



Parallel and Distributed Computing

CS3006

Week No. 4, Lesson No. 1

Wrapping-up Interconnects
Parallel Algorithm Design Life Cycle

February 12, 2024


Original slides by: Dr. Rana Asif Rehman
Additions for Spring 2024 by: Dr. Abdul Qadeer

Agenda

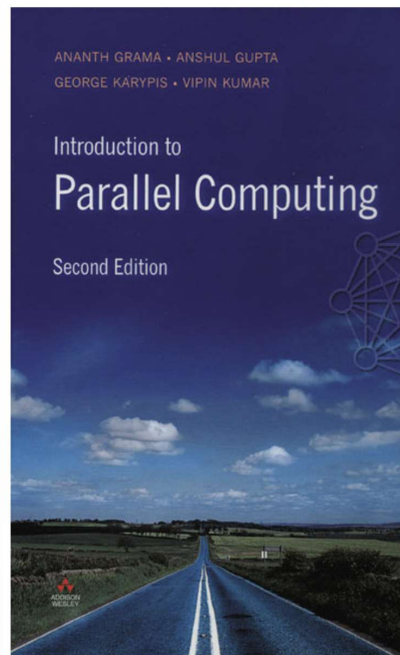


- A Quick Review
- Latencies of messages
- Parallel Algorithm Design Life Cycle
- Tasks, Decomposition, and Task-dependency graphs
- Granularity
 - Fine-grained
 - Coarse-grained
- Concurrency
 - Max-degree of concurrency
 - Critical path length
 - Average-degree of concurrency
- Task-interaction Diagrams
 - Processes and mapping

Quick Review to the Previous Lecture

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

Communication Costs in Parallel Machines (Textbook section: 2.5)



Challenge: Interconnect Diversity

- How to write programs for a hardware with diverse interconnects (Bus, crossbar, mesh, ...)?
 - Ignore diversity and do your best?
 - Tweak program for each kind of interconnect?
- Theoretical cost models
 - Latency components from message size and startup time bigger than hop counts-based latency
- Practice
 - Start with a reasonable program and empirically see the performance numbers
 - Iterate for hardware-dependent optimizations if we don't meet the required performance target

Interconnection Network Performance

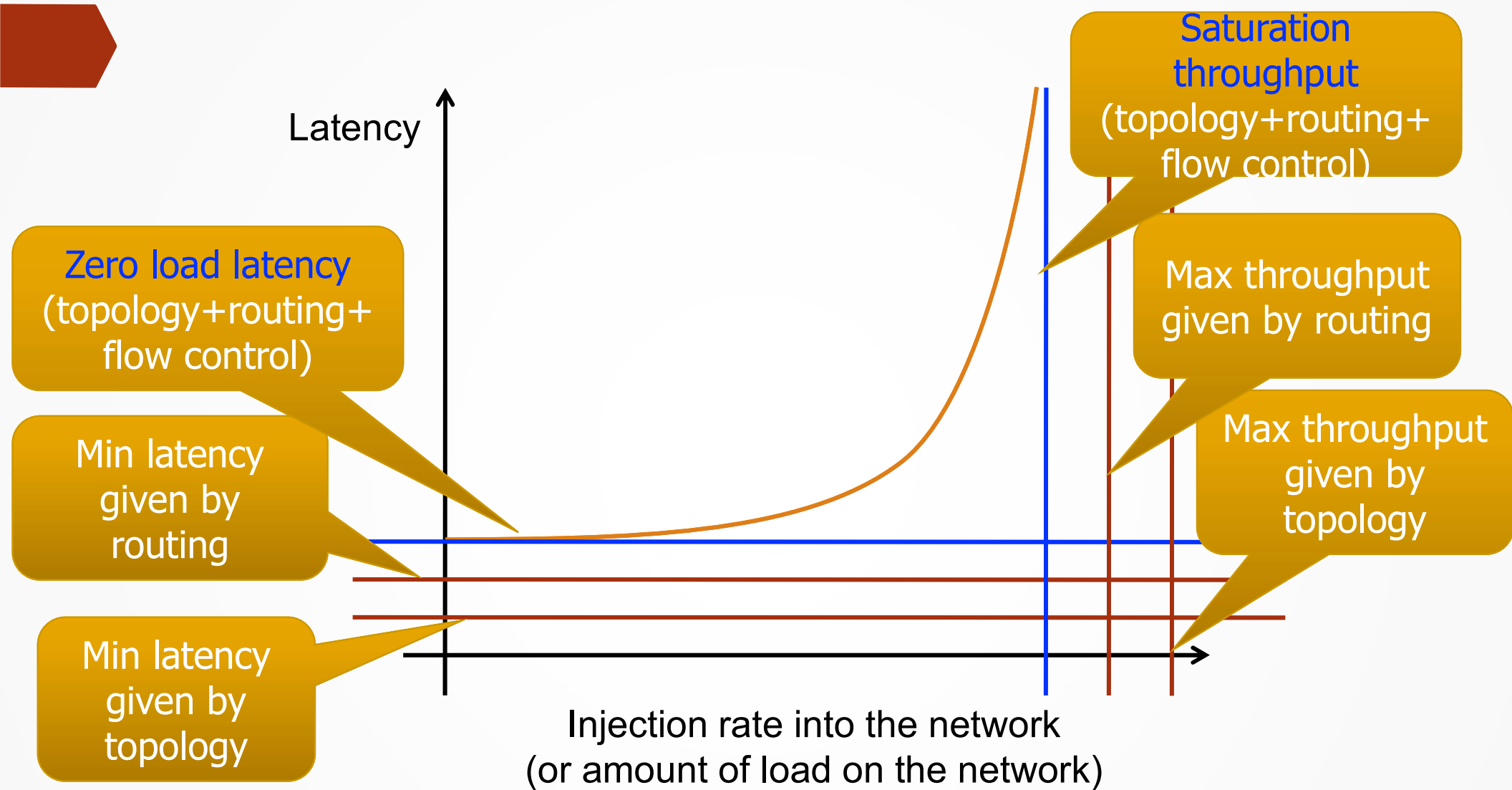
6

Credit: Next 5 slides taken from Prof. Onur Mutlu

See L20 and L21 at:

<https://safari.ethz.ch/architecture/fall2022/doku.php?id=schedule>

Interconnection Network Performance



Saturation throughput: Injection rate at which latency asymptotically approaches "Zero load" latency
"Zero load" latency: Latency with no contention

Ideal Latency

Ideal latency

- Solely due to wire delay between source and destination

$$T_{ideal} = \frac{D}{v} + \frac{L}{b}$$

- D = Manhattan distance
 - The distance between two points measured along axes at right angles.
- v = propagation velocity
- L = packet size
- b = channel bandwidth

