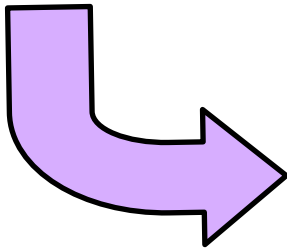
```
procedure RECURSIVE_MIN (A, n)
begin
  if (  $n = 1$  ) then
    min := A[0];
  else
    lmin := spawn RECURSIVE_MIN(&A[0],  $n/2$  );
    rmin := spawn RECURSIVE_MIN(&A[n/2],  $n-n/2$ );
    if (lmin < rmin) then
      min := lmin;
    else
      min := rmin;
  return min;
```



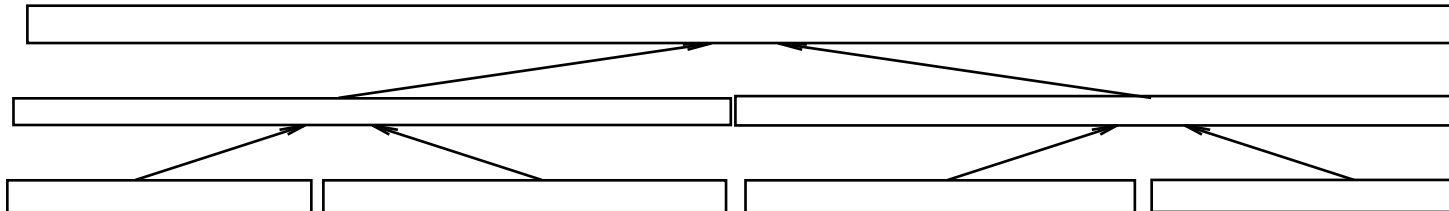
Recursive Decomposition for Min

Finding the minimum in a vector using divide-and-conquer

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



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



Applicable to other associative operations, e.g. sum, AND ...

Decomposition Techniques

- **Decomposition techniques**
 - recursive decomposition
 - data decomposition
 - exploratory decomposition
 - speculative decomposition
- **Characteristics of tasks and interactions**
- **Mapping techniques for load balancing**
- **Methods for minimizing interaction overheads**

Data Decomposition

- **Steps**
 1. identify the data on which computations are performed
 2. partition the data across various tasks
 - partitioning induces a decomposition of the problem
- **Data can be partitioned in various ways**
 - appropriate partitioning is critical to parallel performance
- **Decomposition based on**
 - input data
 - output data
 - input + output data
 - intermediate data

Decomposition Based on Input Data

- Applicable if each output is computed as a function of the input
- May be the only natural decomposition if output is unknown
 - examples
 - finding the minimum in a set or other reductions
 - sorting a vector
- Associate a task with each input data partition
 - task performs computation on its part of the data
 - subsequent processing combines partial results from earlier tasks

Example: Decomposition Based on Input Data

Count the frequency of item sets in database transactions

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

- Partition computation by partitioning the set of transactions
 - a task computes a local count for each item set for its transactions

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

task 1

task 2

- sum local count vectors for item sets to produce total count vector

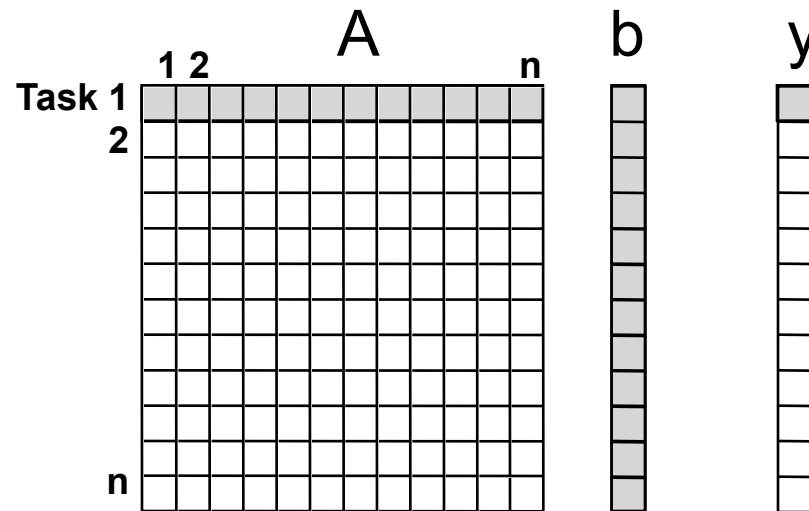
Decomposition Based on Output Data

- If each element of the output can be computed independently
- Partition the output data across tasks
- Have each task perform the computation for its outputs

Decomposition Based on Output Data

- If each element of the output can be computed independently
- Partition the output data across tasks
- Have each task perform the computation for its outputs

**Example:
dense matrix-vector
multiply**



Output Data Decomposition: Example

- **Matrix multiplication: $C = A \times B$**
- **Computation of C can be partitioned into four tasks**

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

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

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

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

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

Output Data Decomposition: Example

- **Matrix multiplication: $C = A \times B$**
- **Computation of C can be partitioned into four tasks**

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

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

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

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

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

Other task decompositions possible

Example: Decomposition Based on Output Data

Count the frequency of item sets in database transactions

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

- Partition computation by partitioning the item sets to count
 - each task computes total count for each of its item sets

