Parallel and Distributed Computing

CS3006

Lecture 5

Parallel Algorithm Design Life Cycle

19th February 2024

Dr. Rana Asif Rehman

Agenda

- A Quick Review
- 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
- Task-interaction Diagrams
  - Processes and mapping

Quick Review to the Previous Lecture

- Static vs Dynamic Interconnections
- Static Network Topologies
  - Linear array
  - Star
  - Mesh
  - Tree
  - Fully-connected
  - Hypercube
- Evaluating Static interconnections
  - Cost
  - Diameter
  - Bisection-width
  - Arc-connectivity

Principles of Parallel Algorithm Design

Principles of Parallel Algorithm Design

Steps in Parallel Algorithm Design

1. **Identification:** Identifying portions of the work that can be performed concurrently.
   - Work-units are also known as tasks
   - E.g., Initializing two mega-arrays are two tasks and can be performed in parallel
2. **Mapping:** The process of mapping concurrent pieces of the work or tasks onto multiple processes running in parallel.
   - Multiple processes can be physically mapped on a single processor.

Principles of Parallel Algorithm Design

Steps in Parallel Algorithm Design

3. **Data Partitioning:** Distributing the input, output, and intermediate data associated with the program.
   - One way is to copy whole data at each processing node
     - Memory challenges for huge-size problems
   - Other way is to give fragments of data to each processing node
     - Communication overheads
4. **Defining Access Protocol**: Managing accesses to data shared by multiple processors (i.e., managing communication).
5. **Synchronizing** the processors at various stages of the parallel program execution.

Principles of Parallel Algorithm Design

- **Decomposition:**
  - The process of dividing a computation into smaller parts, some or all of which may potentially be executed in parallel.
- ***Tasks***
  - **Programmer-defined units of computation** into which the main computation is subdivided by means of decomposition
  - Tasks can be of **arbitrary size**, but once defined, they are regarded as **indivisible units of computation**.
  - The tasks into which a problem is decomposed may not all be of the same size
  - Simultaneous execution of multiple tasks is the key to reducing the time required to solve the entire problem.

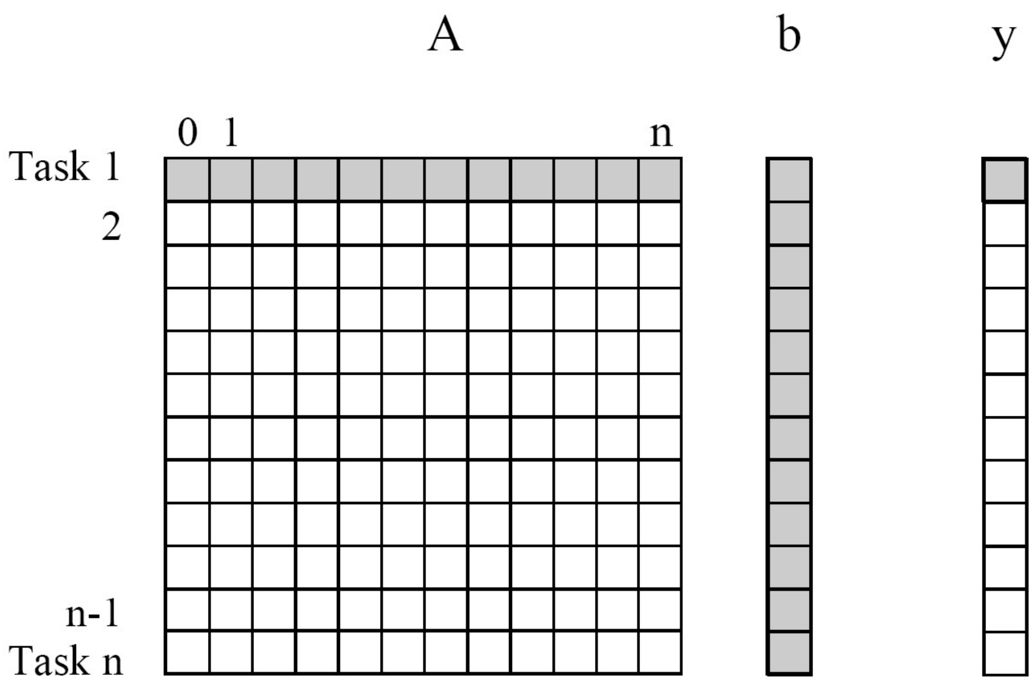


Figure 3.1 Decomposition of dense matrix-vector multiplication into n tasks, where n is the number of rows in the matrix. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