Actual Latency

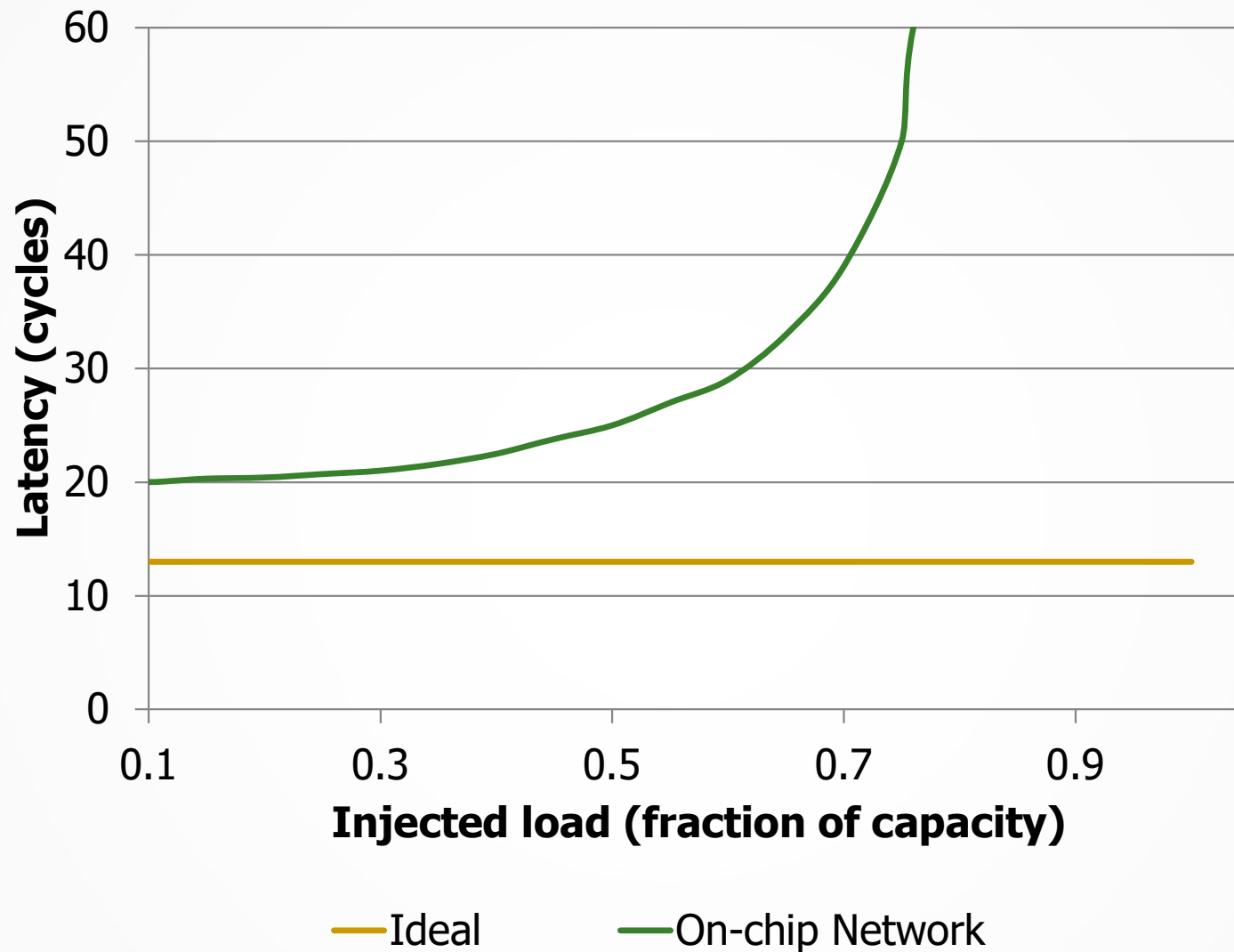
Dedicated wiring impractical

- Long wires segmented with insertion of routers

$$T_{actual} = \frac{D}{v} + \frac{L}{b} + H \cdot T_{router} + T_c$$

- D = Manhattan distance
- v = propagation velocity
- L = packet size
- b = channel bandwidth
- H = hops
- T_{router} = router latency
- T_c = latency due to contention

Load-Latency Curve



Load-Latency Curve Examples

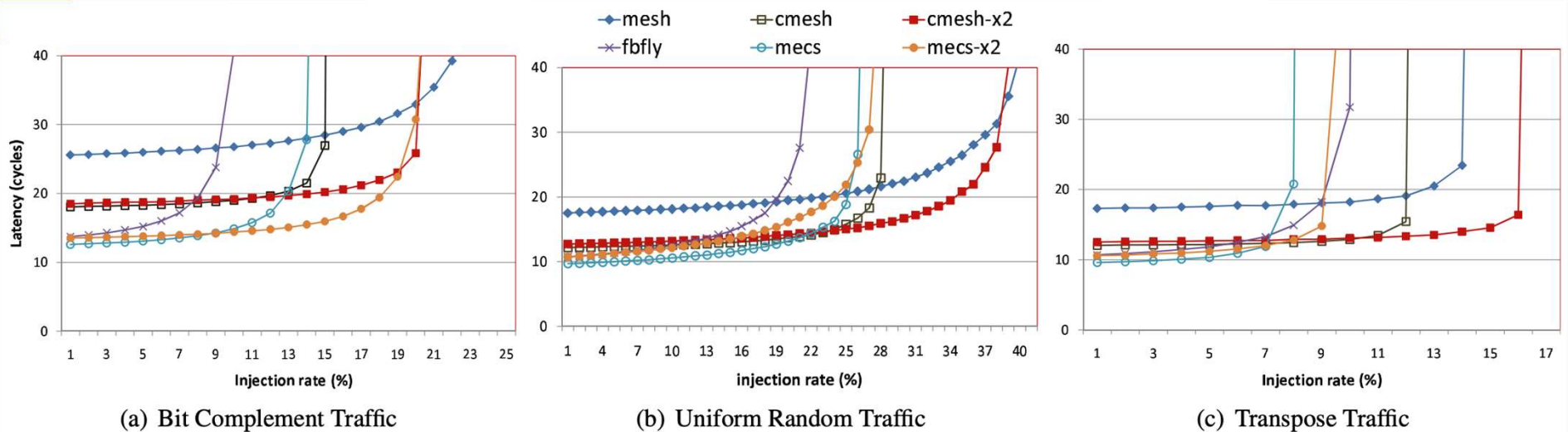


Figure 4. Load-latency graphs for 64-node mesh, CMesh, flattened butterfly and MECS topologies.

