Parallel and Distributed Computing
CS3006 (BDS-6A)
Lecture 06

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science, FAST

14 February, 2023

Previous Lecture

- Network Topologies:
  - Linear array (with or without wraparound links)
  - K-d meshes
  - Hypercubes
  - Tree-based networks (fat-trees or otherwise, static or dynamic)
- Evaluating static interconnections
  - Cost, diameter, bisection width, arc connectivity

Evaluating Static Interconnections

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	$p-1$	$p(p-1)/2$
Star	2	1	1	$p-1$
Complete binary tree	$2\log((p+1)/2)$	1	1	$p-1$
Linear array	$p-1$	1	1	$p-1$
2-D mesh, no wraparound	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2-D wraparound mesh	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$
Hypercube	$\log p$	$p/2$	$\log p$	$(p\log p)/2$

Parallel Algorithm Design Life Cycle

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 & synchronization).

Parallel computing Examples - Chess Player

- A parallel program to play chess might look at all the possible first moves it could make
- Each different first move could be explored by a different processor, to see how the game would continue from that point
- Results have to be combined to figure out which is the best first move
- The famous IBM Deep Blue machine that beat Kasparov
- Brute force computing power, massively parallel with 30 nodes, with each node containing a 120 MHz P2SC microprocessor

Load Balance

- Inefficient if many processors are idle while one processor has lots of work to do and this slows down the whole application

- Best utilizations of parallel processors

- Require load balancing (parallel processors are typically symmetric)
- For example
  - Web Servers
  - Matrix Multiplication

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.

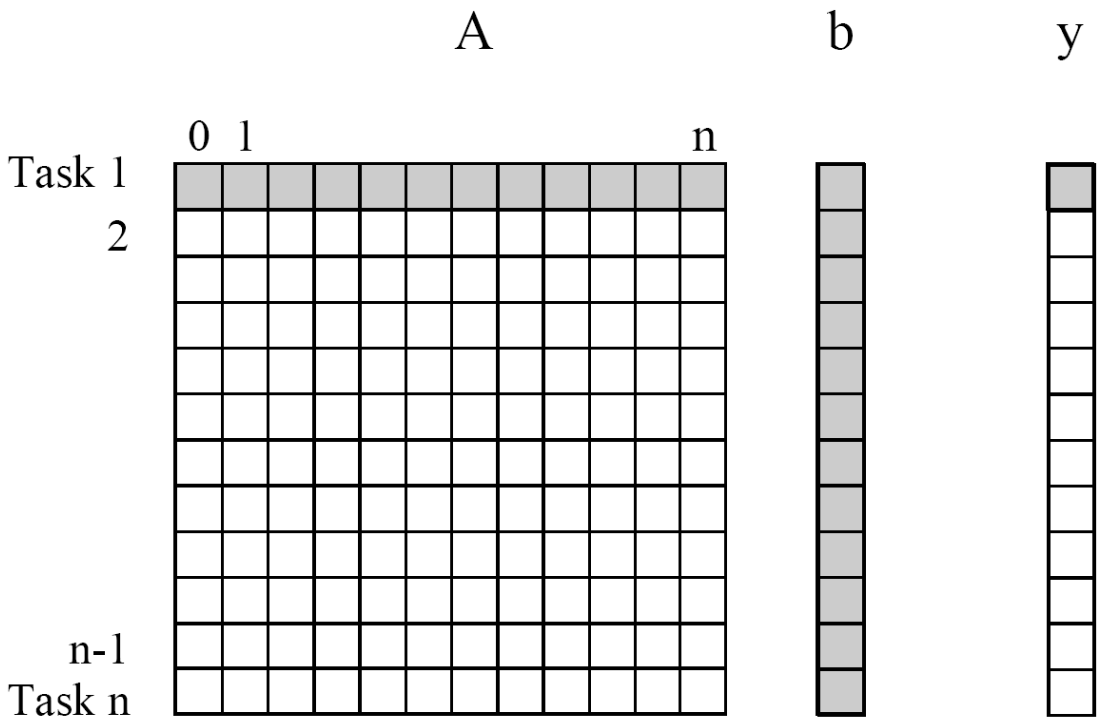


Figure 3.1 Decomposition of dense matrix-vector multiplication into n tasks, where n is the number of rows in the matrix. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

- *Problem can be decomposed into n tasks*
- *Computation of each element of vector y is independent of other elements*
- *No control dependencies so no task-dependency graph*

