

Lecture 4: Principles of Parallel Algorithm Design (part 1)

Constructing a Parallel Algorithm

- *identify portions of work that can be performed concurrently*
- *map concurrent portions of work onto multiple processes running in parallel*
- distribute a program's input, output, and intermediate data
- manage accesses to shared data: avoid conflicts
- synchronize the processes at stages of the parallel program execution

Task Decomposition and Dependency Graphs

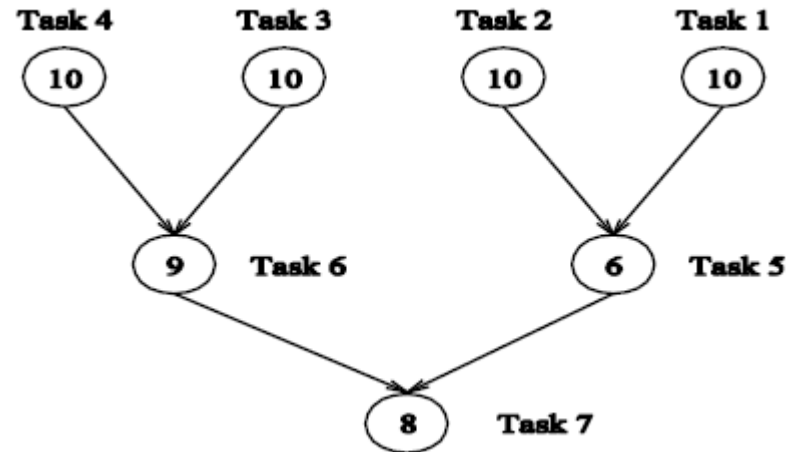
Decomposition: divide a computation into smaller parts, which can be executed concurrently

Task: programmer-defined units of computation.

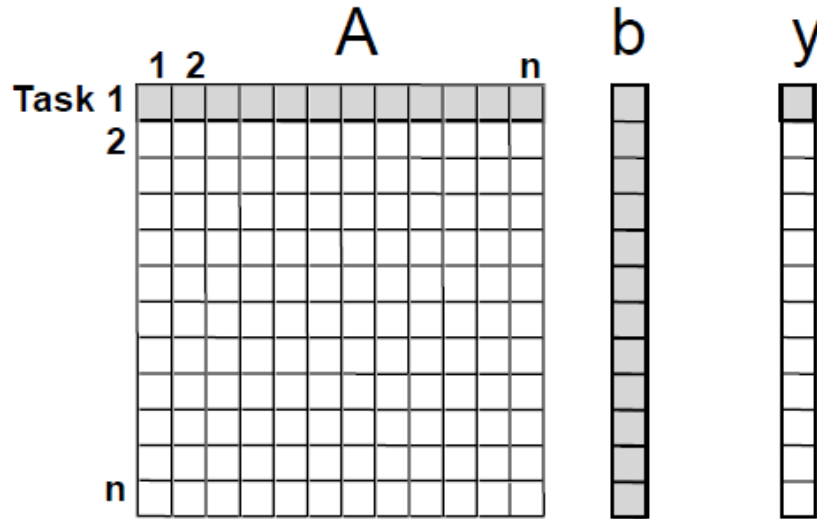
Task-dependency graph:

Node represents a task.

Directed edge represents control dependence.



Example 1: Dense Matrix-Vector Multiplication



- Computing $y[i]$ only use i th row of A and b – treat computing $y[i]$ as a task.
- Remark:
 - Task size is uniform
 - No dependence between tasks
 - All tasks need b

