Principles of Parallel Algorithm Design:

Concurrency and Decomposition

John Mellor-Crummey

Department of Computer Science
Rice University

johnmc@rice.edu

Parallel Algorithm

Recipe to solve a problem using multiple processors

Typical steps for constructing a parallel algorithm

— **identify what pieces of work can be performed concurrently**

— **partition and map work onto independent processors**

— **distribute a program's input, output, and intermediate data**

— **coordinate accesses to shared data: avoid conflicts**

— **ensure proper order of work using synchronization**

**Why "typical"? Some of the steps may be omitted.**

— **if data is in shared memory, distributing it may be unnecessary**

— **if using message passing, there may not be shared data**

— **the mapping of work to processors can be done statically by the programmer or dynamically by the runtime**

Topics for Today

- **Introduction to parallel algorithms**

    —**tasks and decomposition**

    —**threads and mapping**

    —**threads versus cores**

- **Decomposition techniques - part 1**

    —**recursive decomposition**

    —**data decomposition**

Decomposing Work for Parallel Execution

- **Divide work into tasks that can be executed concurrently**

- **Many different decompositions possible for any computation**

- **Tasks may be same, different, or even indeterminate sizes**

- **Tasks may be independent or have non-trivial order**

- **Conceptualize tasks and ordering as task dependency DAG**
  - **—node = task**
  - **—edge = control dependence**
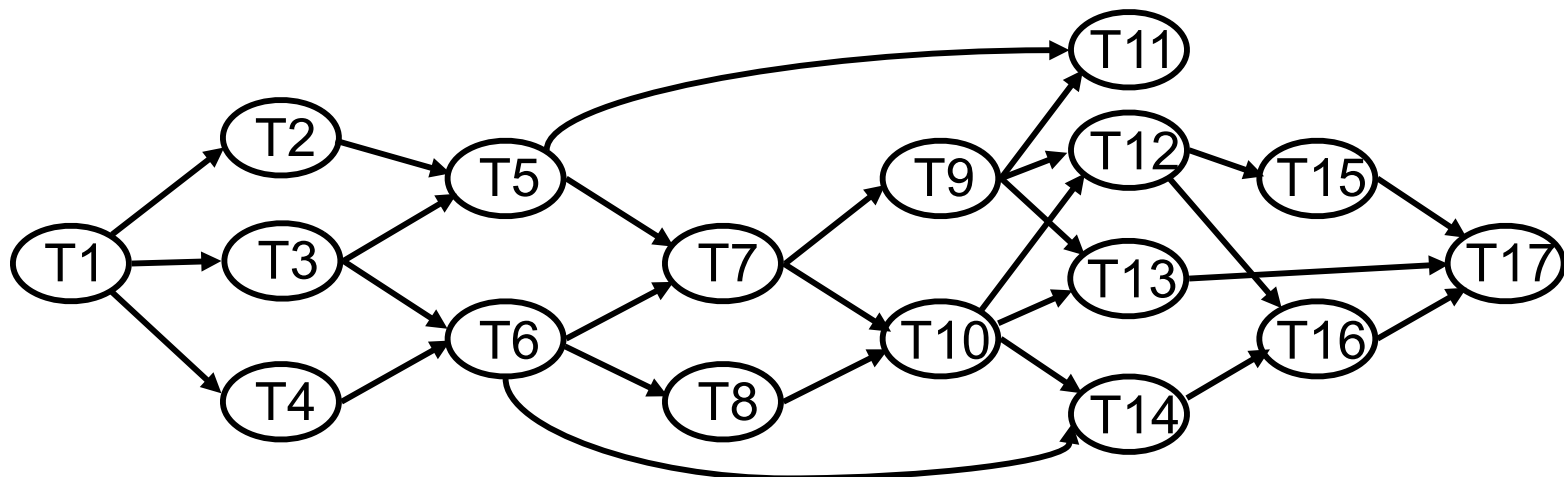
Example: Dense Matrix-Vector Product
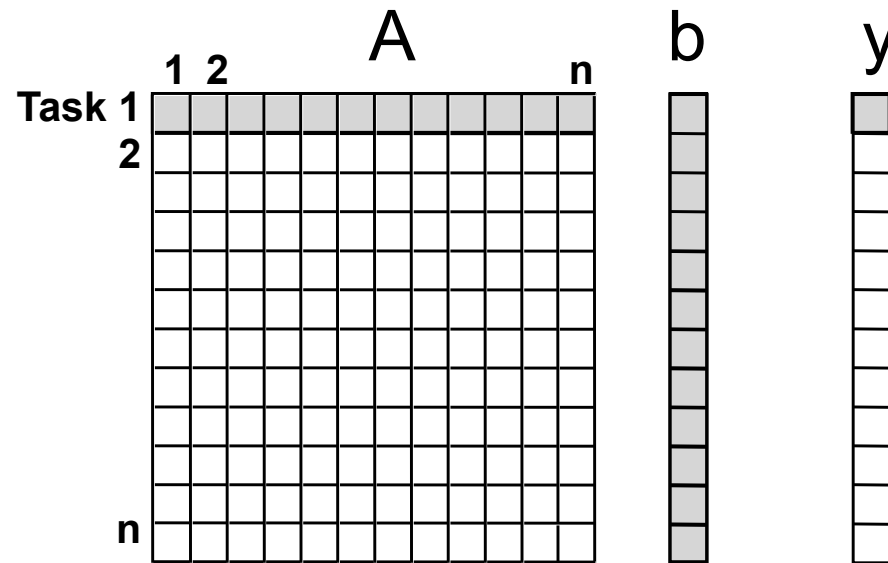
- **Computing each element of output vector y is independent**

- **Easy to decompose dense matrix-vector product into tasks**
  - **—one per element in y**

- **Observations**
  - **—task size is uniform**
  - **—no control dependences between tasks**
  - **—tasks share b**

5

Example: Database Query Processing

Consider the execution of the query:

MODEL = "CIVIC" AND YEAR = 2001 AND
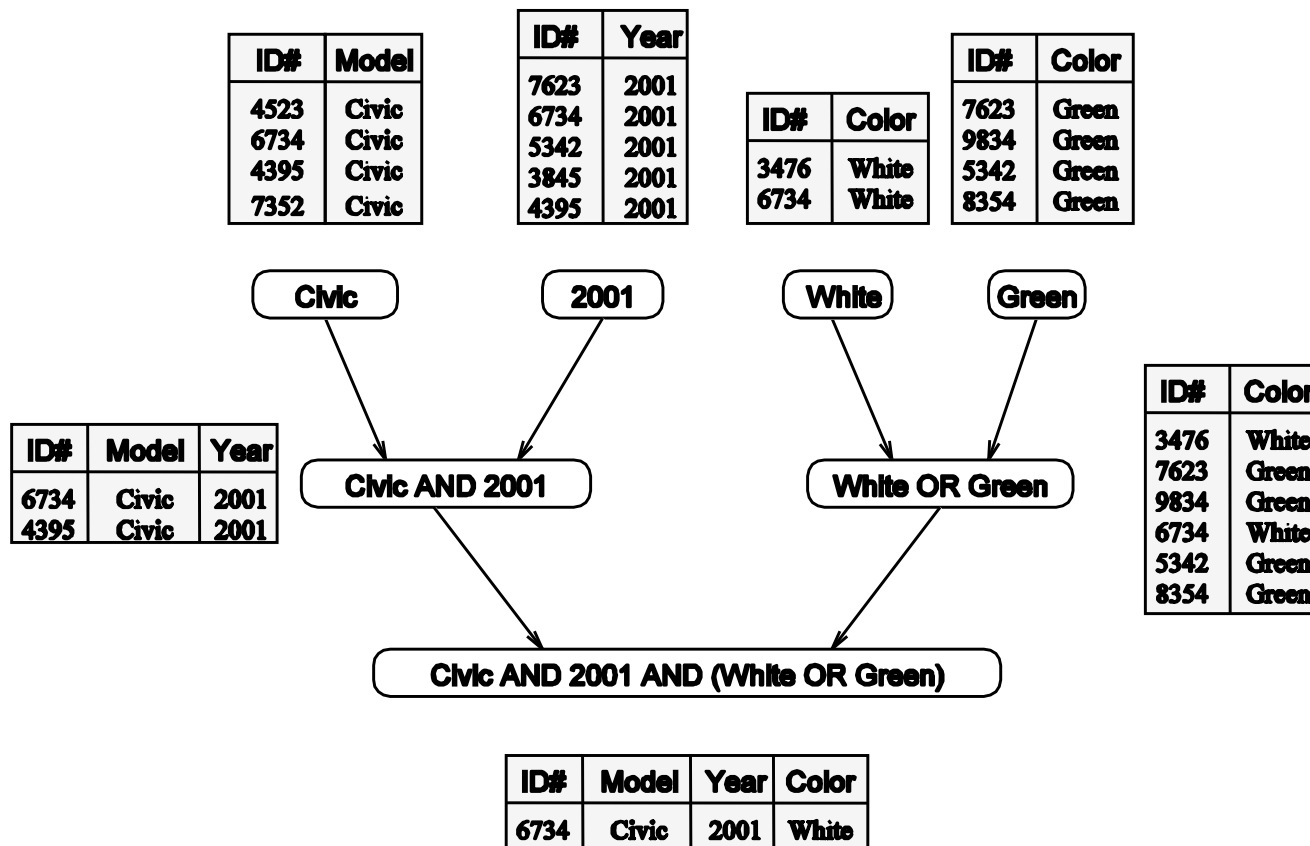
(COLOR = "GREEN" OR COLOR = "WHITE")

on the following database:

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

Example: Database Query Processing

- **Task: compute set of elements that satisfy a predicate**
  - —task result = table of entries that satisfy the predicate

- **Edge: output of one task serves as input to the next**

  MODEL = "CIVIC" AND YEAR = 2001 AND

  (COLOR = "GREEN" OR COLOR = "WHITE")

ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

Civic   2001   White   Green

ID#	Model	Year
6734	Civic	2001
4395	Civic	2001

Civic AND 2001

White OR Green

ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

Civic AND 2001 AND (White OR Green)

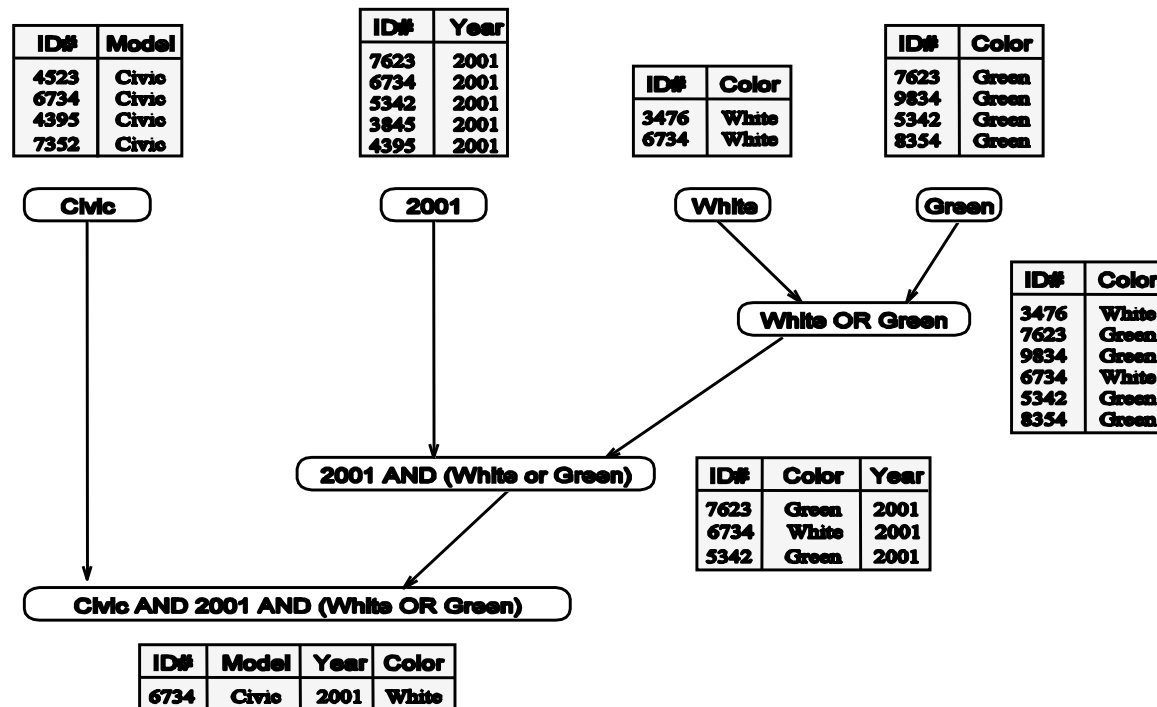| ID#	Model	Year	Color
6734	Civic	2001	White