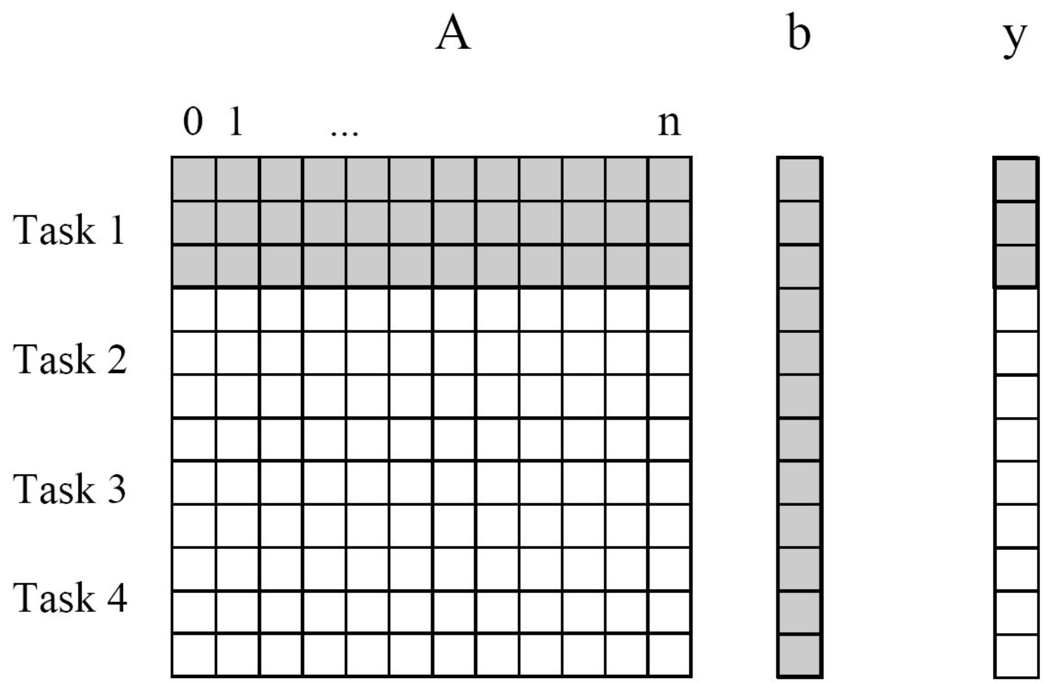


Figure 3.4 Decomposition of dense matrix-vector multiplication into four tasks. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

Principles of Parallel Algorithm Design

Granularity

- The **number and sizes** of tasks into which a problem is decomposed determines the ***granularity*** of the decomposition
  - A decomposition into a large number of small tasks is called ***fine-grained***
  - A decomposition into a small number of large tasks is called ***coarse-grained***
- For matrix-vector multiplication Figure 3.1 would usually be considered fine-grained
- Figure 3.4 shows a coarse-grained decomposition as each tasks computes $n/4$ of the entries of the output vector of length n

Principles of Parallel Algorithm Design

Task-Dependency Graph

- The tasks in the previous examples are independent and can be performed in any sequence.
- In most of the problems, there exist some sort of dependencies between the tasks.
- An abstraction used to express such **dependencies** among tasks and their **relative order of execution** is known as a **task-dependency graph**
- It is a **directed acyclic graph** in which node are tasks and the directed edges indicate the dependencies between them
- The task corresponding to a node can be executed when all tasks connected to this node by incoming edges have completed.

Select ID, model, year, color from dbtable where model=‘civic’ and year=‘2001’ and (color=‘White’ OR color=‘Green’)

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

Table 3.1 A database storing information about used vehicles.