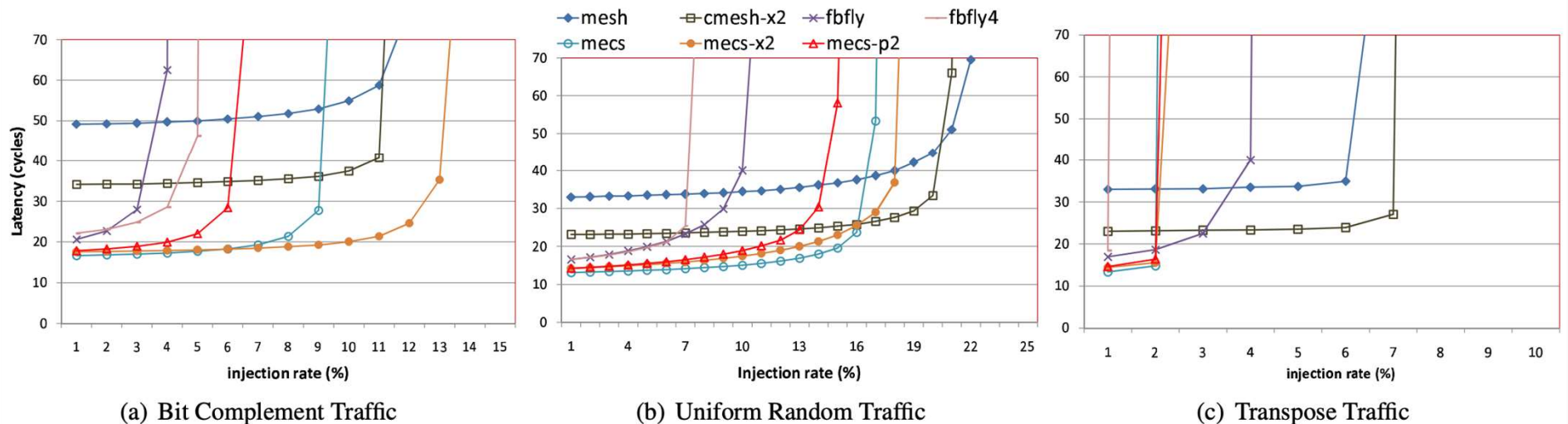


Figure 5. Load-latency graphs for 256-node mesh, CMesh, flattened butterfly and MECS topologies.

Grot+, "Express Cube Topologies for On-Chip Interconnects," HPCA 2009.
MECS is: Multidrop Express Channels

Communication Costs in Parallel Machines

- Along with **idling** (doing nothing) and **contention** (conflict e.g., resource allocation), **communication** is a major overhead in parallel programs.
- The communication cost is usually dependent on a number of features including the following:
 - Programming model for communication
 - Network topology
 - Data handling and routing
 - Associated network protocols
- Usually, distributed systems suffer from major communication overheads.

Message Passing Costs in Parallel Computers

- The total time to transfer a message over a network comprises of the following:
 - **Startup time (t_s):** Time spent at sending and receiving nodes (preparing the message[adding headers, trailers, and parity information] , executing the routing algorithm, establishing interface between node and router, etc.).
 - **Per-hop time (t_h):** This time is a function of number of hops (steps) and includes factors such as switch latencies to copy data from input port to output port.
 - Also known as **node latency**.
 - **Per-word transfer time (t_w):** This time includes all overheads that are determined by the length of the message. If channel bandwidth is r words/s then each word take $t_w = 1/r$ to traverse the link.

Message Passing Costs in Parallel Computers

Store-and-Forward Routing

- A message traversing multiple hops is completely received at an intermediate hop before being forwarded to the next hop.
- The total communication cost for a message of size m words to traverse l communication links is

$$t_{comm} = t_s + (mt_w + t_h)l.$$

- In most platforms, t_h is small and the above expression can be approximated by

$$t_{comm} = t_s + mlt_w.$$

Message Passing Costs in Parallel Computers

Packet Routing

- Store-and-forward makes poor use of communication resources.
- Packet routing breaks messages into packets and pipelines them through the network.
- Since packets may take different paths, each packet must carry routing information, error checking, sequencing, and other related header information.
- The total communication time for packet routing is approximated by: $t_{comm} = t_s + t_h l + t_w m$.
- Here factor t_w also accounts for overheads in packet headers.

Message Passing Costs in Parallel Computers

Cut-Through Routing

- Takes the concept of packet routing to an extreme by further dividing messages into basic units called **flits** or flow control digits.
- Since flits are typically small, the header information must be minimized.
- This is done by forcing all flits to take the same path, in sequence.
- A tracer message first programs all intermediate routers. All flits then take the same route.
- Error checks are performed on the entire message, as opposed to flits.
- No sequence numbers are needed.

Message Passing Costs in Parallel Computers

Cut-Through Routing

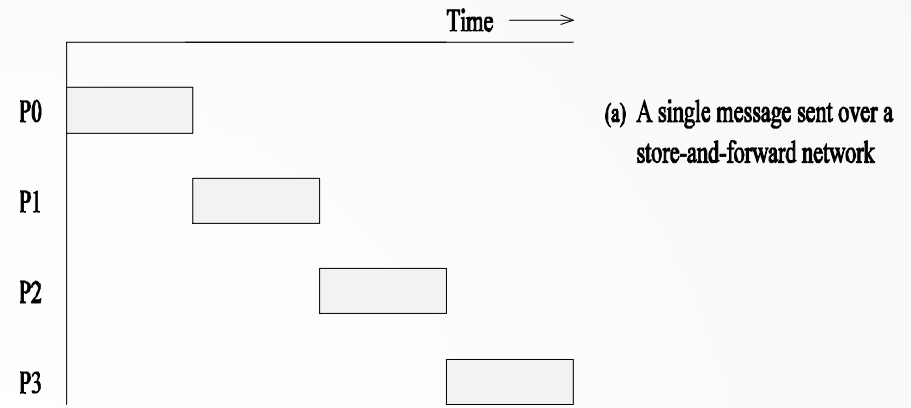
- The total communication time for cut-through routing is approximated by:

$$t_{comm} = t_s + t_h l + t_w m.$$

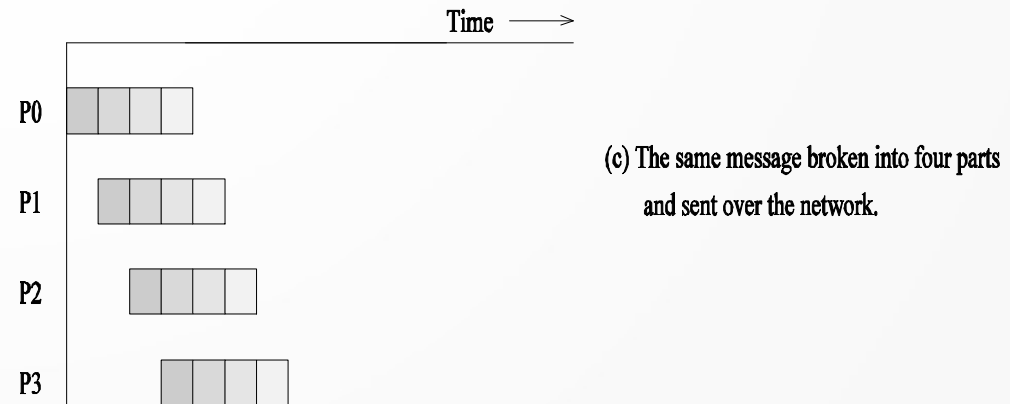
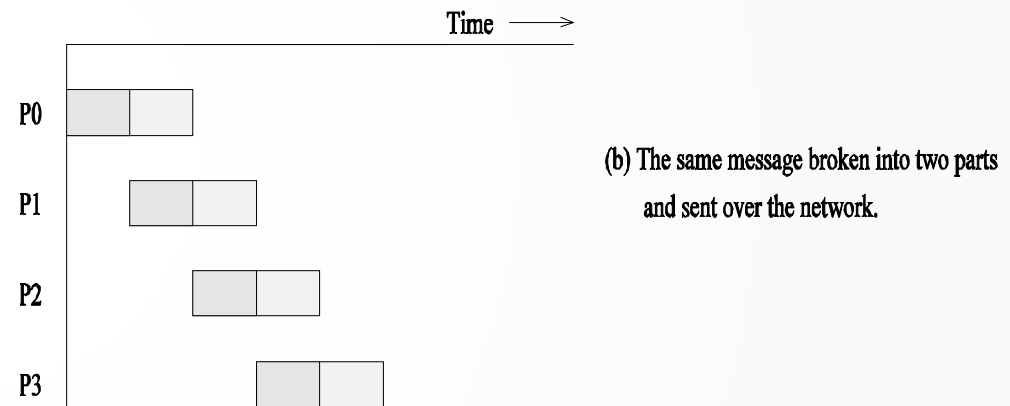
- This is identical to packet routing, however, t_w is typically much smaller.