Example 2: Database Query Processing

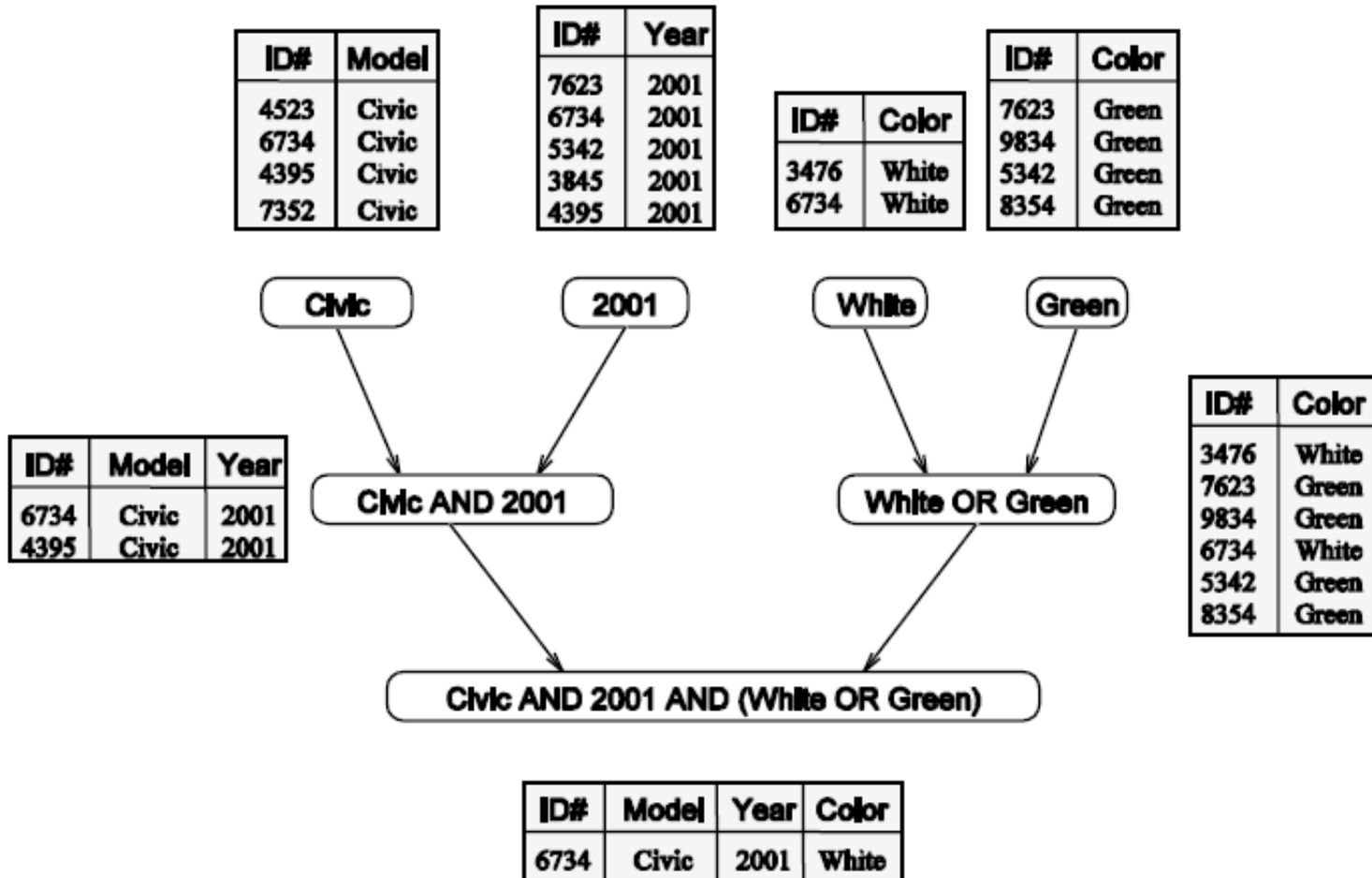
- Executing the query:

Model = "civic" **AND** Year = "2001" **AND** (Color = "green" **OR** Color = "white")