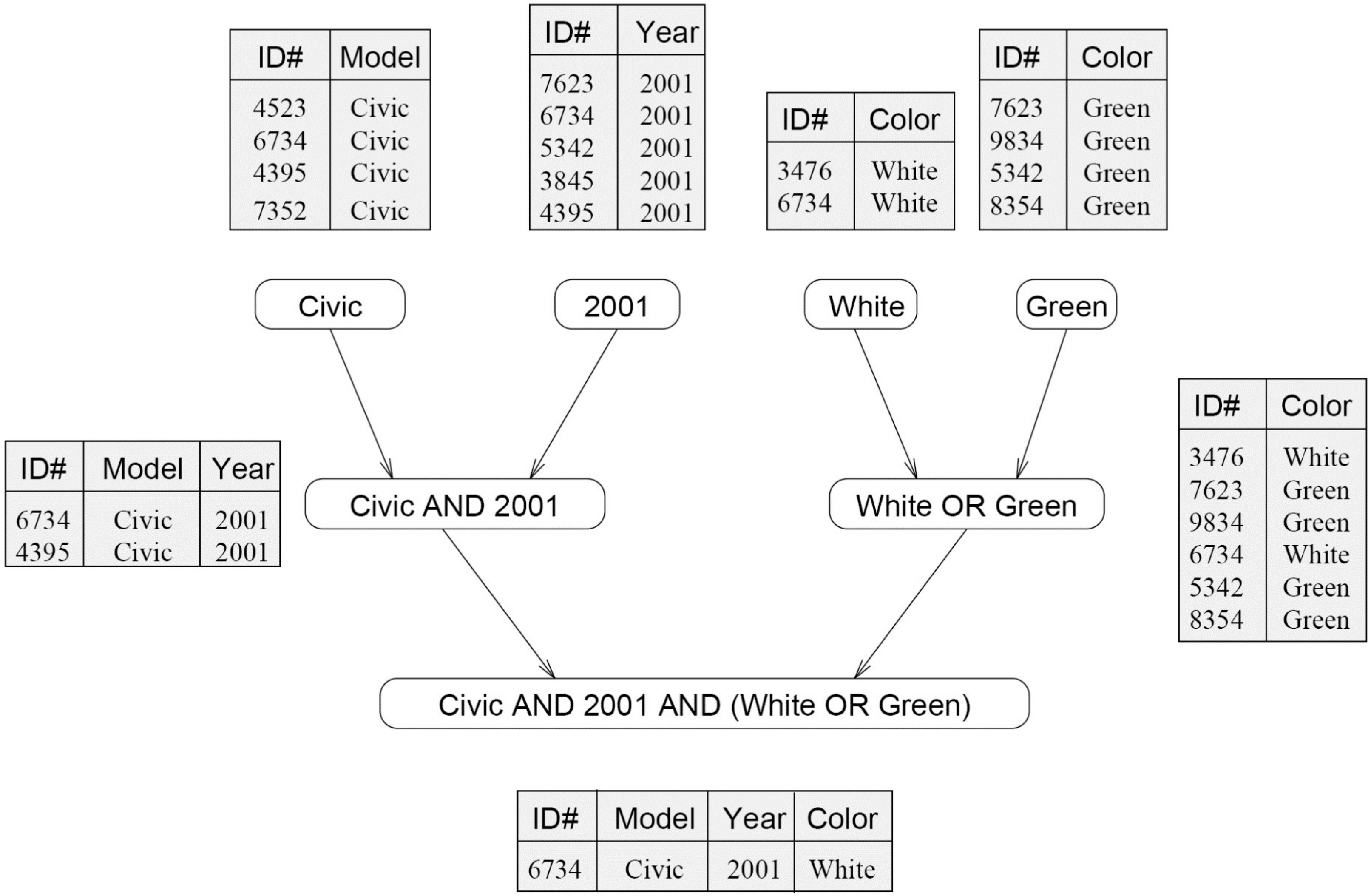


Figure 3.2 The different tables and their dependencies in a query processing operation.