7

Example: Database Query Processing

- **Alternate task decomposition for query**

MODEL = "CIVIC" AND YEAR = 2001 AND

(COLOR = "GREEN" OR COLOR = "WHITE")

ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

Civic   2001   White   Green

White OR Green

ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

2001 AND (White or Green)

ID#	Color	Year
7623	Green	2001
6734	White	2001
5342	Green	2001

Civic AND 2001 AND (White OR Green)

ID#	Model	Year	Color
6734	Civic	2001	White

- **Different decompositions may yield different parallelism and different amounts of work**

Granularity of Task Decompositions

- **Granularity = task size**
  - —depends on the number of tasks
- **Fine-grain = large number of tasks**
- **Coarse-grain = small number of tasks**
- **Granularity examples for dense matrix-vector multiply**
  - —fine-grain: each task represents an individual element in y
  - —coarser-grain: each task computes 3 elements in y
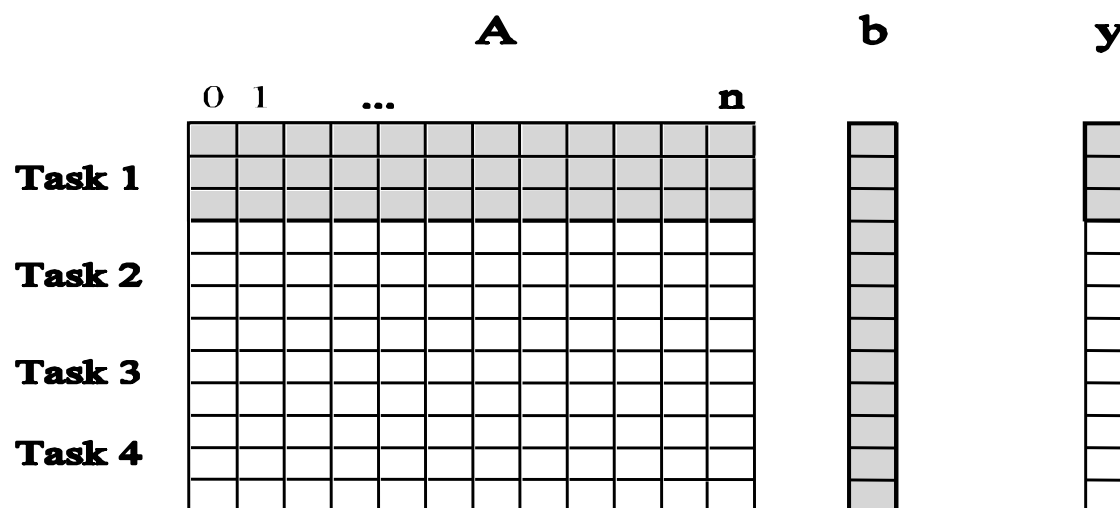
Degree of Concurrency

- **Definition: number of tasks that can execute in parallel**

- **May change during program execution**

- **Metrics**

  —**maximum degree of concurrency**
  - **largest # concurrent tasks at any point in the execution**

  —**average degree of concurrency**
  - **average number of tasks that can be processed in parallel**

- **Degree of concurrency vs. task granularity**

  —**inverse relationship**

Example: Dense Matrix-Vector
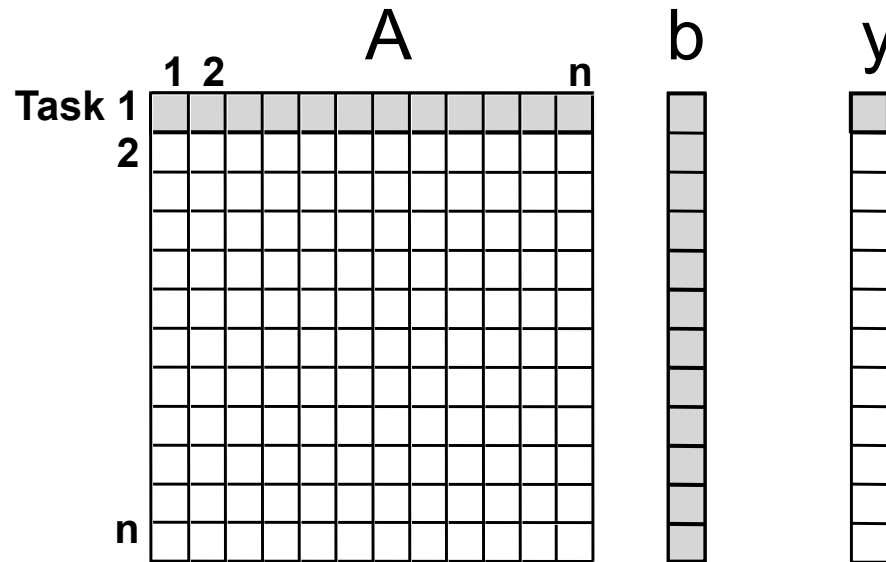
- **Computing each element of output vector y is independent**

- **Easy to decompose dense matrix-vector product into tasks**
  - **—one per element in y**

- **Observations**
  - **—task size is uniform**
  - **—no control dependences between tasks**
  - **—tasks share b**

  Question: Is *n* the maximum number of tasks possible?