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

task 1

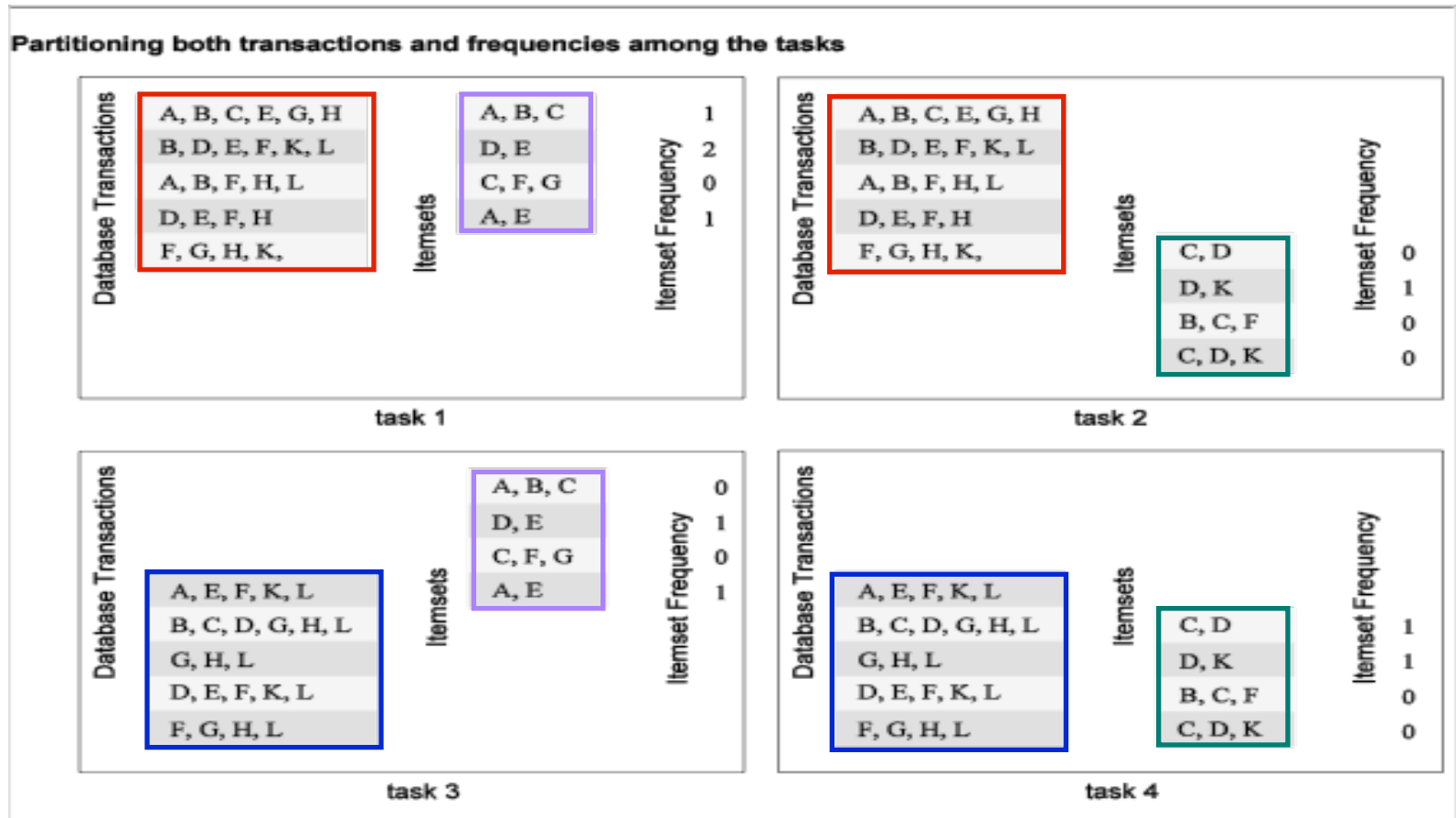
Database Transactions	A, B, C, E, G, H	Itemsets	C, D	1	Itemset Frequency
	B, D, E, F, K, L		D, K	2	
	A, B, F, H, L		B, C, F	0	
	D, E, F, H		C, D, K	0	
	F, G, H, K,				
	A, E, F, K, L				Itemset Frequency
	B, C, D, G, H, L				
	G, H, L				
	D, E, F, K, L				
	F, G, H, L				

task 2

- append total counts for item subsets to produce result

Partitioning Input *and* Output Data

- Partition on both input and output for more concurrency
- Example: item set counting



Intermediate Data Partitioning

- If computation is a sequence of transforms
 - (from input data to output data)
- Can decompose based on data for intermediate stages