Message Passing Costs in Parallel Computers

(a) through a store-and-forward communication network;



b) and (c) extending the concept to cut-through routing.



Message Passing Costs in Parallel Computers

Simplified Cost Model for Communicating Messages

- The cost of communicating a message between two nodes l hops away using cut-through routing is given by

$$t_{comm} = t_s + lt_h + t_w m.$$

- In this expression, t_h is typically smaller than t_s and t_w . For this reason, the second term in the RHS does not show, particularly, when m is large.
- For these reasons, we can approximate the cost of message transfer by

$$t_{comm} = t_s + t_w m.$$

Message Passing Costs in Parallel Computers

Simplified Cost Model for Communicating Messages

- It is important to note that the original expression for communication time is valid for only **uncongested networks**.
- Different communication patterns congest different networks to varying extents.
- It is important to **understand and account for** this in the communication time accordingly.



Principles of Parallel Algorithm Design

(Chapter 3 of textbook)

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.

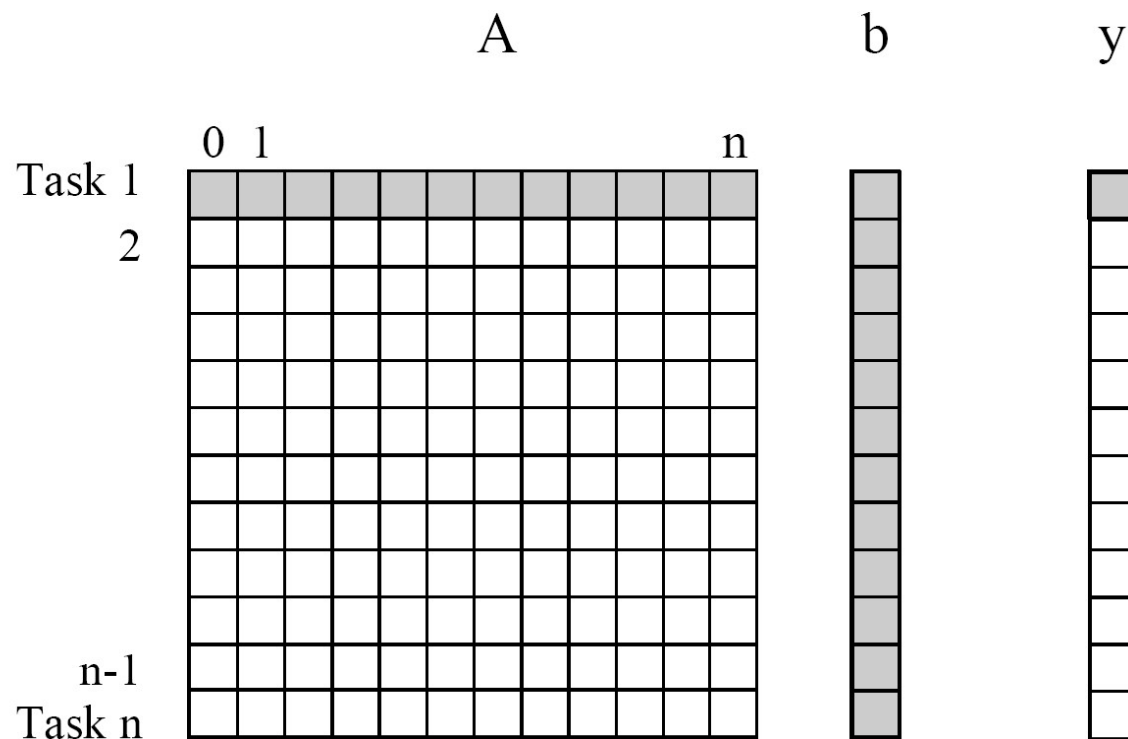


Figure 3.1 Decomposition of dense matrix-vector multiplication into n tasks, where n is the number of rows in the matrix. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

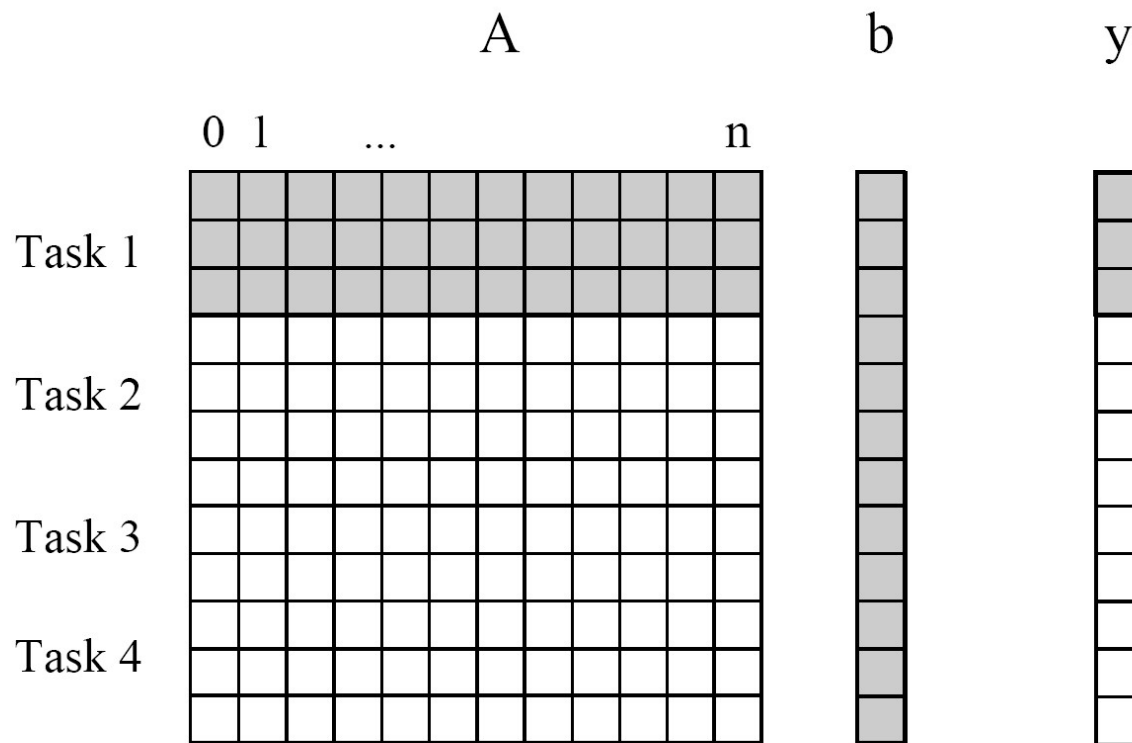


Figure 3.4 Decomposition of dense matrix-vector multiplication into four tasks. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

Principles of Parallel Algorithm Design

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 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.