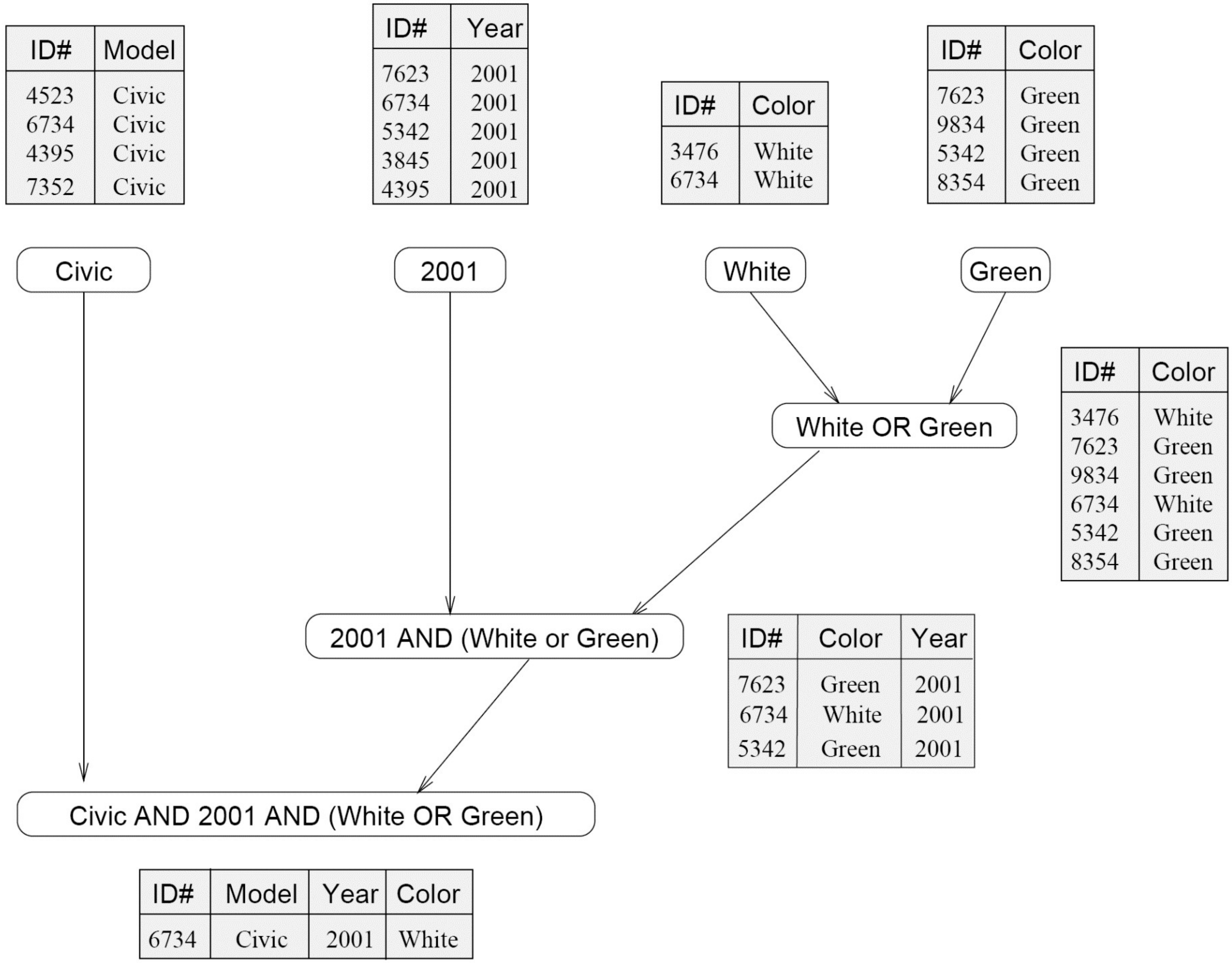


Figure 3.3 An alternate data-dependency graph for the query processing operation.