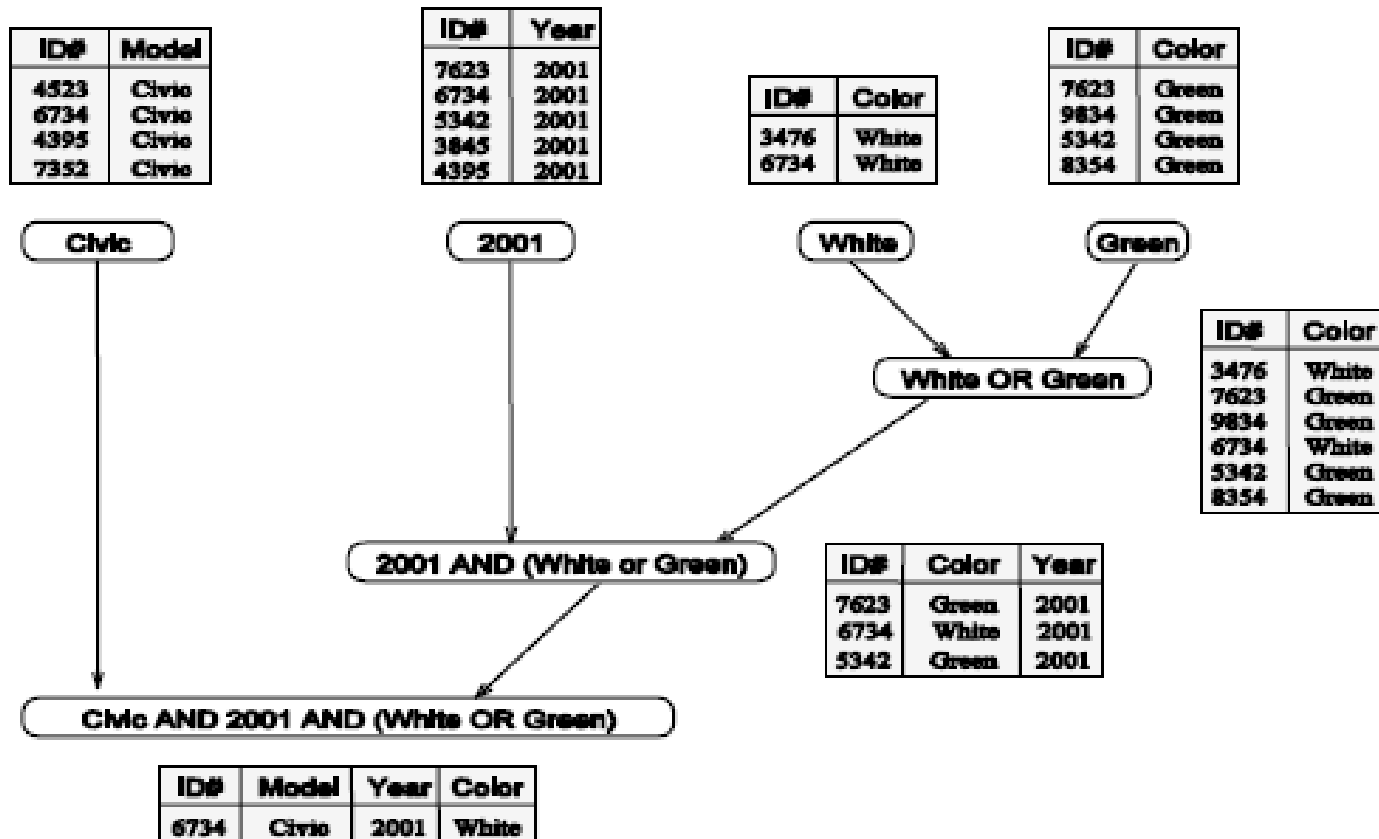
on the following database:

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

- **Task:** create sets of elements that satisfy a (or several) criteria.
- **Edge:** output of one task serves as input to the next



- An alternate task-dependency graph for query



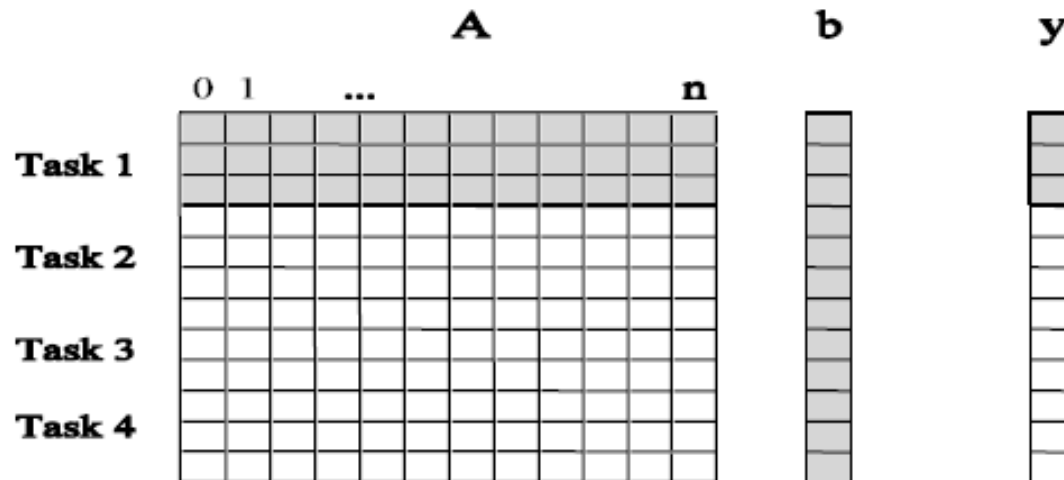
- Different task decomposition leads to different parallelism

Granularity of Task Decomposition

- **Fine-grained** decomposition: large number of small tasks
- **Coarse-grained** decomposition: small number of large tasks