Critical Path

- **Edge in task dependency graph represents task serialization**

- **Critical path = longest weighted path though graph**

- **Critical path length = lower bound on parallel execution time**

Critical Path Length

Examples: database query task dependency graphs
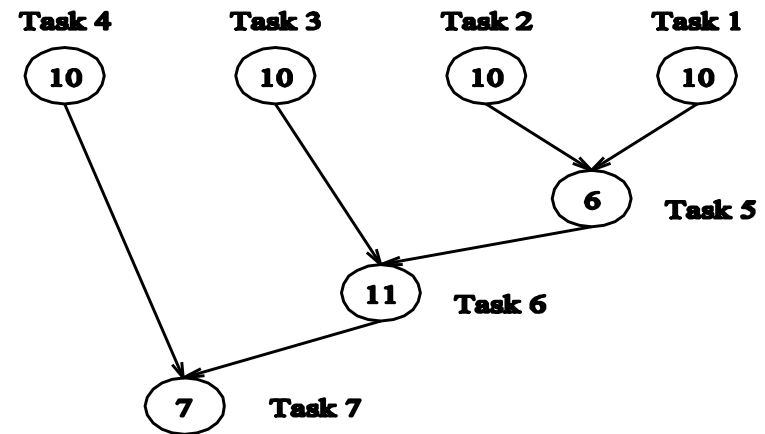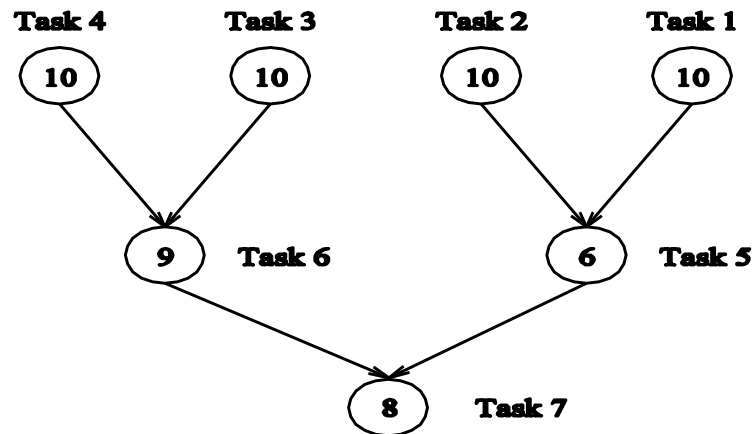
Note: number in vertex represents task cost

Questions:

What are the tasks on the critical path for each dependency graph?

What is the shortest parallel execution time for each decomposition?

How many processors are needed to achieve the minimum time?

What is the maximum degree of concurrency?

What is the average parallelism?

Critical Path Length

Example: dependency graph for dense-matrix vector product
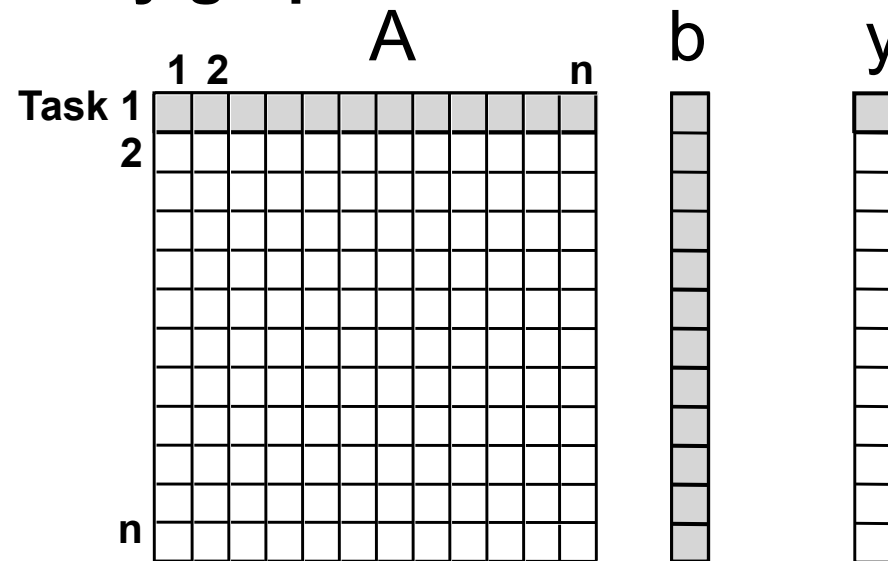
Questions:

What does a task dependency graph look like for DMVP?

What is the shortest parallel execution time for the graph?

How many processors are needed to achieve the minimum time?

What is the maximum degree of concurrency?

What is the average parallelism?

Limits on Parallel Performance

- **What bounds parallel execution time?**
  - **—minimum task granularity**
    - **e.g. dense matrix-vector multiplication ≤ n^2 concurrent tasks**
  - **—dependencies between tasks**
  - **—parallelization overheads**
    - **e.g., cost of communication between tasks**
  - **—fraction of application work that can't be parallelized**
    - **more about Amdahl's law in a later lecture ...**

- **Measures of parallel performance**
  - **—speedup = T_1/T_p**
  - **—parallel efficiency = $T_1/(pT_p)$**

Task Interaction Graphs

- **Tasks generally exchange data with others**
  - **example: dense matrix-vector multiply**
    - if vector b is not replicated in all tasks, tasks will have to communicate elements of b

- **Task interaction graph**
  - **node = task**
  - **edge = interaction or data exchange**

- **Task interaction graphs vs. task dependency graphs**
  - *task interaction graphs* represent <u>data dependences</u>
  - *task dependency graphs* represent <u>control dependences</u>