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.

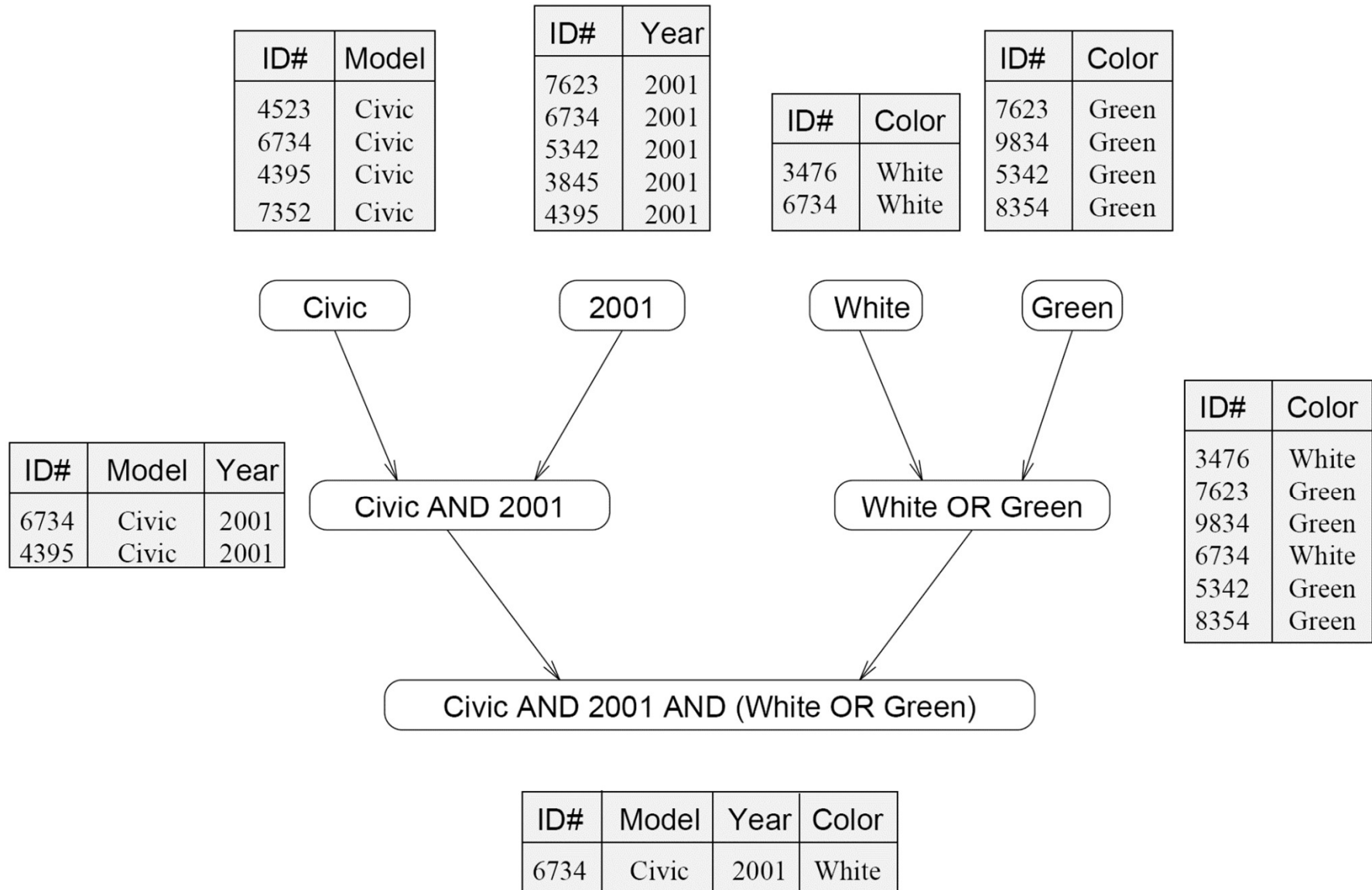


Figure 3.2 The different tables and their dependencies in a query processing operation.