Example: Intermediate Data Partitioning

Dense Matrix Multiply

Decomposition of intermediate data: yields 8 + 4 tasks

Stage I

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

Stage II

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

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

Intermediate Data Partitioning: Example

Tasks: dense matrix multiply decomposition of intermediate data

Task 01: $D_{1,1,1} = A_{1,1} B_{1,1}$

Task 03: $D_{1,1,2} = A_{1,1} B_{1,2}$

Task 05: $D_{1,2,1} = A_{2,1} B_{1,1}$

Task 07: $D_{1,2,2} = A_{2,1} B_{1,2}$

Task 09: $C_{1,1} = D_{1,1,1} + D_{2,1,1}$

Task 11: $C_{2,1} = D_{1,2,1} + D_{2,2,1}$

Task 02: $D_{2,1,1} = A_{1,2} B_{2,1}$

Task 04: $D_{2,1,2} = A_{1,2} B_{2,2}$

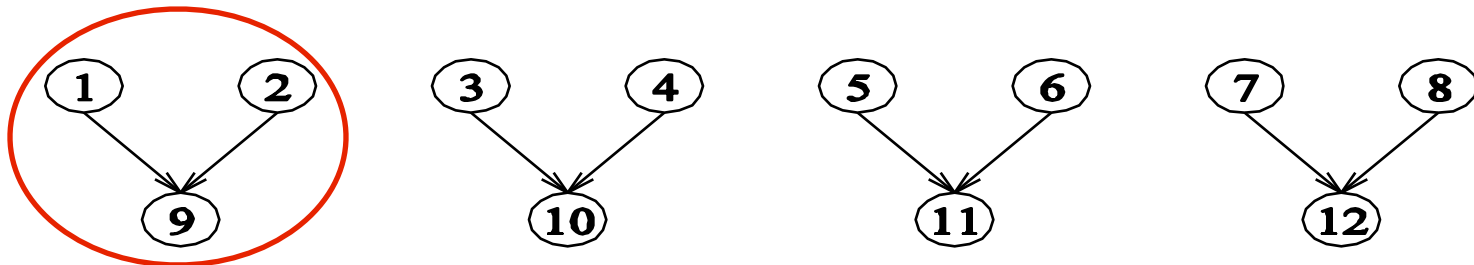
Task 06: $D_{2,2,1} = A_{2,2} B_{2,1}$

Task 08: $D_{2,2,2} = A_{2,2} B_{2,2}$

Task 10: $C_{1,2} = D_{1,1,2} + D_{2,1,2}$

Task 12: $C_{2,2} = D_{1,2,2} + D_{2,2,2}$

Task dependency graph



Owner Computes Rule

- Each datum is assigned to a thread
- Each thread computes values associated with its data
- Implications
 - input data decomposition
 - all computations using an input datum are performed by its thread
 - output data decomposition
 - an output is computed by the thread assigned to the output data

Topics for Today

- **Decomposition techniques**
 - recursive decomposition
 - data decomposition
 -  —exploratory decomposition
 - speculative decomposition

Exploratory Decomposition

- Exploration (search) of a state space of solutions
 - problem decomposition reflects shape of execution
- Examples
 - game playing
 - discrete optimization
 - 0/1 integer programming
 - theorem proving

Exploratory Decomposition Example

Solving a 15 puzzle

- Sequence of three moves from state (a) to final state (d)


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

- From an arbitrary state, must search for a solution

Exploratory Decomposition Example

Solving a 15 puzzle

- Sequence of three moves from state (a) to final state (d)


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

- From an arbitrary state, must search for a solution

Exploratory Decomposition Example

Solving a 15 puzzle

- Sequence of three moves from state (a) to final state (d)

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


- From an arbitrary state, must search for a solution

Exploratory Decomposition Example

Solving a 15 puzzle

- Sequence of three moves from state (a) to final state (d)

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


- From an arbitrary state, must search for a solution

Exploratory Decomposition Example

Solving a 15 puzzle

- Sequence of three moves from state (a) to final state (d)

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


- From an arbitrary state, must search for a solution


Exploratory Decomposition Example

Solving a 15 puzzle

- Sequence of three moves from state (a) to final state (d)

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


- From an arbitrary state, must search for a solution


Exploratory Decomposition Example

Solving a 15 puzzle

- Sequence of three moves from state (a) to final state (d)

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	11	12
13	14	15	

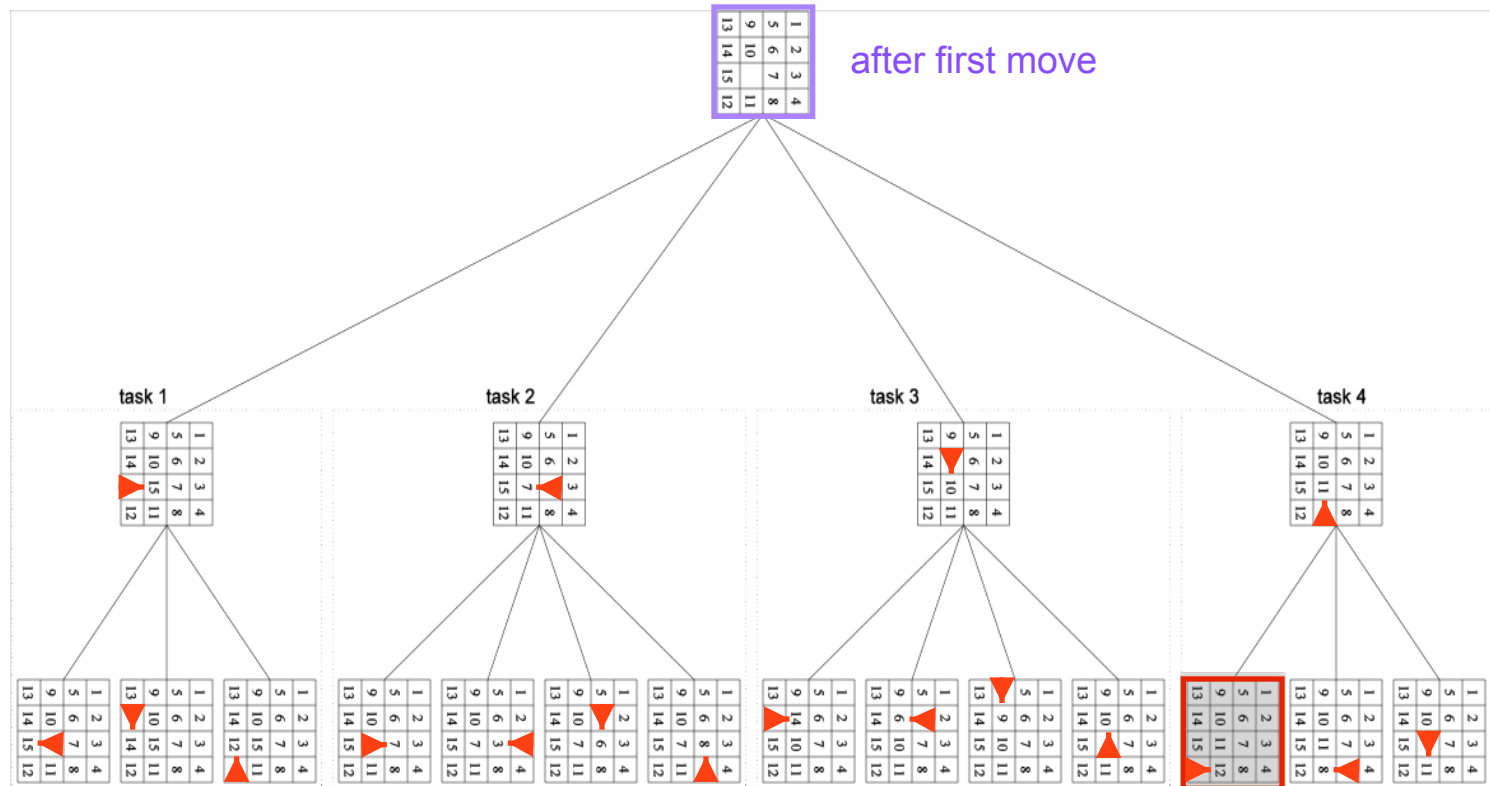
- From an arbitrary state, must search for a solution