Vector Multiplication (n x 1)

- So the multiplication program like:

```
for (row = 0; row < n; row++)
  y[row] = dot_product( get_row(A, row), get_col(b));
```

  can be transformed to:

```
for (row = 0; row < n; row++)
  y[row]= create_thread( dot_product(get_row(A, row), get_col(b)));
```

- In this case, one may think of the thread as an instance of a function that returns before the function has finished executing

Vector Multiplication (n x n)

```
for (row = 0; row < n; row++)
    for (column = 0; column < n; column++)
        c[row][column] = dot_product( get_row(a, row), get_col(b, col));
```

Multithreaded:

```
for (row = 0; row < n; row++)
    for (column = 0; column < n; column++)
      c[row][column] = create_thread( dot_product(get_row(a, row), get_col(b, col)));
```

Task-Dependency Graph

- The tasks in the previous examples are independent and can be performed in any sequence.
- In most of the problems, some sort of dependency exists 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 nodes 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.

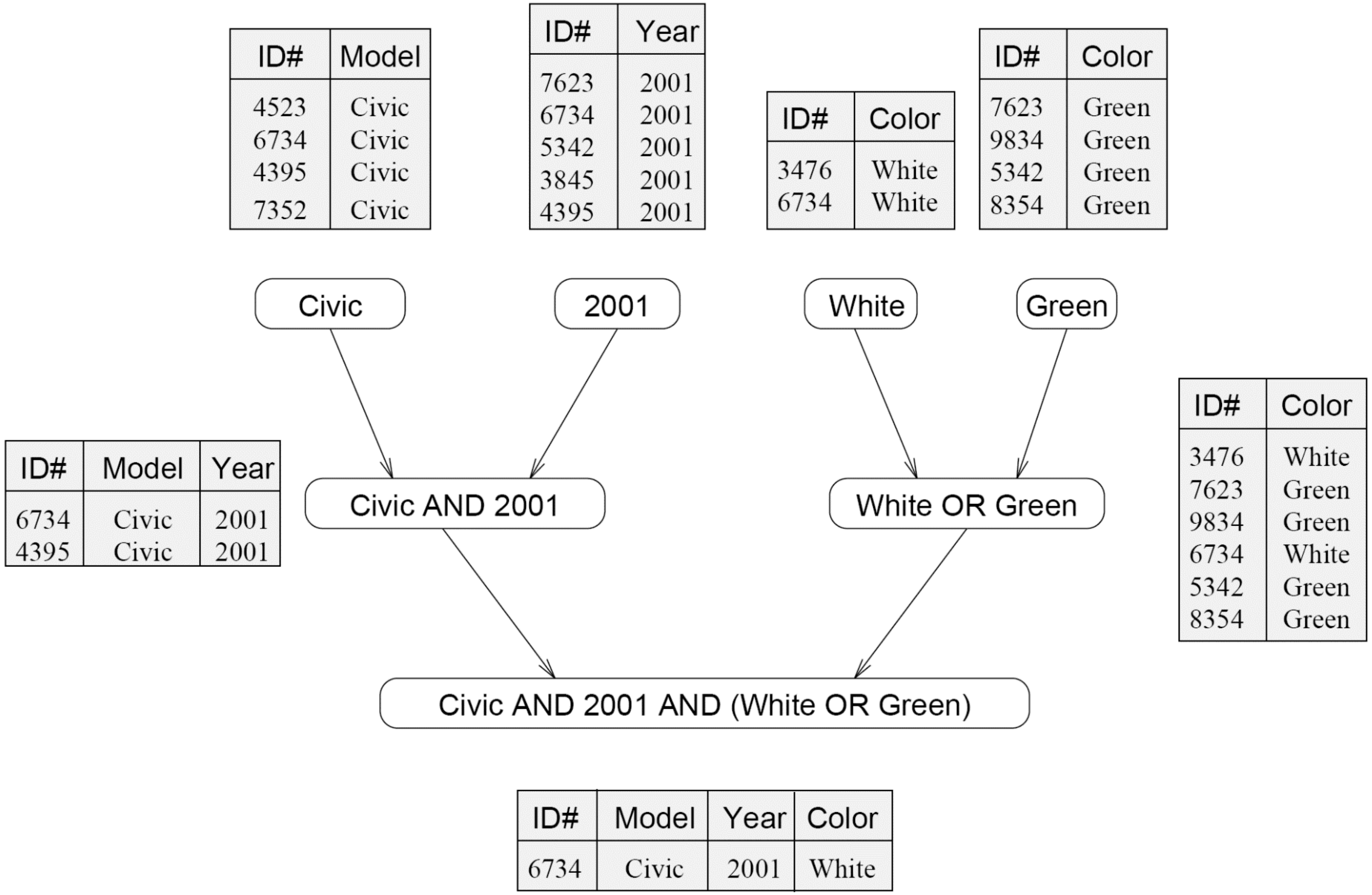
DB Query

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.