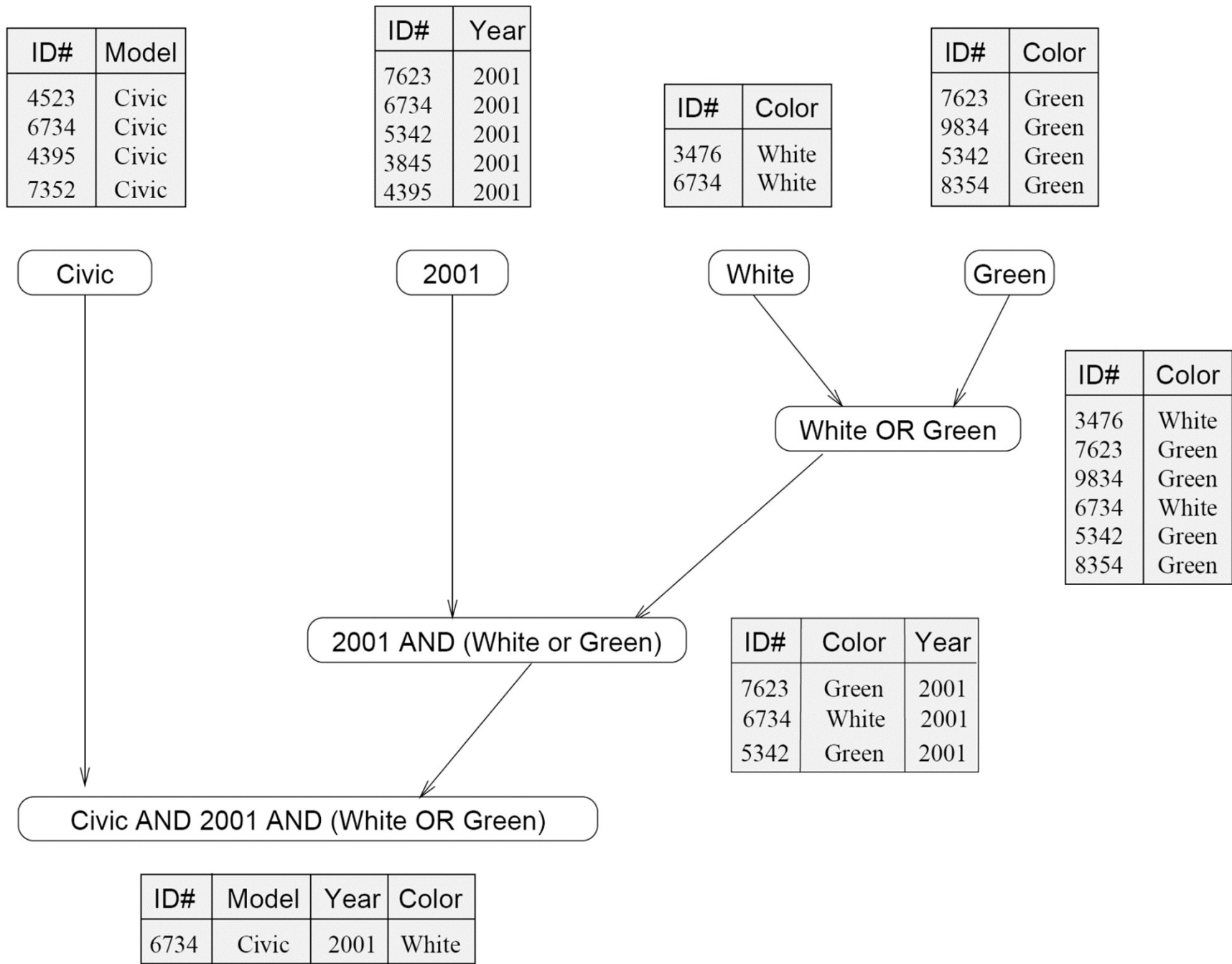


Figure 3.3 An alternate data-dependency graph for the query processing operation.

Principles of Parallel Algorithm Design

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

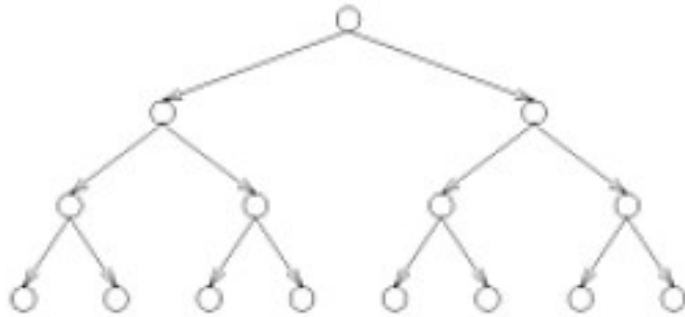
principles of Parallel Algorithm Design

Maximum Degree of Concurrency

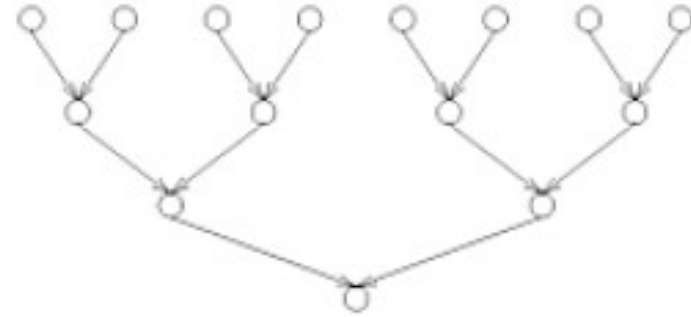
- The maximum number of tasks that can be executed simultaneously in a parallel program at any given time is known as its *maximum degree of concurrency*
- Usually, it is always less than total number of tasks due to dependencies.
- E.g., max-degree of concurrency in the task-graphs of Figures 3.2 and 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

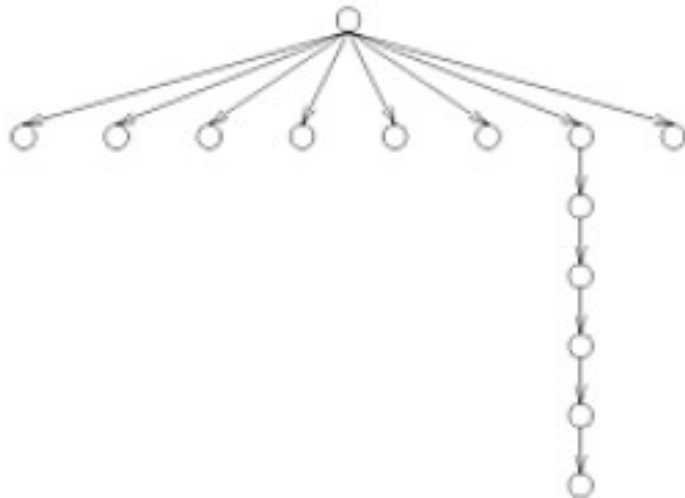
Determine Maximum Degree of Concurrency?



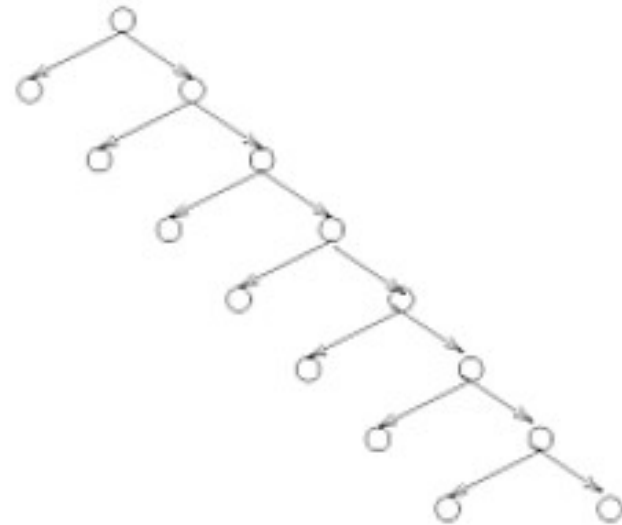
(a)



(b)



(c)



(d)

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(finier)