Exploratory Decomposition: Example

Solving a 15 puzzle

Search

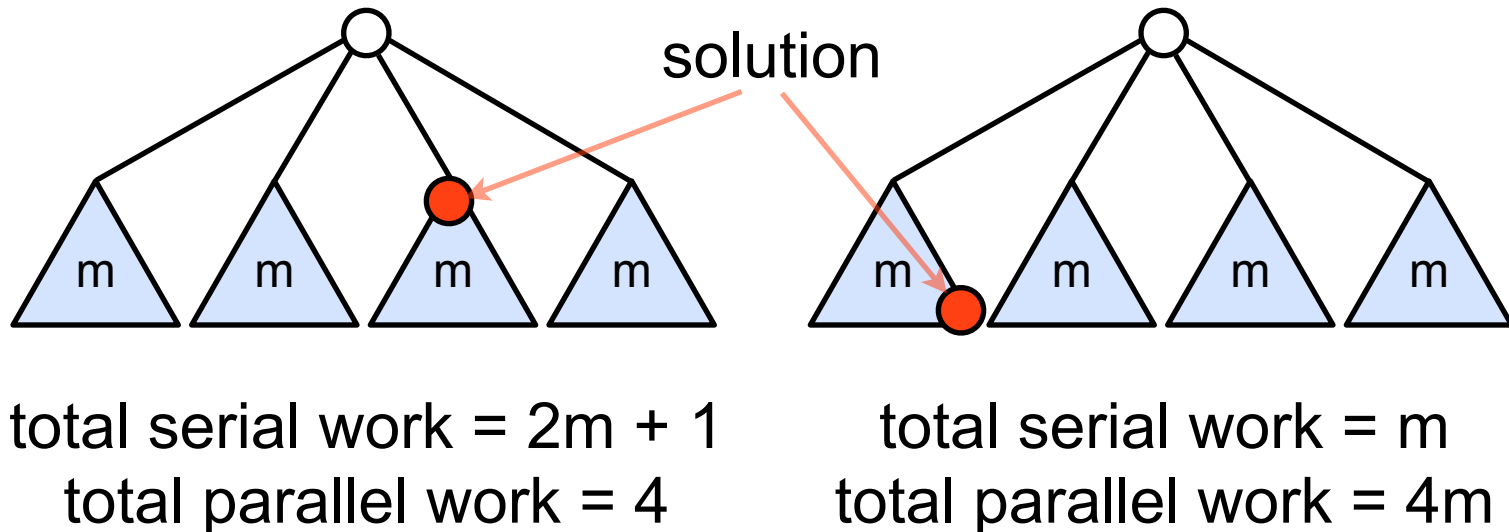
- generate successor states of the current state
- explore each as an independent task



final state (solution)

Exploratory Decomposition Speedup

- Parallel formulation may perform a different amount of work



- Can cause super- or sub-linear speedup

Topics for Today

- **Decomposition techniques**
 - recursive decomposition
 - data decomposition
 - exploratory decomposition
 - speculative decomposition



Speculative Decomposition

- Dependencies between tasks are not always known a-priori
 - makes it impossible to identify independent tasks
- Conservative approach
 - identify independent tasks only when no dependencies left
- Optimistic (speculative) approach
 - schedule tasks even when they may potentially be erroneous
- Drawbacks for each
 - conservative approaches
 - may yield little concurrency
 - optimistic approaches
 - may require a roll-back mechanism if a dependence is encountered

Speculative Decomposition in Practice

Discrete event simulation

- Data structure: centralized time-ordered event list
- Simulation
 - extract next event in time order
 - process the event
 - if required, insert new events into the event list
- Optimistic event scheduling
 - assume outcomes of all prior events
 - speculatively process next event
 - if assumption is incorrect, roll back its effects and continue