Task Interaction Graph Example

Sparse matrix-vector multiplication

- Computation of each result element = independent task

- Only non-zero elements of sparse matrix A participate

- If, b is partitioned among tasks …

  — structure of the task interaction graph = graph of the matrix A
  (i.e. the graph for which A represents the adjacency structure)
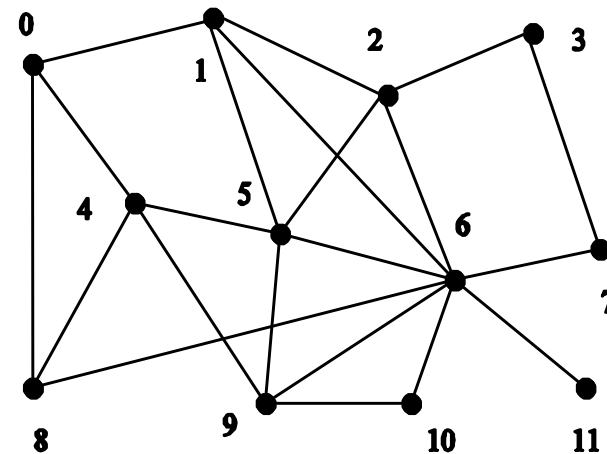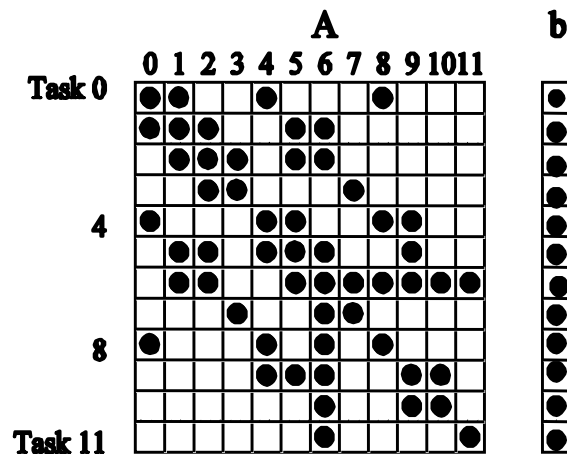
Interaction Graphs, Granularity, & Communication

- Finer task granularity increases communication overhead

- Example: sparse matrix-vector product interaction graph
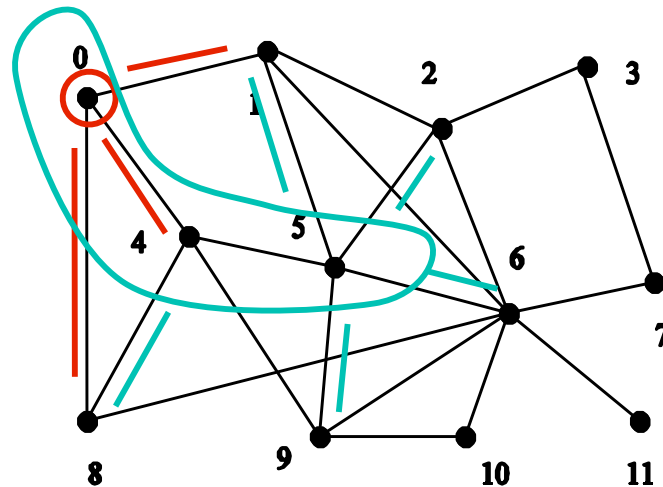
- Assumptions:
  — each node takes unit time to process
  — each interaction (edge) causes an overhead of a unit time

- If node 0 is a task: communication = 3; computation = 4

- If nodes 0, 4, and 5 are a task: communication = 5; computation = 15
  — coarser-grain decomposition → smaller communication/computation

18

Tasks, Threads, and Mapping

- **Generally**
  - **—# of tasks > # threads available**
  - **—parallel algorithm must map tasks to threads**

- **Why threads rather than CPU cores?**
  - **—aggregate tasks into threads**
    - **thread = processing or computing agent that performs work**
    - **assign collection of tasks and associated data to a thread**
  - **—operating system maps threads to physical cores**
    - **operating systems often enable one to bind a thread to a core**
    - **for multithreaded cores, the OS can bind multiple software threads to distinct hardware threads associated with a core**

Tasks, Threads, and Mapping

- **Mapping tasks to threads is critical for parallel performance**

- **On what basis should one choose mappings?**
  - **—using task dependency graphs**
    - – **schedule independent tasks on separate threads**
      - **minimum idling**
      - **optimal load balance**
  - **—using task interaction graphs**
    - – **want threads to have minimum interaction with one another**
      - **minimum communication**

Tasks, Threads, and Mapping

A good mapping must minimize parallel execution time by

- **Mapping independent tasks to different threads**

- **Assigning tasks on critical path to threads ASAP**

- **Minimizing interactions between threads**
  - **—map tasks with dense interactions to the same thread**

- **Difficulty: criteria often conflict with one another**
  - **—e.g. no decomposition minimizes interactions but no speedup!**

Tasks, Threads, and Mapping Example
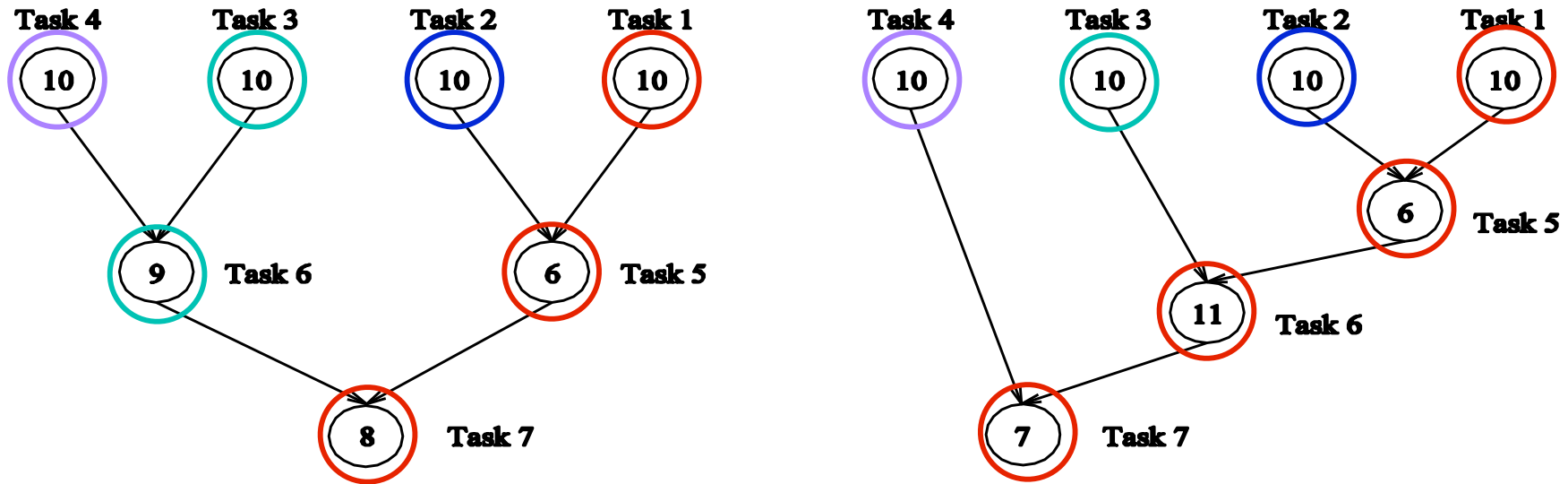
Example: mapping database queries to threads