principles of Parallel Algorithm Design

Average Degree of Concurrency

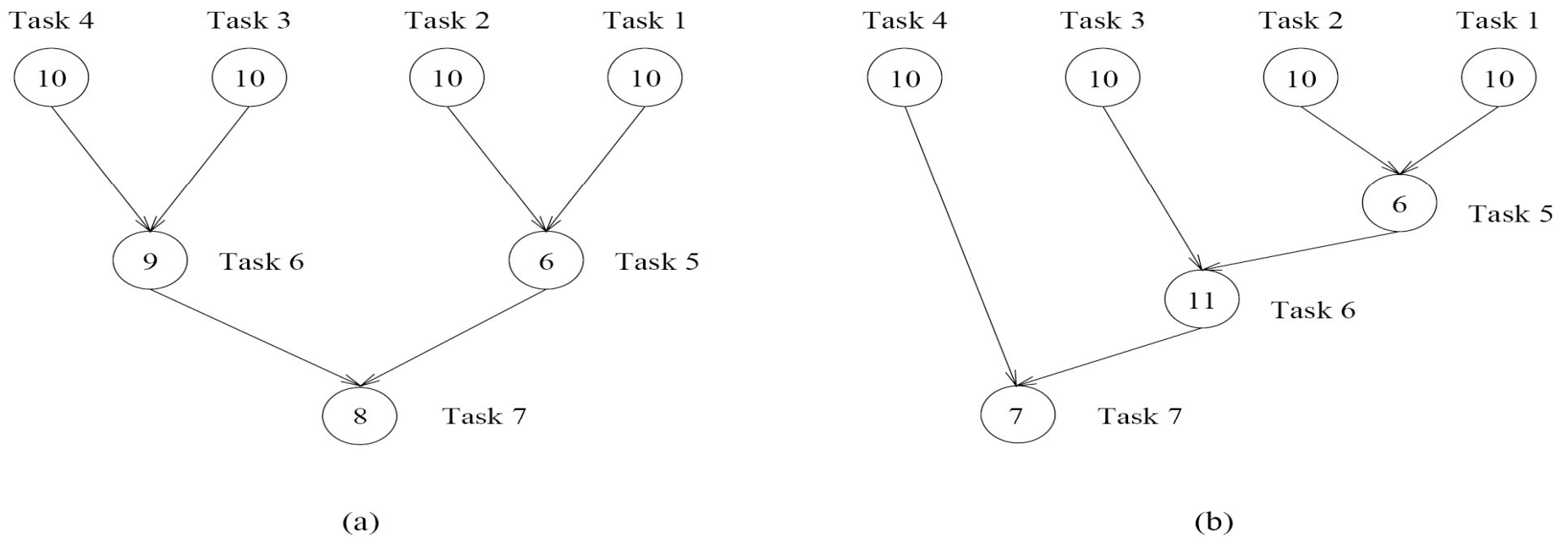


Figure 3.5 Abstractions of the task graphs of Figures 3.2 and 3.3, respectively.

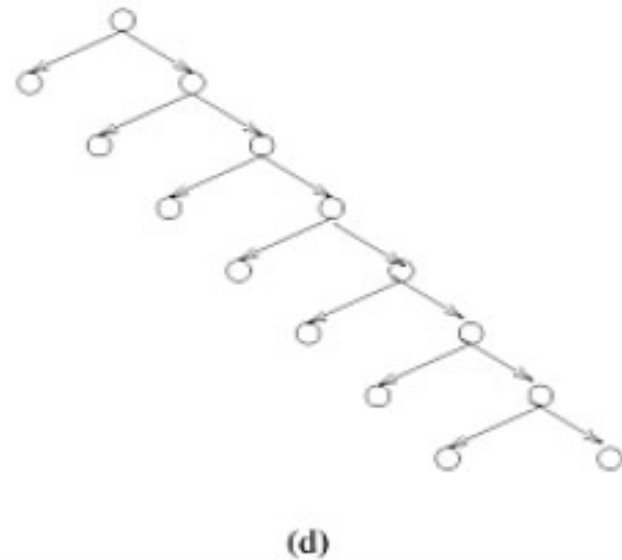
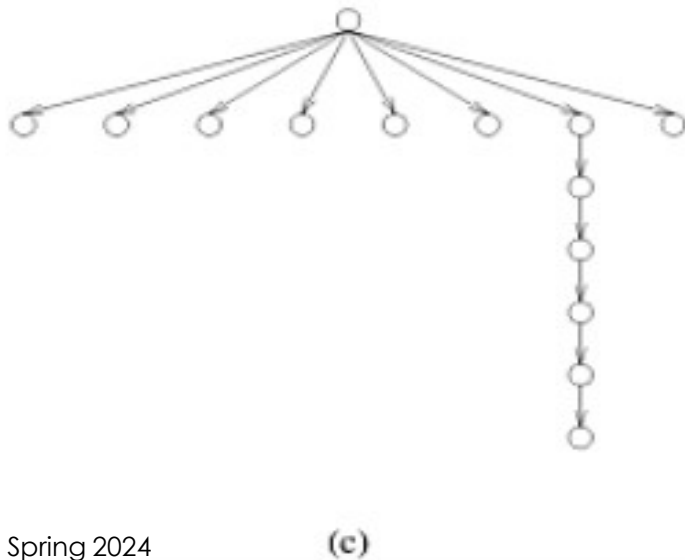
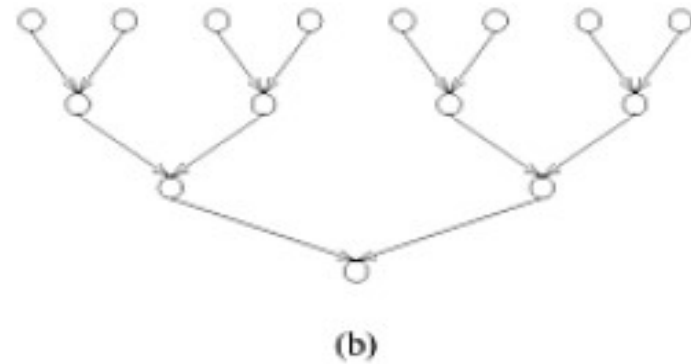
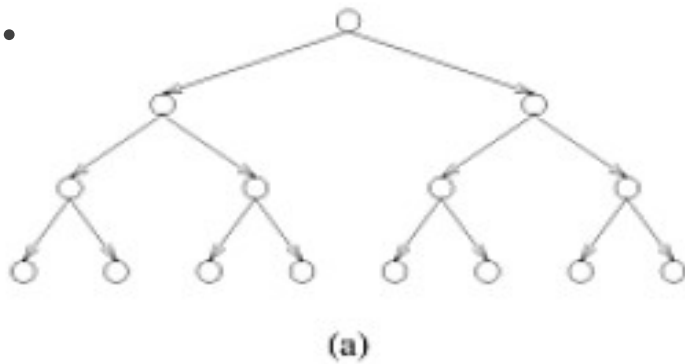
Critical path lengths: 27 and 34

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? Assume work done by each node as 1 unit.



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)