Execution of the 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

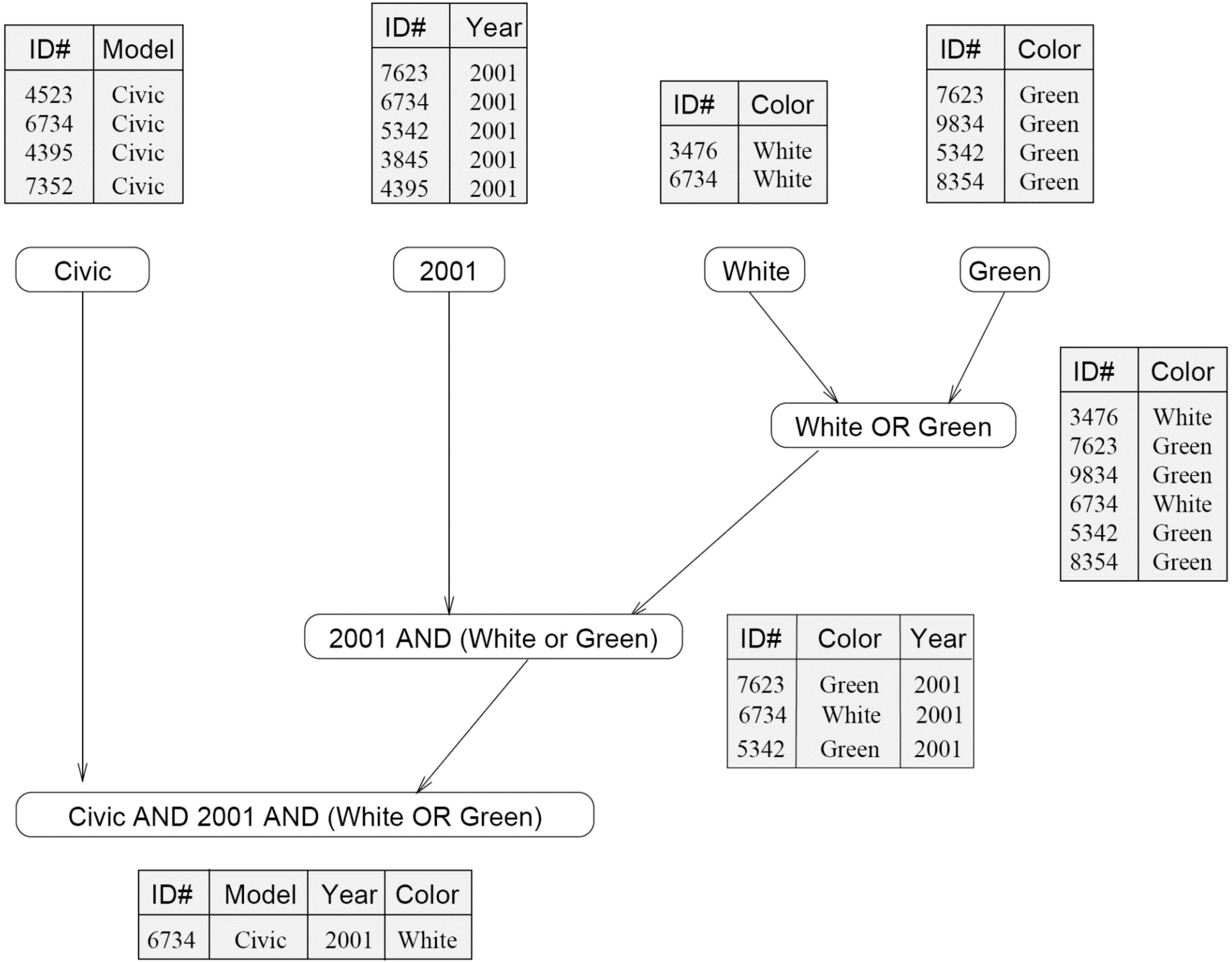
ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