- **Consider the dependency graphs in levels**
  - *no nodes in a level depend upon one another*
  - *compute levels using topological sort*
- **Assign all tasks within a level to different threads**

Topics for Today

- **Introduction to parallel algorithms**
  - —tasks and decomposition
  - —threads and mapping
  - —threads versus cores

- **Decomposition techniques - part 1**
  - — **recursive decomposition**
  - — **data 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
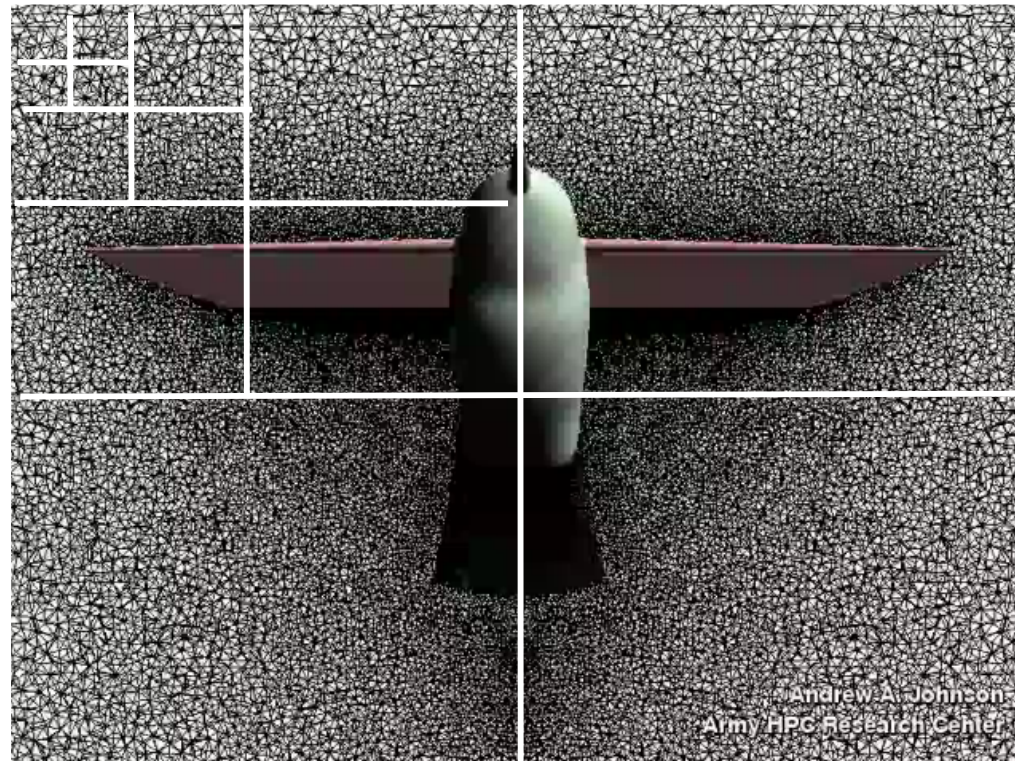
Recursive Decomposition for Quicksort

Sort a vector v:

1. **Select a pivot**

2. **Partition v around pivot into vleft and vright**

3. **In parallel, sort vleft and sort vright**

26
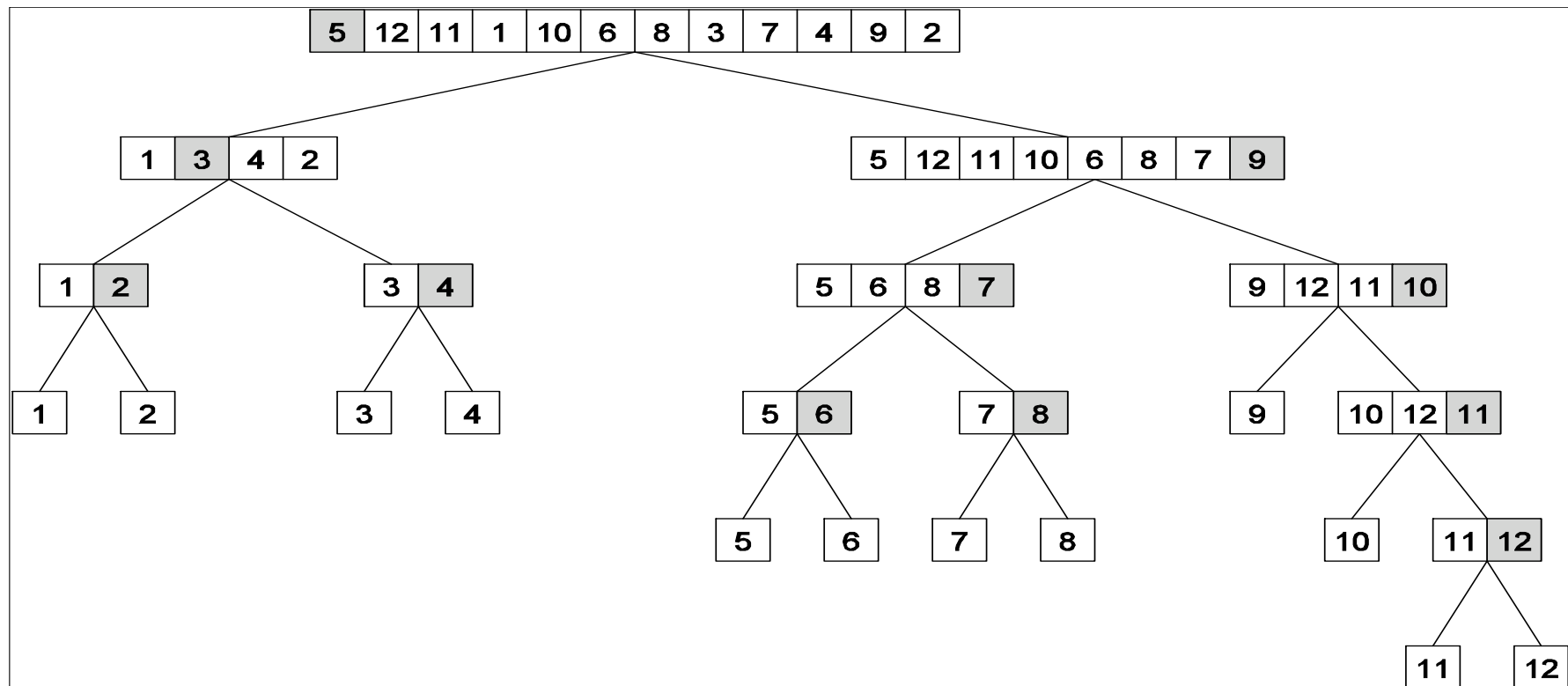
Recursive Decomposition for Min

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;
```
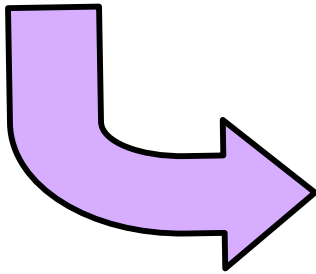
```
procedure RECURSIVE_MIN (A, n)
begin
  if ( n = 1 ) then
    min := A[0];
  else
    lmin := spawn RECURSIVE_MIN(&A[0], n/2 );
    rmin := spawn RECURSIVE_MIN(&A[n/2], n-n/2);
    if (lmin < rmin) then
      min := lmin;
    else
      min := rmin;
  return min;
```
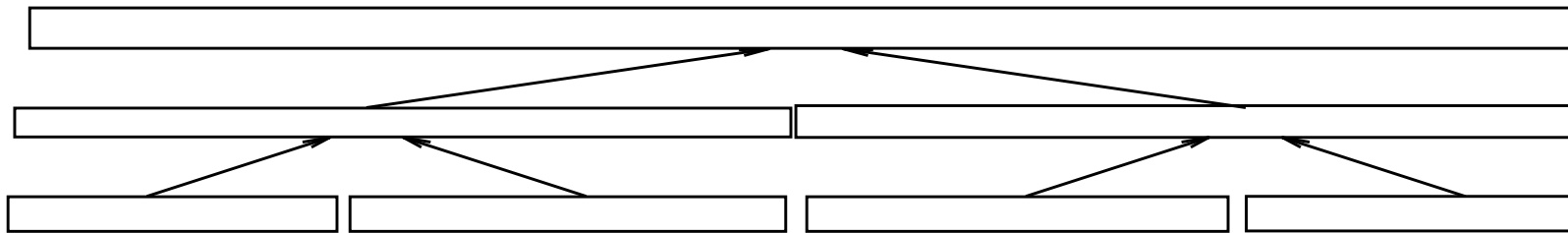
Applicable to other associative operations, e.g. sum, AND …

27

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

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

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

30

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**
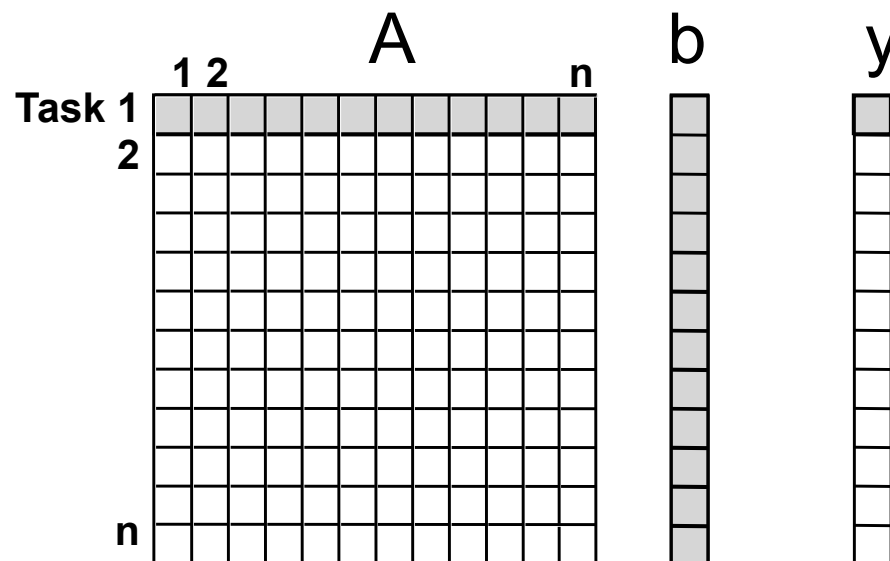
Output Data Decomposition: Example

- **Matrix multiplication: C = A x B**

- **Computation of C can be partitioned into four tasks**

$$\left(\begin{array}{cc} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{array} \right) \cdot \left(\begin{array}{cc} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{array} \right) \rightarrow \left(\begin{array}{cc} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{array} \right)$$

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