Time Warp

David Jefferson. "Virtual Time,"
ACM TOPLAS, 7(3):404-425, July 1985

Speculative Decomposition in Practice

Time Warp OS <http://bit.ly/twos-94>

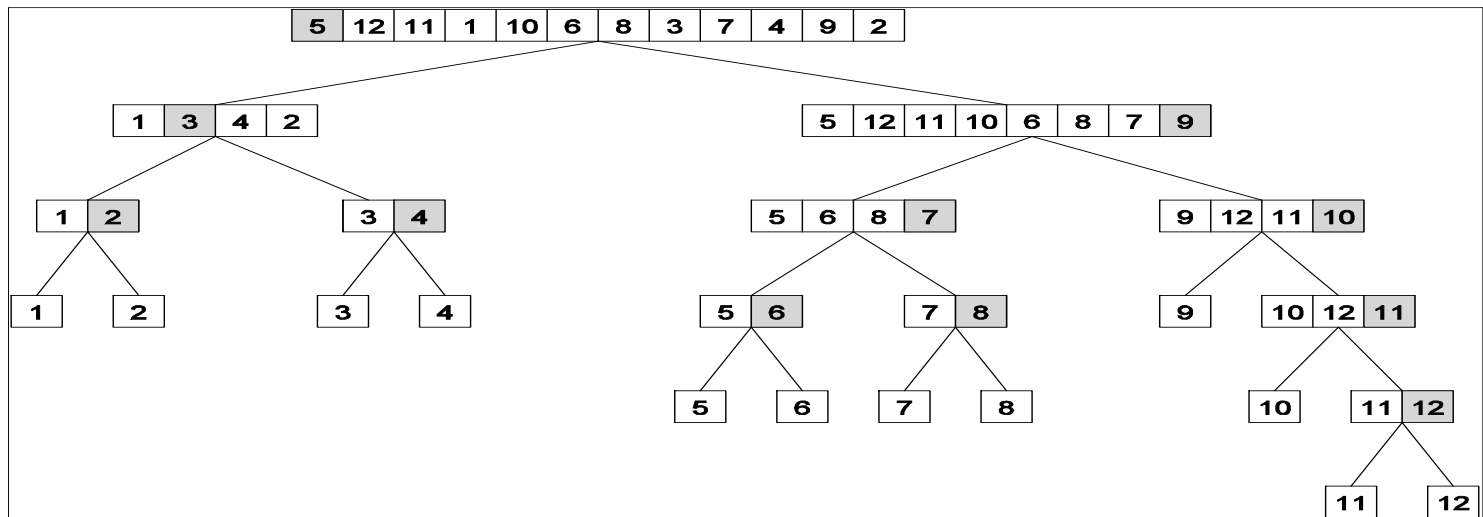
- A special operating system for military simulations
 - expensive computational tasks
 - composed of many interacting subsystems
 - highly irregular temporal behavior
- Optimistic execution and process rollback
 - don't treat rollback as a special case for handling exceptions, breaking deadlock, aborting transactions, ...
 - use rollback as frequently as other systems use blocking
- Why a special OS?
 - rollback forces a rethinking of all OS issues
 - scheduling, synchronization, message queueing, flow control, memory management, error handling, I/O, and commitment
 - building Time Warp on top of an OS would require two levels of synchronization, two levels of message queues, ...

Hybrid Decomposition

Use multiple decomposition strategies together

Often necessary for adequate concurrency

- Quicksort
 - recursive decomposition alone limits concurrency

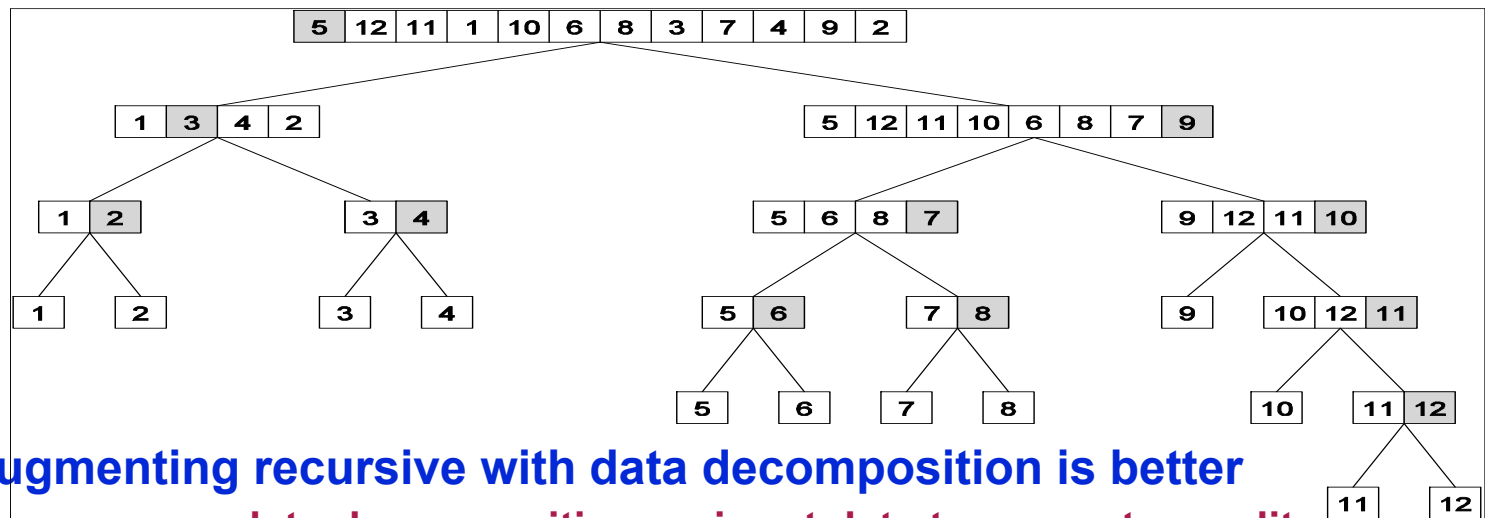


Hybrid Decomposition

Use multiple decomposition strategies together

Often necessary for adequate concurrency

- Quicksort
 - recursive decomposition alone limits concurrency



—augmenting recursive with data decomposition is better

— can use data decomposition on input data to compute a split

Topics for Today

- **Decomposition techniques**
 - recursive decomposition
 - data decomposition
 - exploratory decomposition
 - speculative decomposition
- **Characteristics of tasks and interactions**
- **Mapping techniques for load balancing**