ID#	Model	Year
6734	Civic	2001
4395	Civic	2001

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

ID#	Model	Year	Color
6734	Civic	2001	White

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



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

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

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

ID#	Model	Year	Color
6734	Civic	2001	White

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

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 task computes $n/4$ of the entries of the output vector of length n

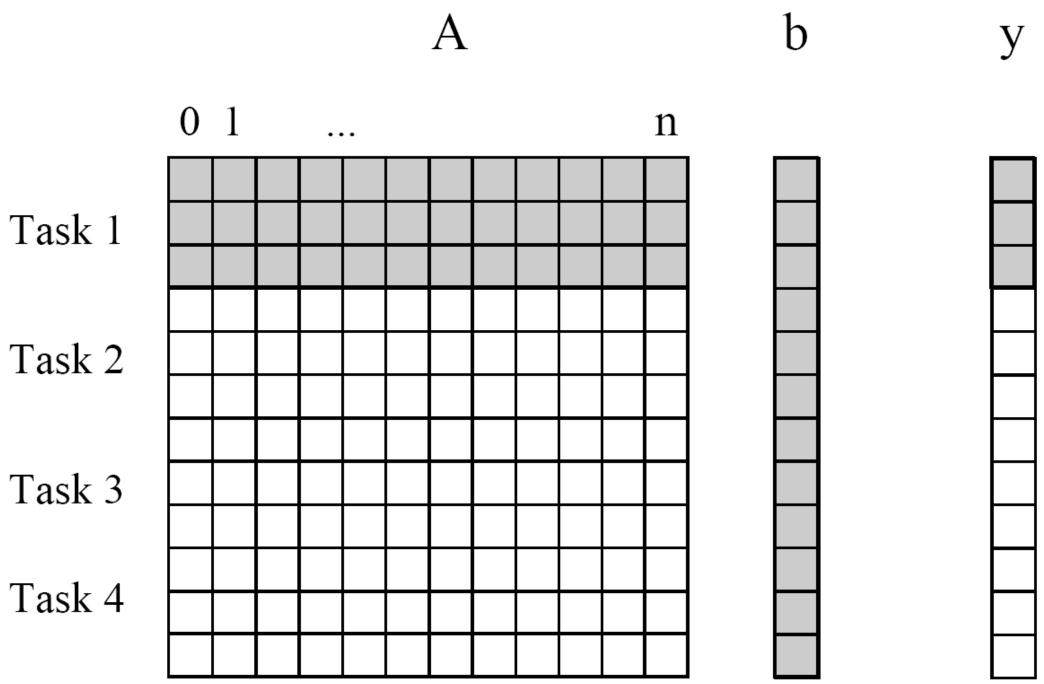


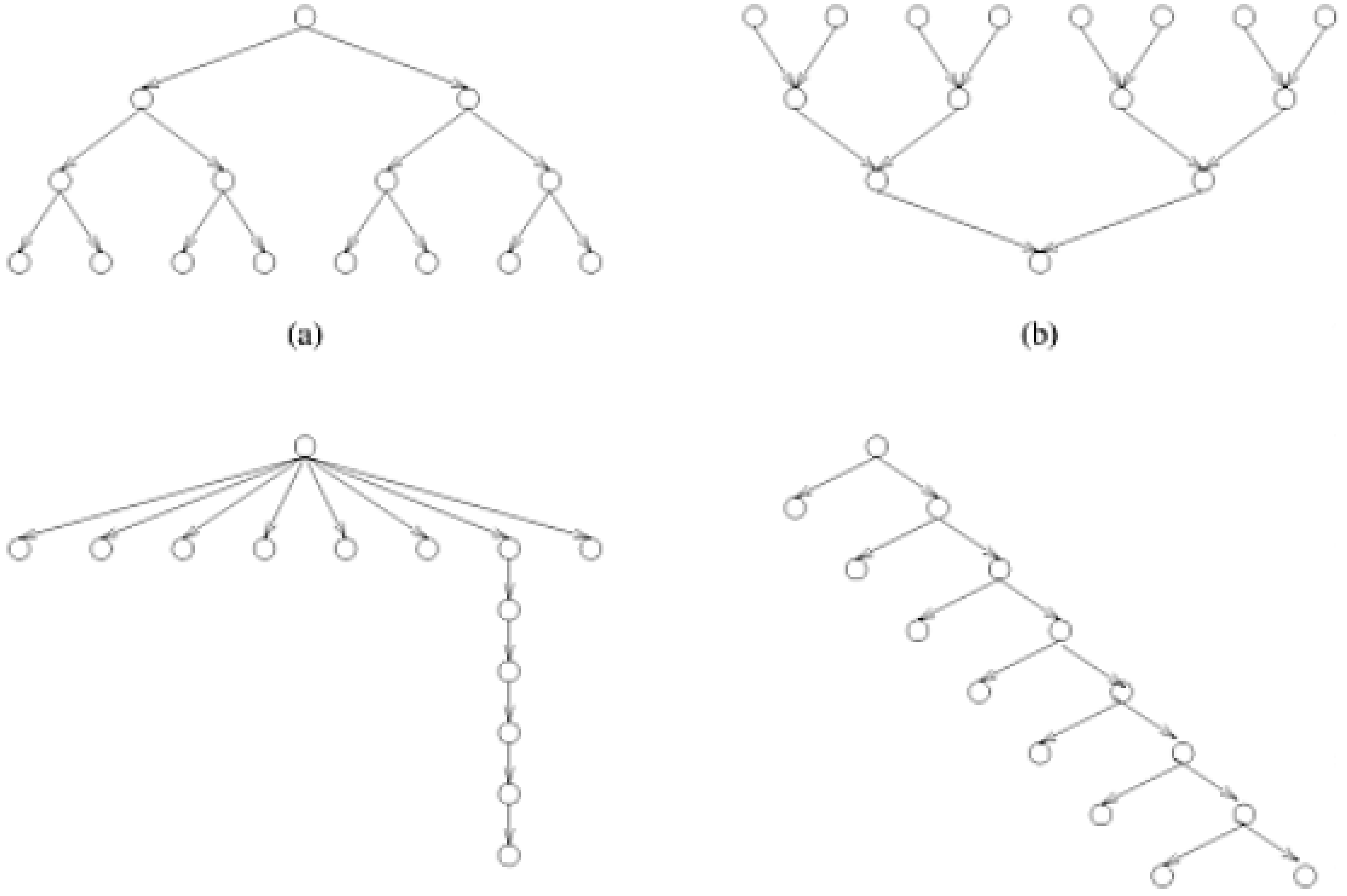
Figure 3.4 Decomposition of dense matrix-vector multiplication into four tasks. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

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***

Maximum Degree of Concurrency

Determine the Maximum Degree of Concurrency?



(a) (b) (c) (d)

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**?**

Critical Path

- ***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)