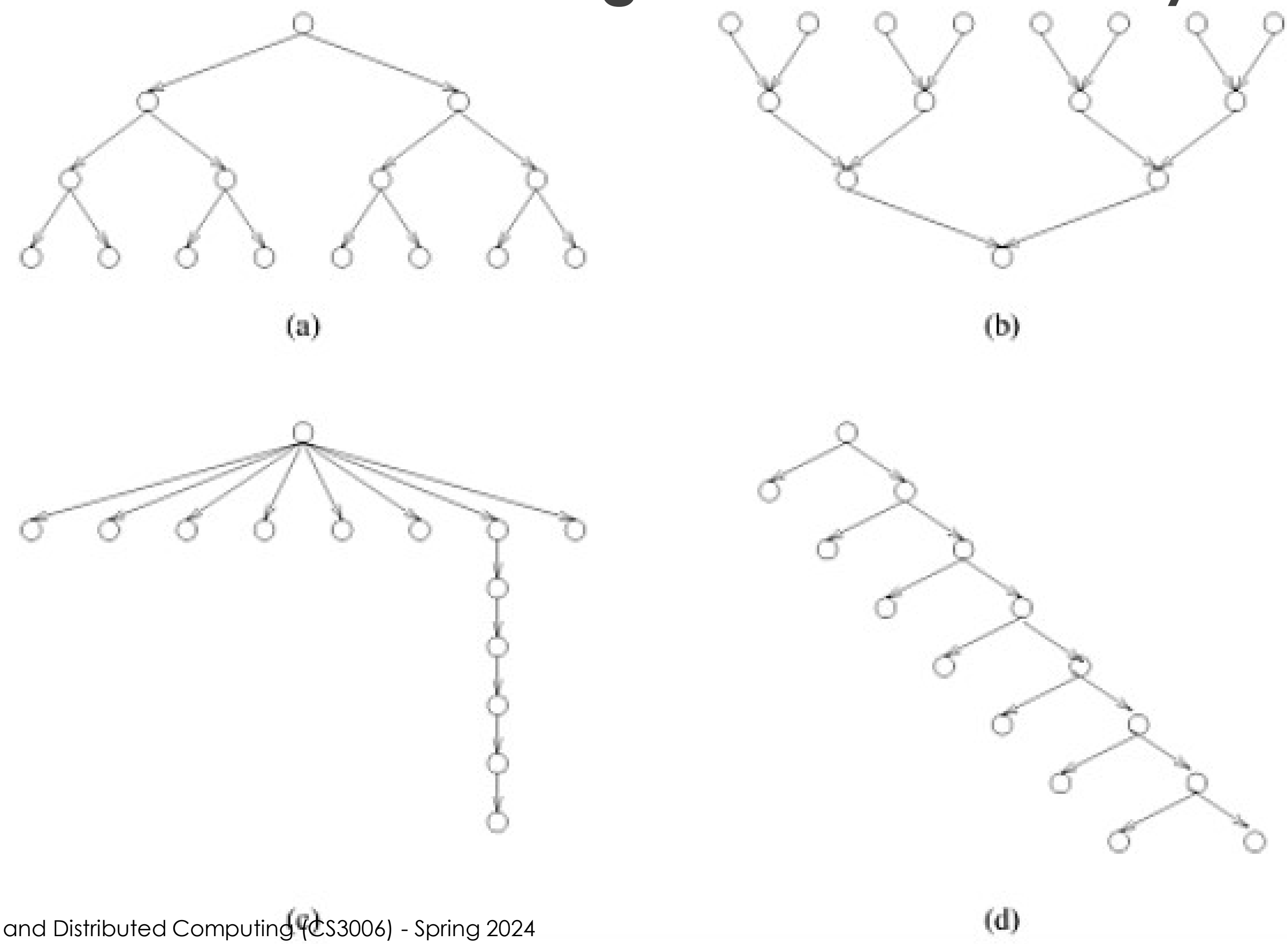
principles of Parallel Algorithm Design

Maximum Degree of Concurrency

- The maximum number of tasks that can be executed simultaneously in a parallel program at any given time is known as its *maximum degree of concurrency*
- Usually, it is always less than total number of tasks due to dependencies.
- E.g., max-degree of concurrency in the task-graphs of Figures 3.2 and 3.3 is 4.
- **Rule of thumb:** For task-dependency graphs that are trees, the maximum degree of concurrency **is always equal to the number of leaves in the tree**

principles of Parallel Algorithm Design

Determine Maximum Degree of Concurrency?



principles of Parallel Algorithm Design

Average Degree of Concurrency

- A relatively better measure for the performance of a parallel program
- The average number of tasks that can run concurrently over the entire duration of execution of the program
- The ratio of the ***total amount of work*** to the ***critical-path length***
  - So, what is the critical path in the graph**?**

principles of Parallel Algorithm Design

Average Degree of Concurrency

- **Critical Path:** The longest directed path between any pair of start and finish nodes is known as the critical path.
- **Critical Path Length:** The sum of the weights of nodes along this path
  - the weight of a node is the size or the amount of work associated with the corresponding task.
- A shorter critical path favors a higher average-degree of concurrency.
- Both, maximum and average degree of concurrency increases as tasks become smaller(finer)