Characteristics of Tasks

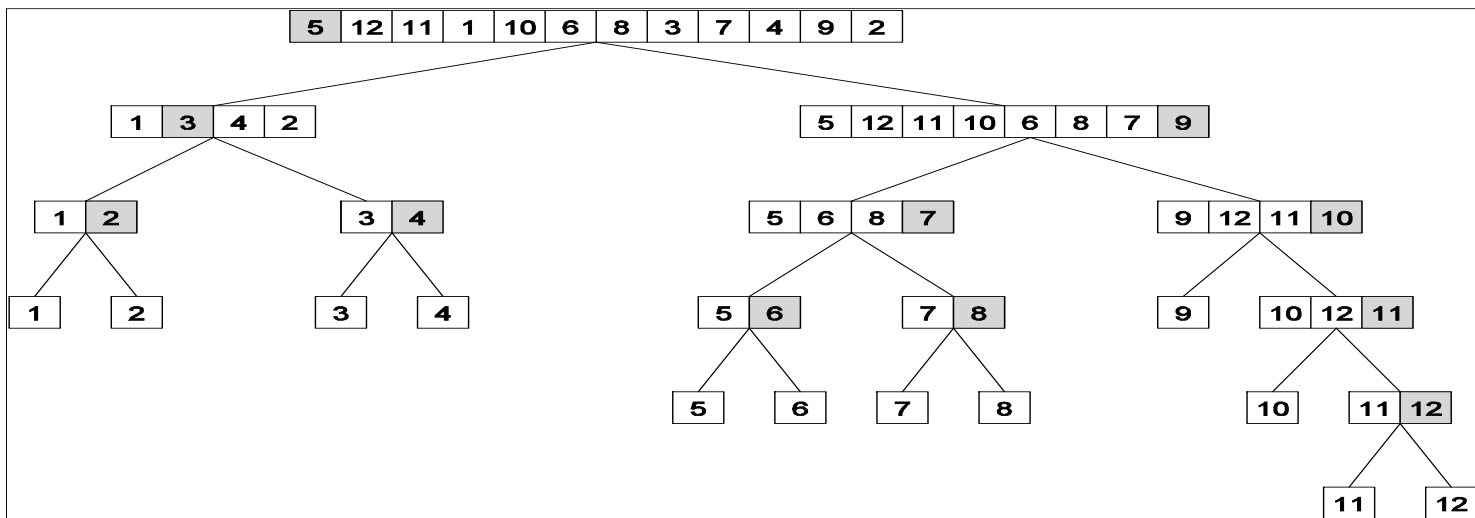
- **Key characteristics**
 - generation strategy
 - associated work
 - associated data size
- **Impact choice and performance of parallel algorithms**

Task Generation

- **Static task generation**
 - identify concurrent tasks a-priori
 - typically decompose using data or recursive decomposition
 - examples
 - matrix operations
 - graph algorithms on static graphs
 - image processing applications
 - other regularly structured problems
- **Dynamic task generation**
 - identify concurrent tasks as a computation unfolds
 - typically decompose using exploratory or speculative decompositions
 - examples
 - puzzle solving
 - game playing

Task Size

- Uniform: all the same size
 - Non-uniform
 - sometimes sizes known or can be estimated *a-priori*
 - sometimes not
 - example: tasks in quicksort
- size of each partition depends upon pivot selected



Size of Data Associated with Tasks

- Data may be small or large compared to the computation
 - $\text{size(input)} < \text{size(computation)}$, e.g., 15 puzzle
 - $\text{size(input)} = \text{size(computation)} > \text{size(output)}$, e.g., min
 - $\text{size(input)} = \text{size(output)} < \text{size(computation)}$, e.g., sort
- Implications
 - small data: task can easily migrate to another thread
 - large data: ties the task to a thread
 - possibly can avoid communicating the task context
reconstruct/recompute the context elsewhere

Characteristics of Task Interactions

Orthogonal classification criteria

- **Static vs. dynamic**
- **Regular vs. irregular**
- **Read-only vs. read-write**
- **One-sided vs. two-sided**

Characteristics of Task Interactions

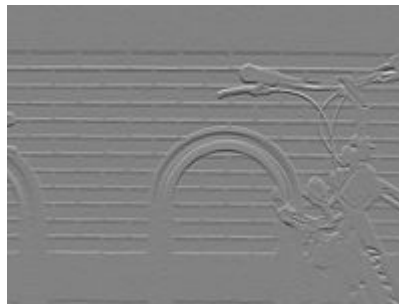
- **Static interactions**
 - tasks and interactions are known a-priori
 - simpler to code
- **Dynamic interactions**
 - timing or interacting tasks cannot be determined a-priori
 - harder to code
 - especially using two-sided message passing APIs

Characteristics of Task Interactions

- **Regular interactions**
 - interactions have a pattern that can be described with a function
 - e.g. mesh, ring
 - regular patterns can be exploited for efficient implementation
 - e.g. schedule communication to avoid conflicts on network links
- **Irregular interactions**
 - lack a well-defined topology
 - modeled by a graph