Average Degree of Concurrency

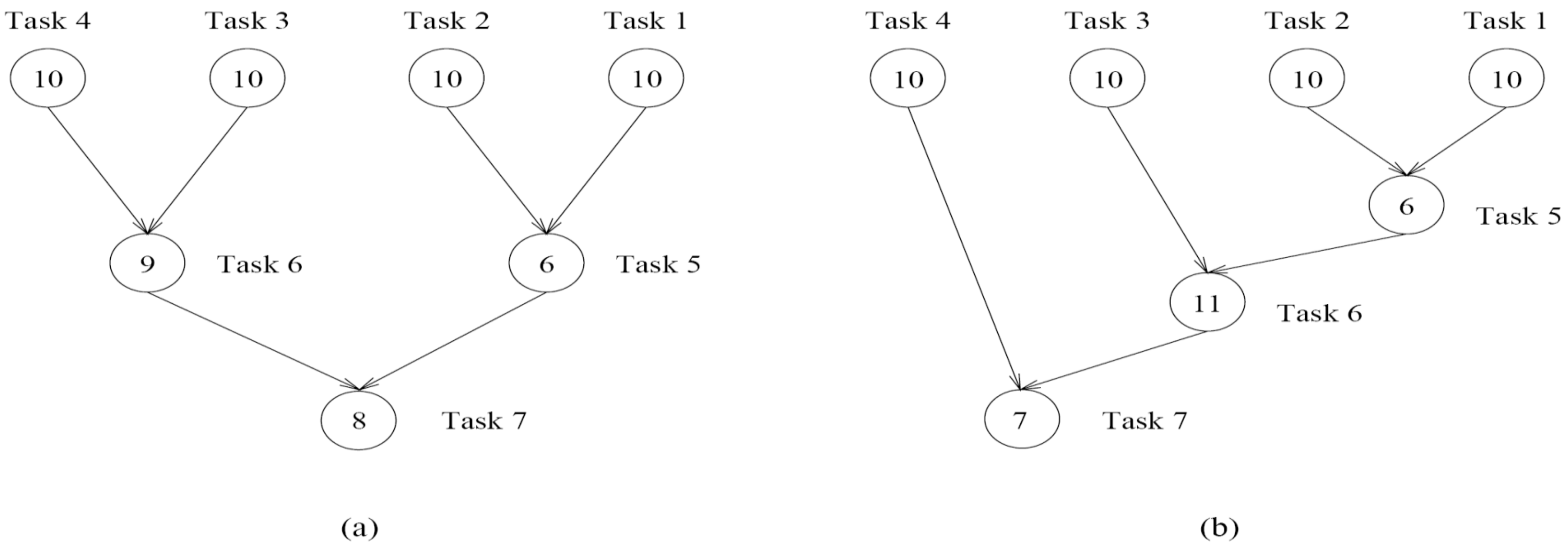


Figure 3.5 Abstractions of the task graphs of Figures 3.2 and 3.3, respectively.