principles of Parallel Algorithm Design

Average Degree of Concurrency

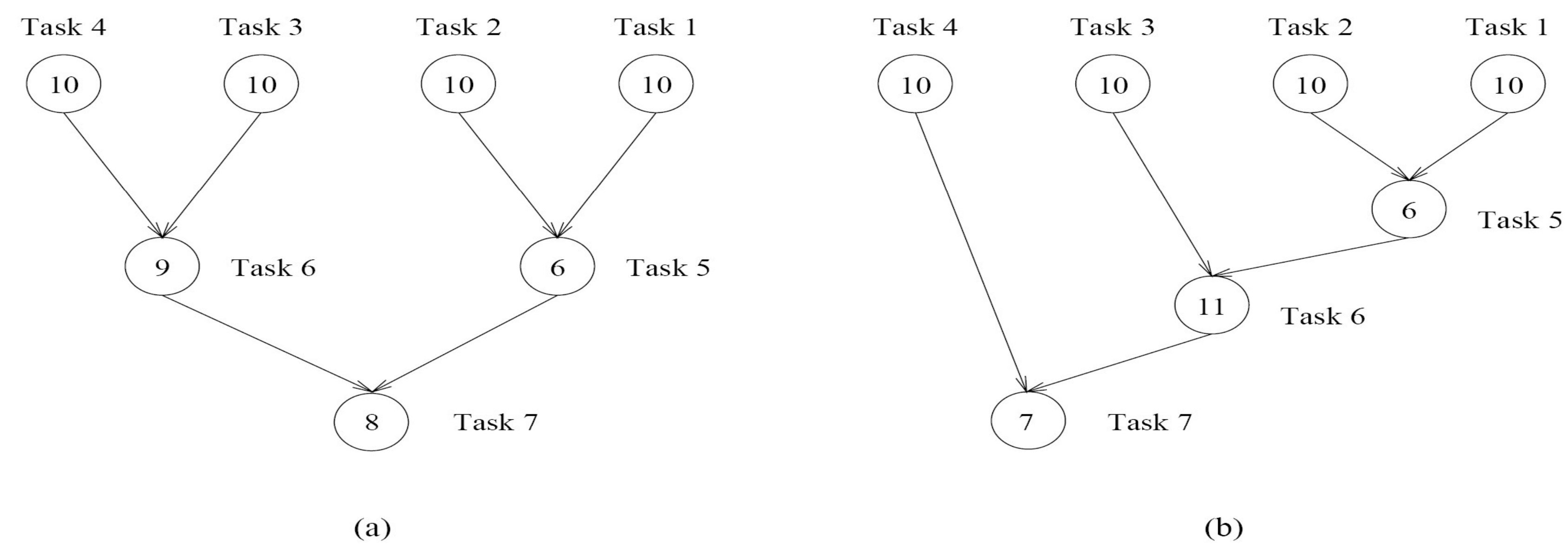


Figure 3.5 Abstractions of the task graphs of Figures 3.2 and 3.3, respectively.

Critical path lengths: 27 and 34

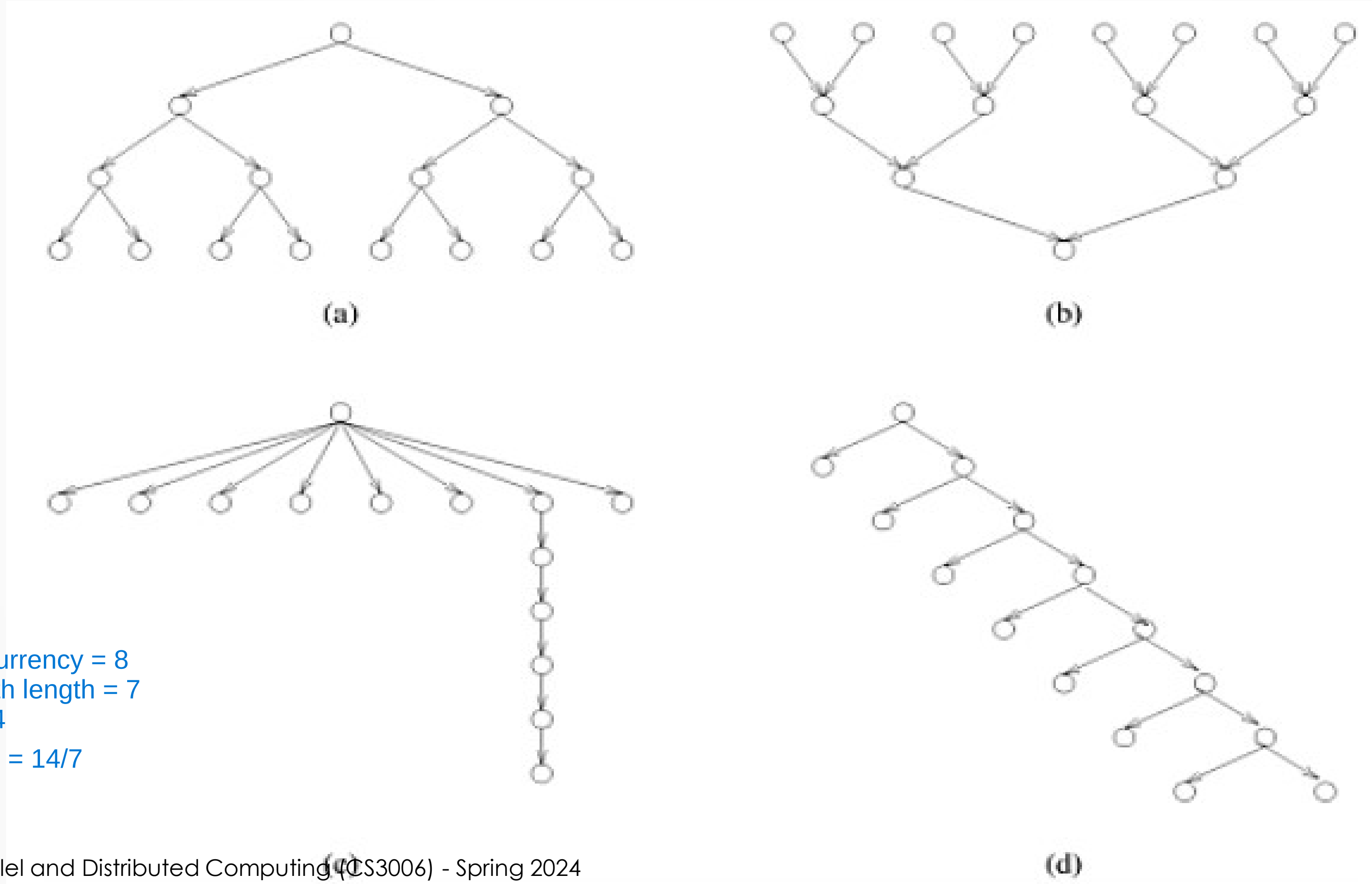
AVG DEG = 63/27 = 2.33

Total amount of work: 63 and 64

Average degree of concurrency: 2.33 and 1.88

principles of Parallel Algorithm Design

Determine critical path length and average-concurrency?