Static Regular Task Interaction Pattern

Image operations, e.g., edge detection



$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

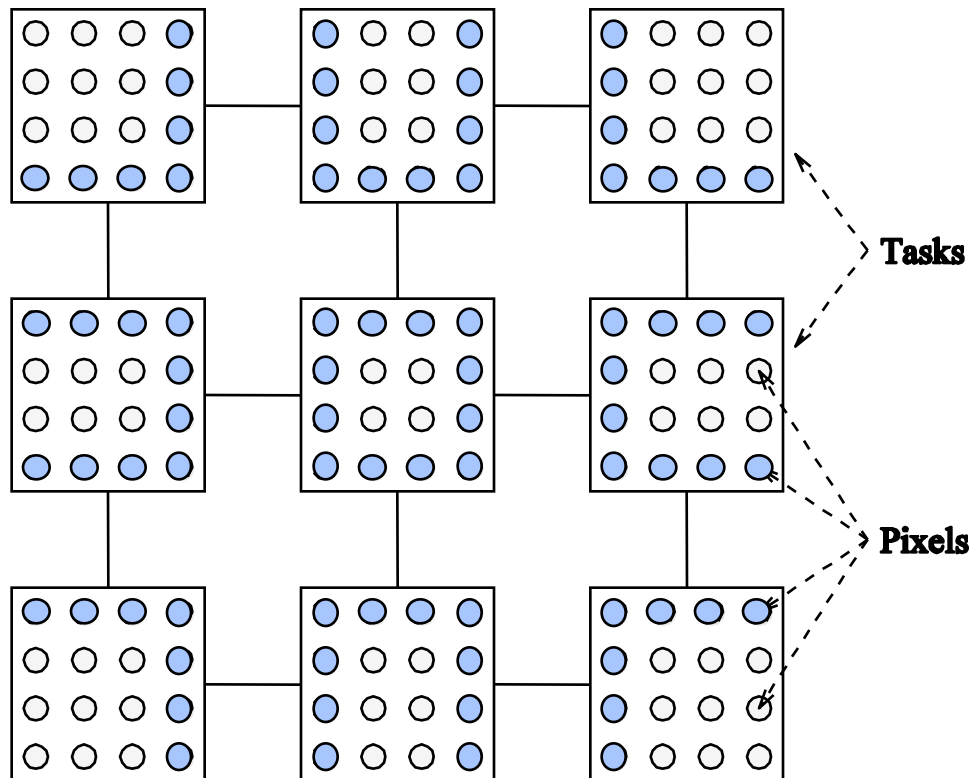
$$G = \sqrt{G_x^2 + G_y^2}$$

Sobel Edge Detection Stencils

Static Regular Task Interaction Pattern

Image operations, e.g., edge detection

Nearest neighbor interactions on a 2D mesh



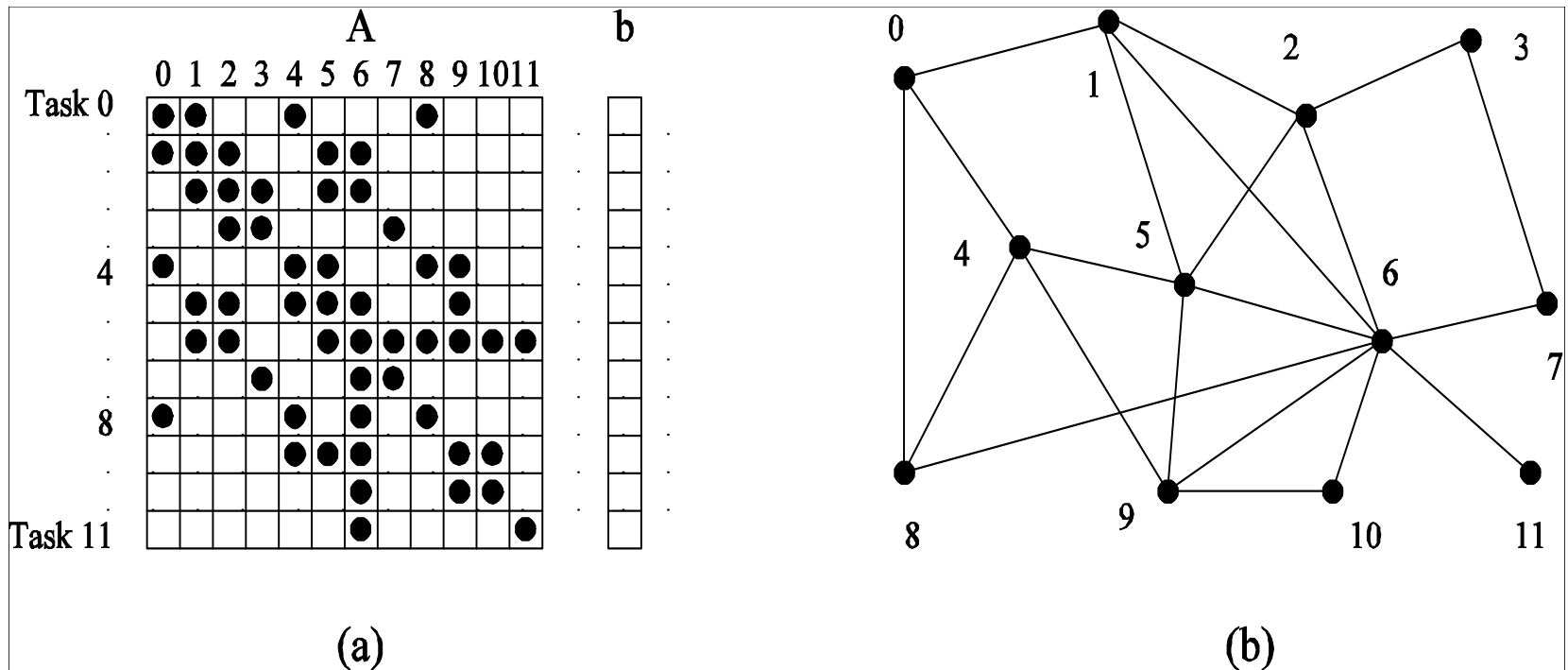
Sobel Edge
Detection Stencils

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Static Irregular Task Interaction Pattern