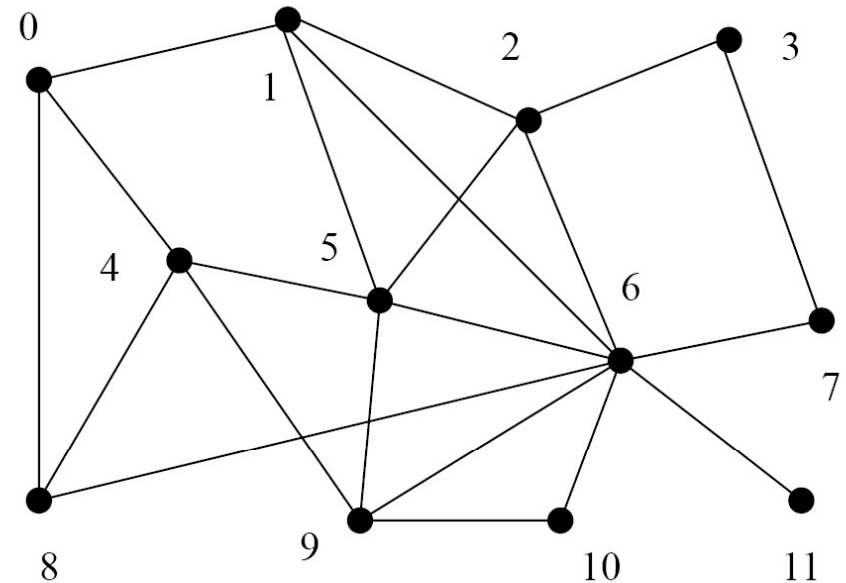
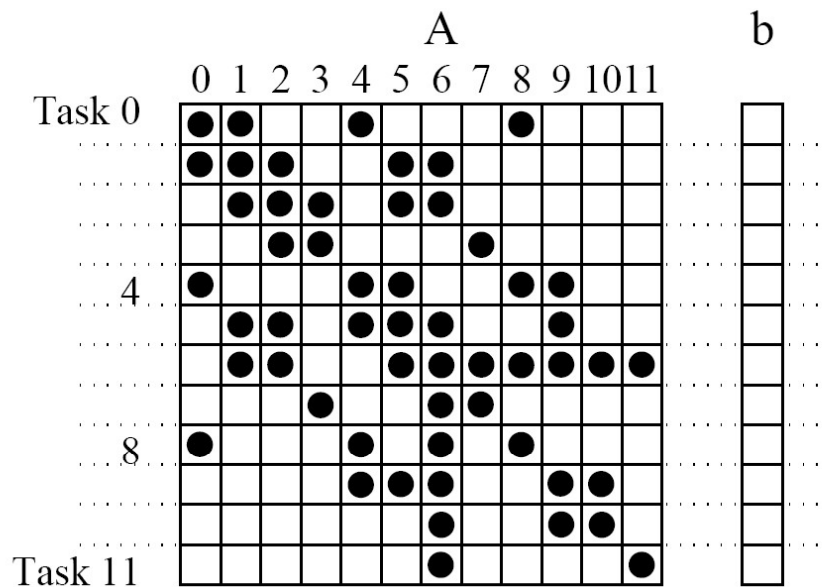


Figure 3.6 A decomposition for sparse matrix-vector multiplication and the corresponding task-interaction graph. In the decomposition Task i computes $\sum_{0 \leq j \leq 11, A[i,j] \neq 0} A[i, j] \cdot b[j]$.

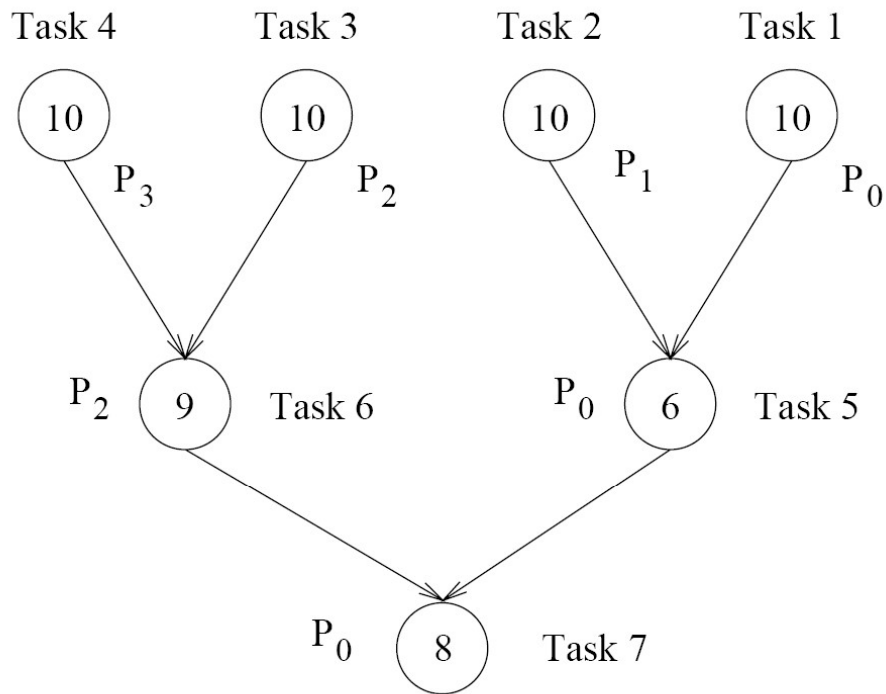
principles of Parallel Algorithm Design

Processes and Mapping

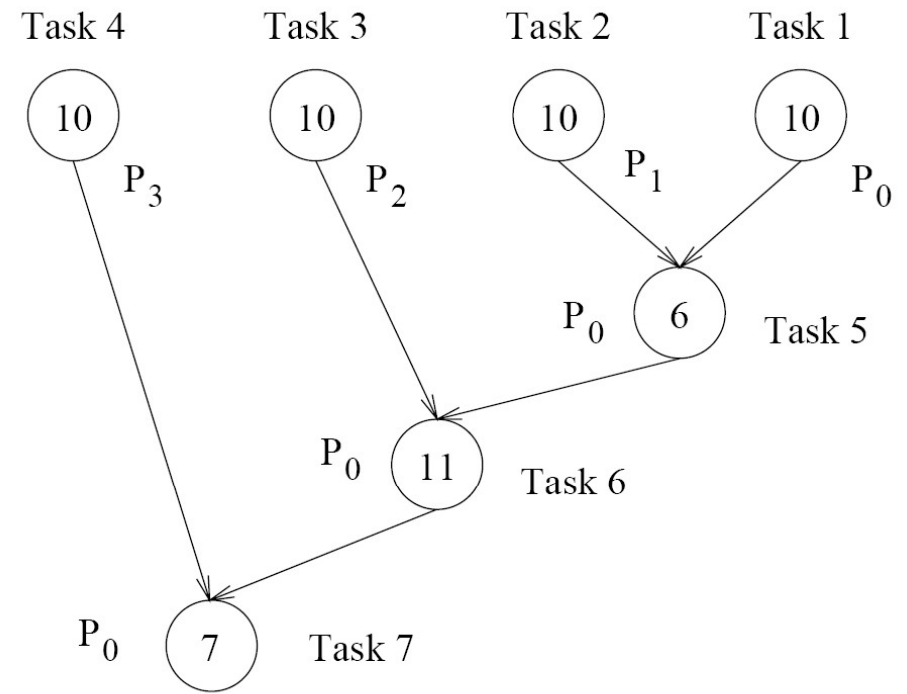
- Logical processing or computing agent that performs tasks is called **process**.
- The mechanism by which tasks are assigned to processes for execution is called **mapping**.
- Multiple tasks can be mapped on a single process
- Independent task should be mapped onto different processes
- Map tasks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

principles of Parallel Algorithm Design

Processes and Mapping



(a)



(b)

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

Point to ponder: Auto parallelization?

- **Can some algorithm or AI-assisted program find parallelism automatically?**
 - “Tools and compilers for automatic parallelization at the current state of the art seem to work well only for highly structured programs or portions of programs” **[your textbook]**
 - **Traditionally this direction of research hasn’t shown much promise**

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).