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

  - **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**

Partitioning both transactions and frequencies among the tasks
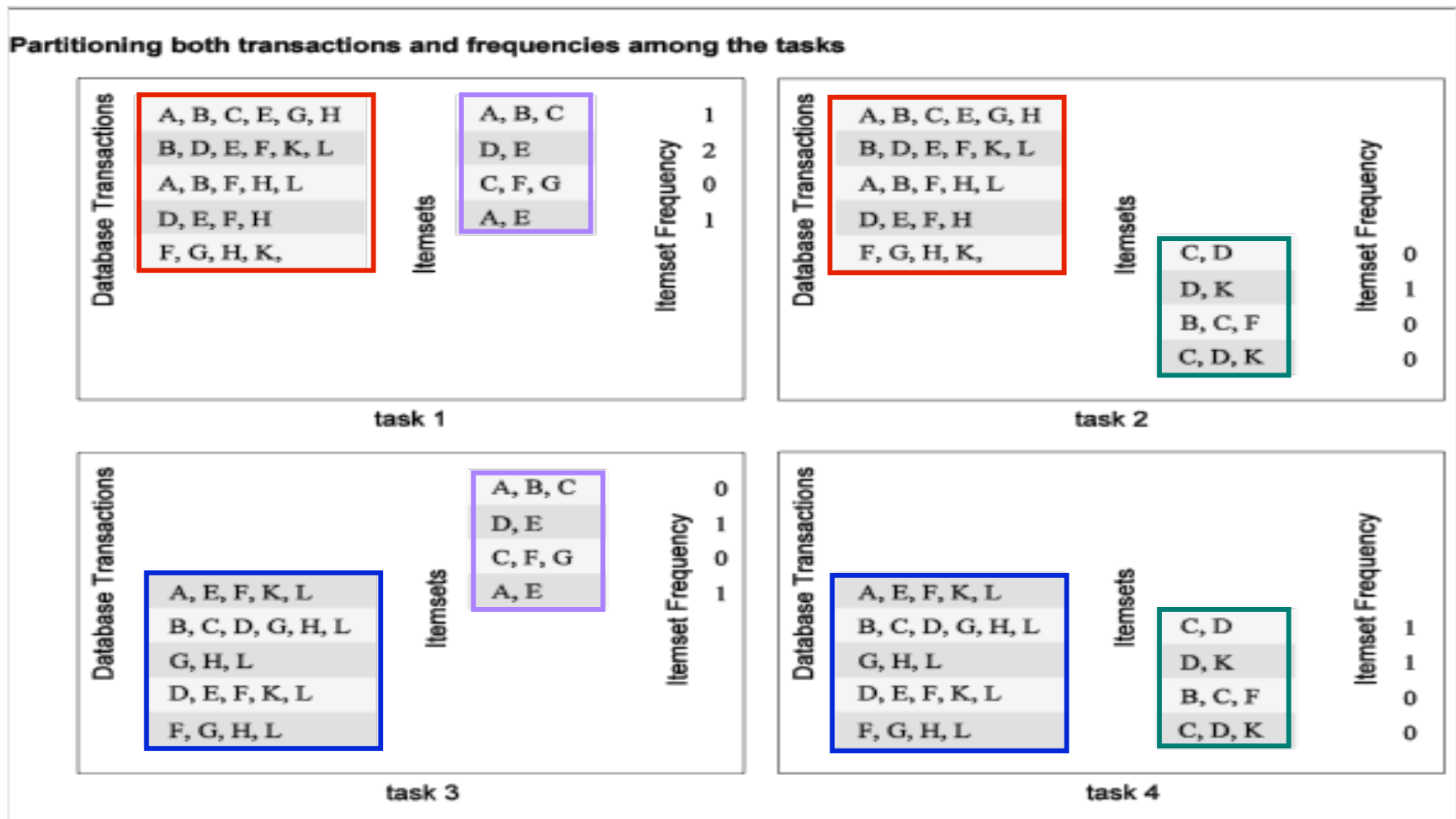
Intermediate Data Partitioning

- **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

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

Stage II

$$\left(\begin{array}{cc} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{array} \right) + \left(\begin{array}{cc} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,2} & D_{2,2,2} \end{array} \right) \rightarrow \left(\begin{array}{cc} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{array} \right)$$

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

Task dependency graph

37
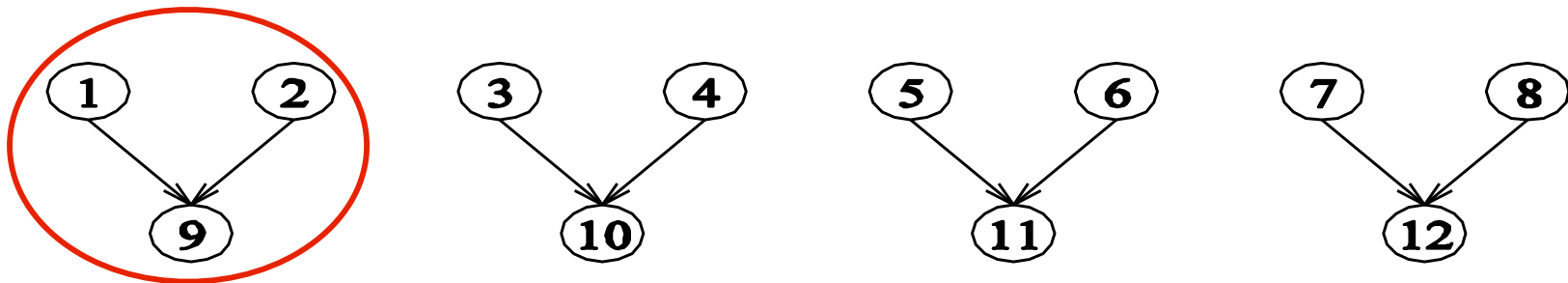
Owner Computes Rule

- **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 Next Class

- **Decomposition techniques - part 2**
  - —**exploratory decomposition**
  - —**hybrid decomposition**

- **Characteristics of tasks and interactions**
  - —**task generation, granularity, and context**
  - —**characteristics of task interactions**

- **Mapping techniques for load balancing**
  - —**static mappings**
  - —**dynamic mappings**

- **Methods for minimizing interaction overheads**

- **Parallel algorithm design templates**

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**