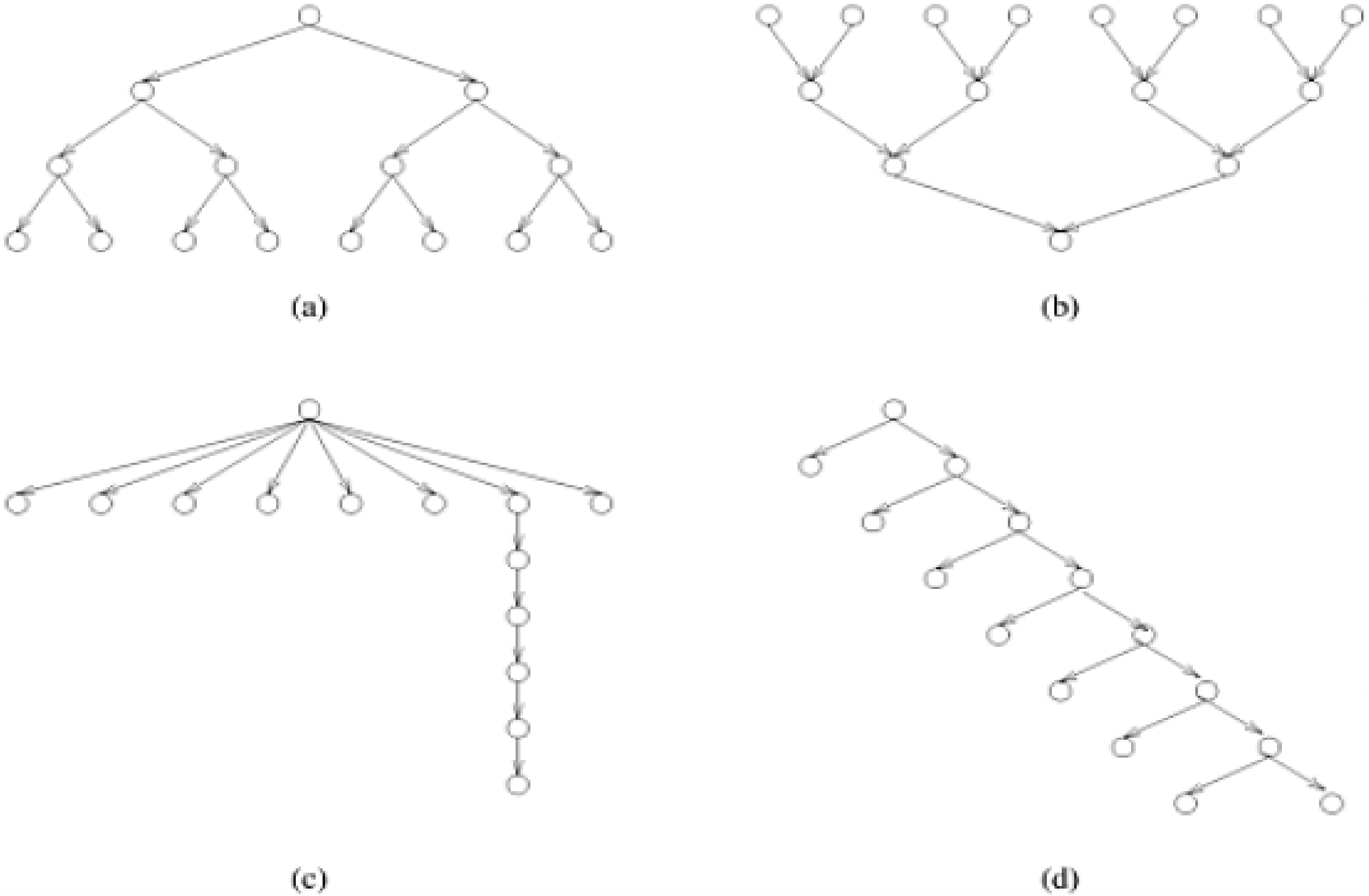
Critical path lengths: 27 and 34

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?



(a) (b) (c) (d)

Task Interaction Graph

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

Task Interaction 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 the 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**)**

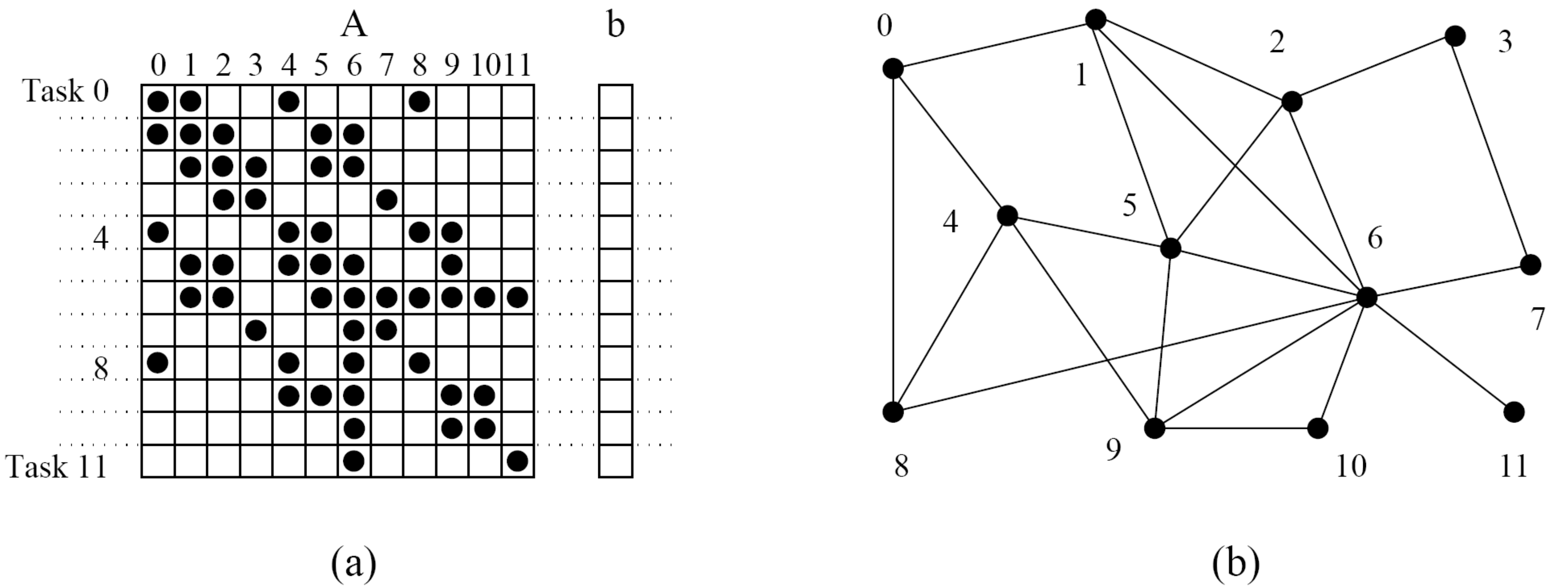


Figure 3.6 A decomposition for sparse matrix-vector multiplication and the corresponding task-interaction graph. In the decomposition Task i computes $\sum_{0 \le j \le 11, A[i,j] \ne 0} A[i, j].b[j]$.

Sources

- Slides of Dr. Rana Asif Rahman & Dr. Haroon Mahmood, FAST
- (Chapter 2) Kumar, V., Grama, A., Gupta, A., & Karypis, G. (1994). Introduction to parallel computing (Vol. 110). Redwood City, CA: Benjamin/Cummings.
- Quinn, M. J. Parallel Programming in C with MPI and OpenMP,(2003).