max concurrency = 8
critical path length = 7
total = 14
avg conc = 14/7

principles of Parallel Algorithm Design

Task Interact Graph

- Depicts pattern of interaction between the tasks
- Dependency graphs only show that how output of first task becomes input to the next level task.
- But how the tasks interact with each other to access distributed data is only depicted by task interaction graphs
- The nodes in a task-interaction graph represent tasks
- The edges connect tasks that interact with each other

principles of Parallel Algorithm Design

Task Interact Graph

- The edges in a task interaction graph are usually **undirected**
  - but directed edges can be used to indicate the direction of flow of data, if it is unidirectional.
- The edge-set of a task-interaction graph is usually a **superset** of the edge-set of the task-dependency graph
- In database query processing example, the task-interaction graph is the **same** as the task-dependency graph.

principles of Parallel Algorithm Design

Task Interact Graph (Sparse-matrix multiplication)

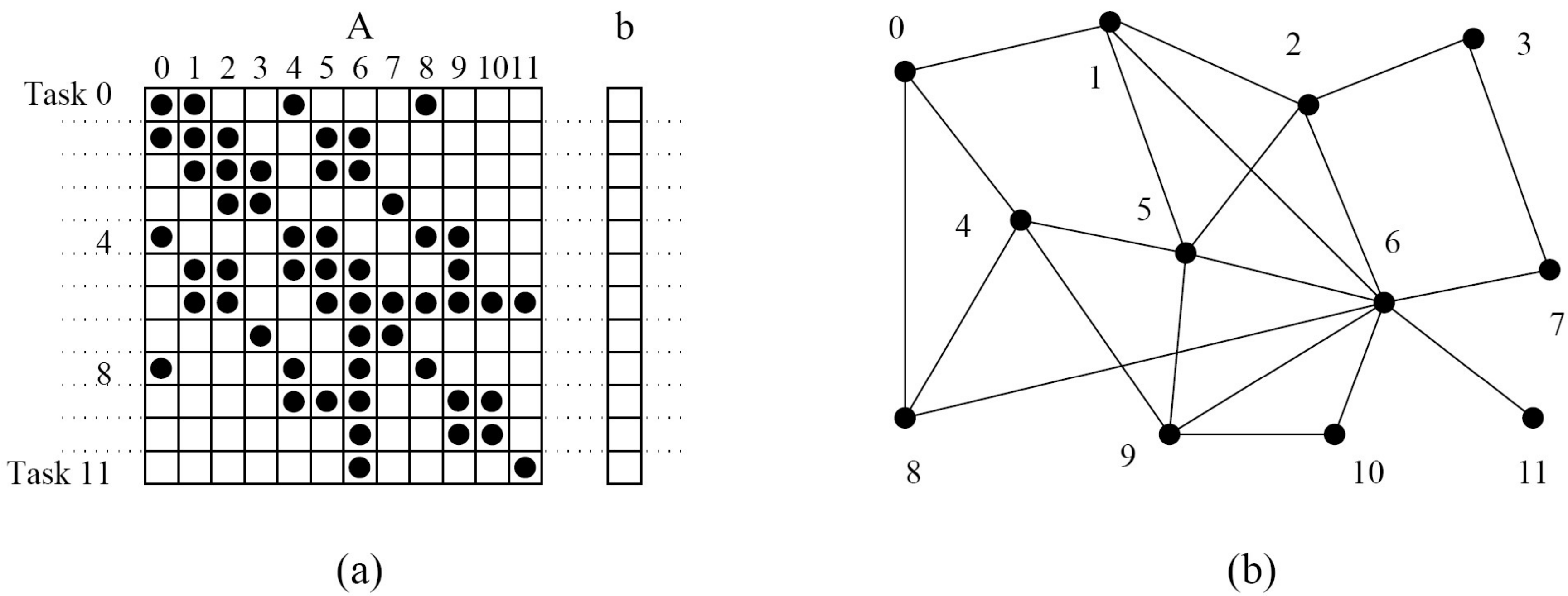


Figure 3.6 A decomposition for sparse matrix-vector multiplication and the corresponding task-interaction graph. In the decomposition Task i computes $\sum_{0 \le j \le 11, A[i,j] \ne 0} A[i, j].b[j]$.