Sparse matrix-vector multiply




Characteristics of Task Interactions

- **Read-only interactions**
 - tasks only read data associated with other tasks
- **Read-write interactions**
 - read and modify data associated with other tasks
 - harder to code: requires synchronization
 - need to avoid read-write and write-write ordering races

Characteristics of Task Interactions

- **One-sided**
 - initiated & completed independently by 1 of 2 interacting tasks
 - READ or WRITE
 - GET or PUT
- **Two-sided**
 - both tasks coordinate in an interaction
 - SEND and RECV

Topics for Today

- **Decomposition techniques**
 - recursive decomposition
 - data decomposition
 - exploratory decomposition
 - speculative decomposition
- **Characteristics of tasks and interactions**
-  **Mapping techniques for load balancing**

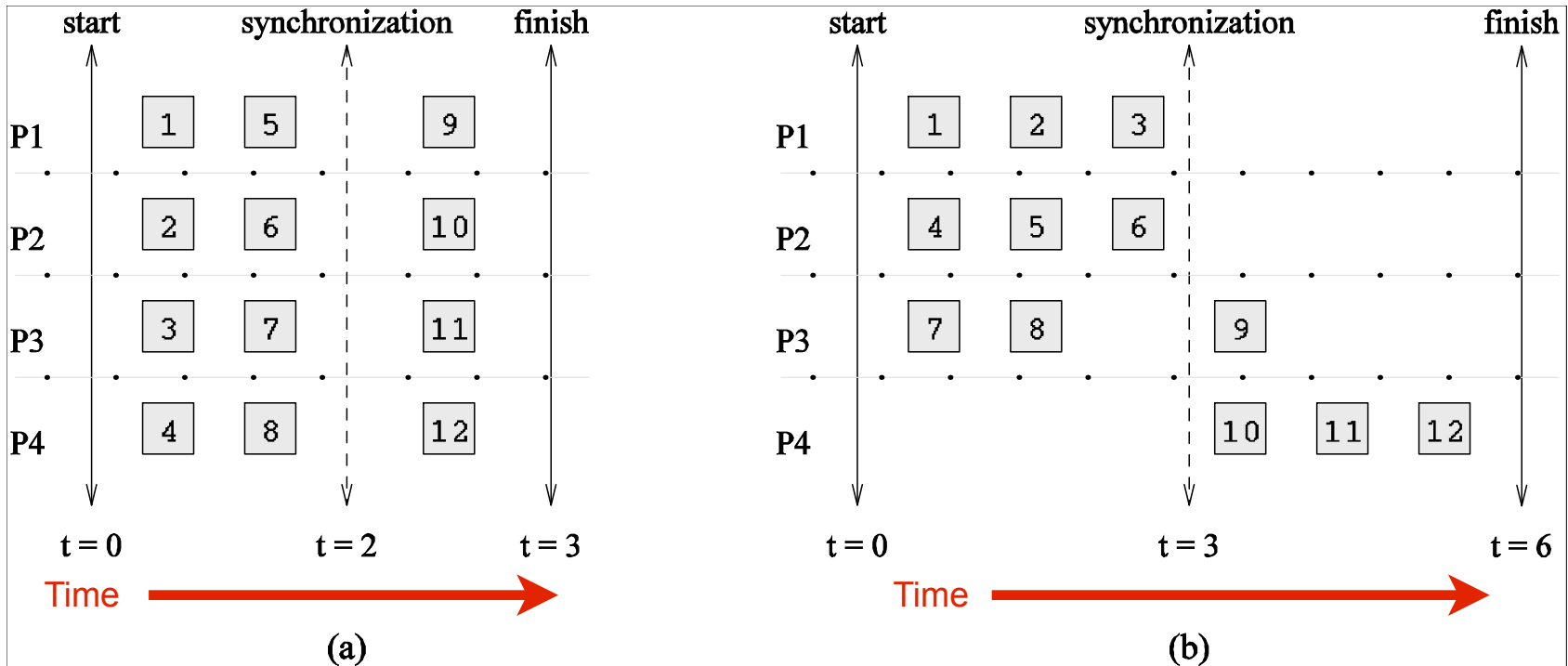
Mapping Techniques

Map concurrent tasks to threads for execution

- **Overheads of mappings**
 - serialization (idling)
 - communication
- **Select mapping to minimize overheads**
- **Conflicting objectives: minimizing one increases the other**
 - assigning all work to one thread
 - minimizes communication
 - significant idling
 - minimizing serialization introduces communication

Mapping Techniques for Minimum Idling

- Must simultaneously minimize idling and load balance
- Balancing load alone does not minimize idling



Mapping Techniques for Minimum Idling

Static vs. dynamic mappings

- **Static mapping**
 - *a-priori* mapping of tasks to threads or processes
 - requirements
 - a good estimate of task size
 - even so, computing an optimal mapping may be NP hard
 - e.g., even decomposition analogous to bin packing
 - https://en.wikipedia.org/wiki/Bin_packing_problem
- **Dynamic mapping**
 - map tasks to threads or processes at runtime
 - why?
 - tasks are generated at runtime, or
 - their sizes are unknown

Factors that influence choice of mapping

- size of data associated with a task
- nature of underlying domain

References

- **Adapted from slides “Principles of Parallel Algorithm Design” by Ananth Grama**
- **Based on Chapter 3 of “Introduction to Parallel Computing” by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. Addison Wesley, 2003**