Matrix-vector multiplication example

-- **coarse-grain**: each task computes 3 elements of $y[]$



Degree of Concurrency

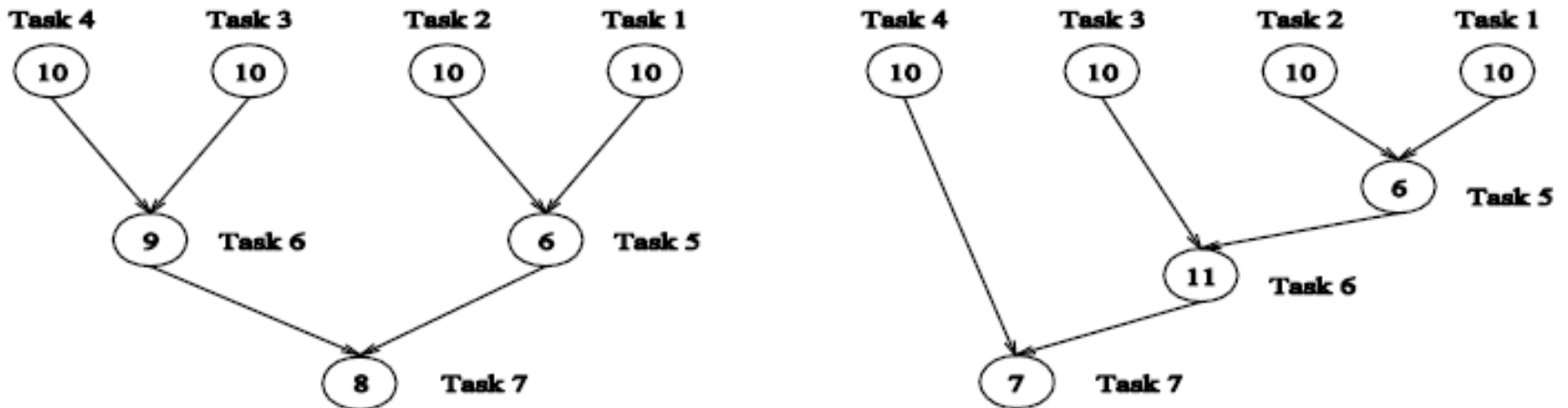
- **Degree of Concurrency:** # of tasks that can execute in parallel
 - **maximum degree of concurrency:** largest # of concurrent tasks at any point of the execution
 - **average degree of concurrency:** average # of tasks that can be executed concurrently
- Degree of Concurrency vs. Task Granularity
 - Inverse relation

Critical Path of Task Graph

- **Critical path:** The longest directed path between any pair of *start node* (node with no incoming edge) and *finish node* (node with on outgoing edges).
- **Critical path length:** The sum of weights of nodes along critical path.
 - The weights of a node is the size or the amount of work associated with the corresponding task
- **Average degree of concurrency** = total amount of work / critical path length

Example: Critical Path Length

Task-dependency graphs of query processing operation



Left graph:

Critical path length = 27

Average degree of concurrency = $63/27 = 2.33$

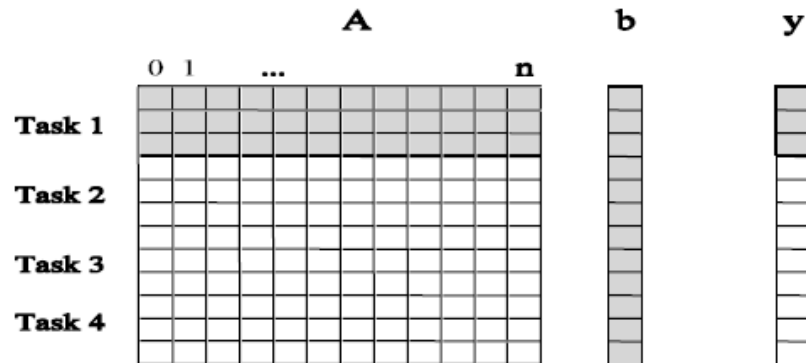
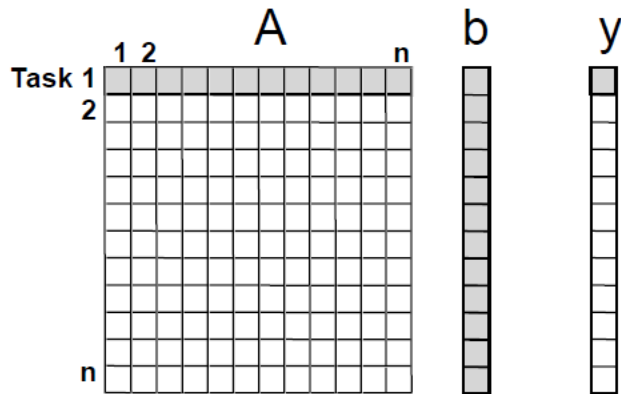
Right graph:

Critical path length = 34

Average degree of concurrency = $64/34 = 1.88$

Limits on Parallelization

- Facts bounds on parallel execution
 - Maximum task granularity is finite
 - Matrix-vector multiplication $O(n^2)$
 - Interactions between tasks
 - Tasks often share input, output, or intermediate data, which may lead to interactions not shown in task-dependency graph.



Ex. For the matrix-vector multiplication problem, all tasks are independent, and all need access to the entire input vector b .

- **Speedup** = sequential execution time/parallel execution time
- **Parallel efficiency** = sequential execution time/(parallel execution time × processors used)