principles of Parallel Algorithm Design

Processes and Mapping

- Logical processing or computing agent that performs tasks is called **process**.
- The mechanism by which tasks are assigned to processes for execution is called ***mapping.***
- Multiple tasks can be mapped on a single process
- Independent task should be mapped onto different processes
- Map tasks with high mutual-interactions onto a single process

principles of Parallel Algorithm Design

Processes and Mapping

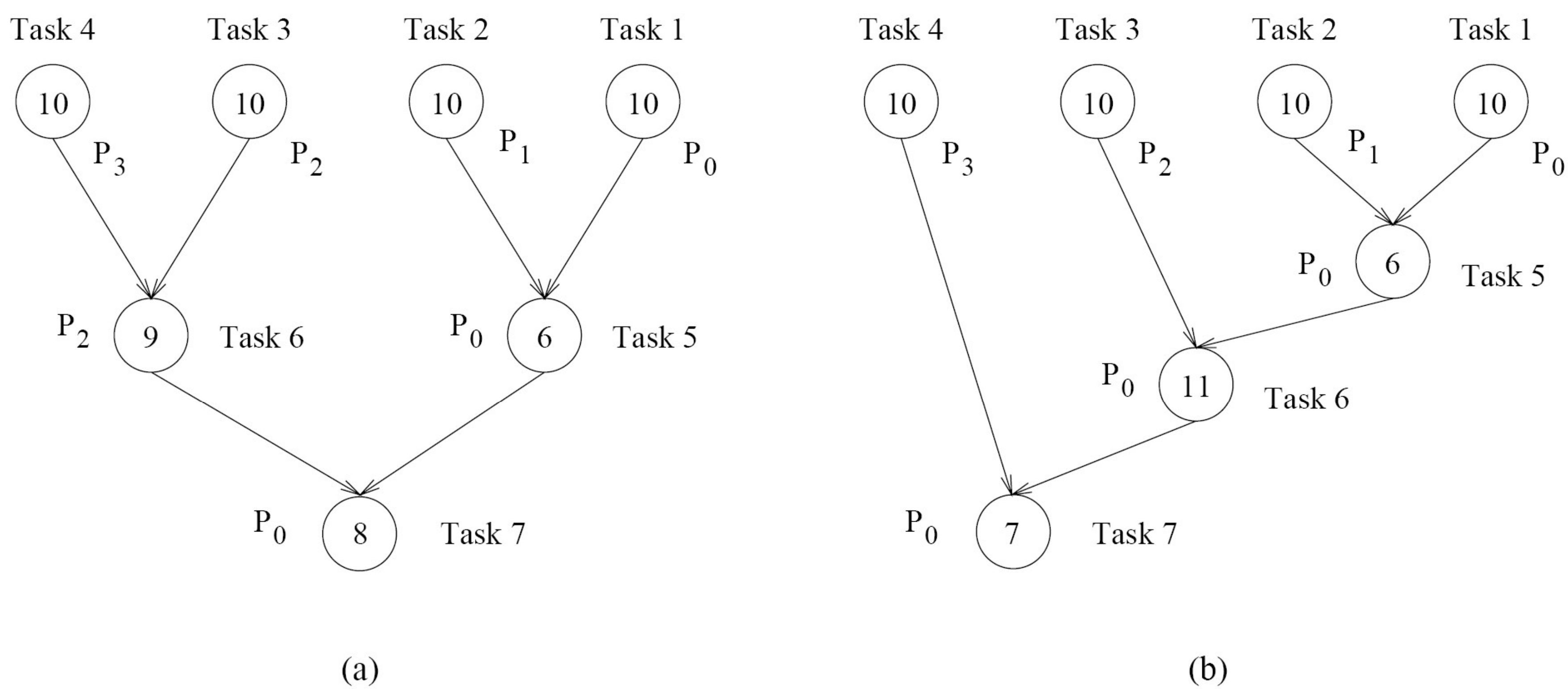


Figure 3.7 Mappings of the task graphs of Figure 3.5 onto four processes.

principles of Parallel Algorithm Design

Processes and Processors

- **Processes** are logical computing agents that perform tasks
- **Processors** are the hardware units that physically perform computations
- Depending on the problem, multiple processes can be mapped on a single processor
- But, in most of the cases, there is one-to-one correspondence between processors and processes
- So, we assume that there are as many processes as the number of physical CPUs on the parallel computer

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).