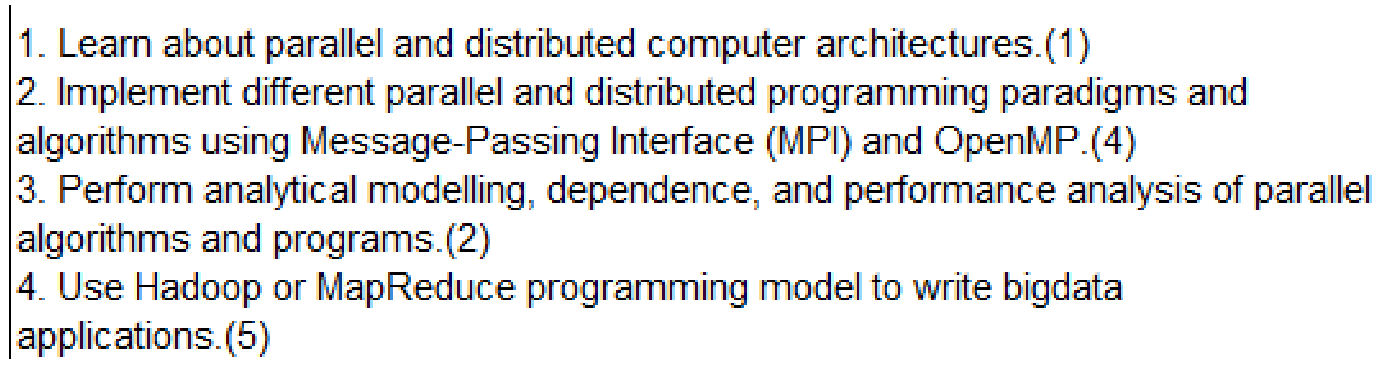
CS 3006 Parallel and Distributed Computer

Fall 2022



1. Learn about parallel and distributed computer architectures.(1)
2. Implement different parallel and distributed programming paradigms and algorithms using Message-Passing Interface (MPI) and OpenMP.(4)
3. Perform analytical modelling, dependence, and performance analysis of parallel algorithms and programs.(2)
4. Use Hadoop or MapReduce programming model to write bigdata applications.(5)

Week # 5 – Lecture # 13, 14, 15

22nd, 24th, 25th Safar ul Muzaffar, 1444

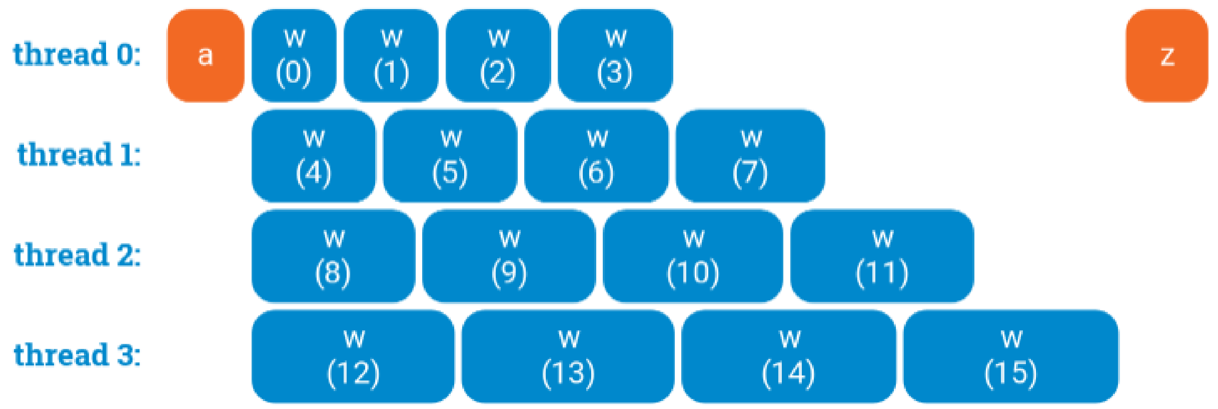
19th, 21st, 22nd September 2022

Dr. Nadeem Kafi Khan

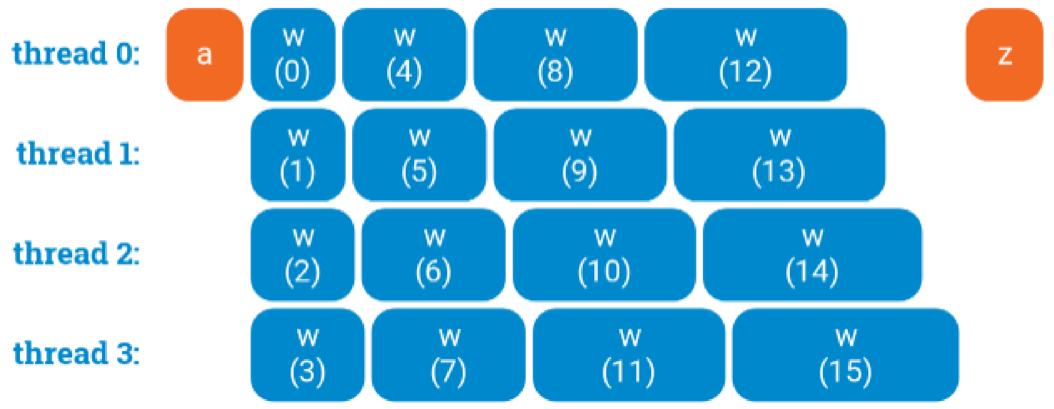
Lecture # 13 – Topics (Lab # 5)

- **#pragma omp for**
- **How it works with #pragma omp parallel**
- **Shorthand: #pragma omp parallel for**
- **Two common mistakes whileusing #pragma omp for**

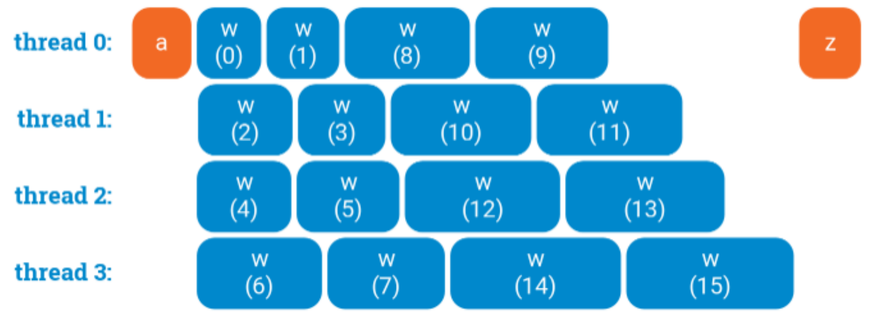
```
a();
#pragma omp parallel for
for (int i = 0; i < 16; ++i) {
    w(i);
}
z();
```



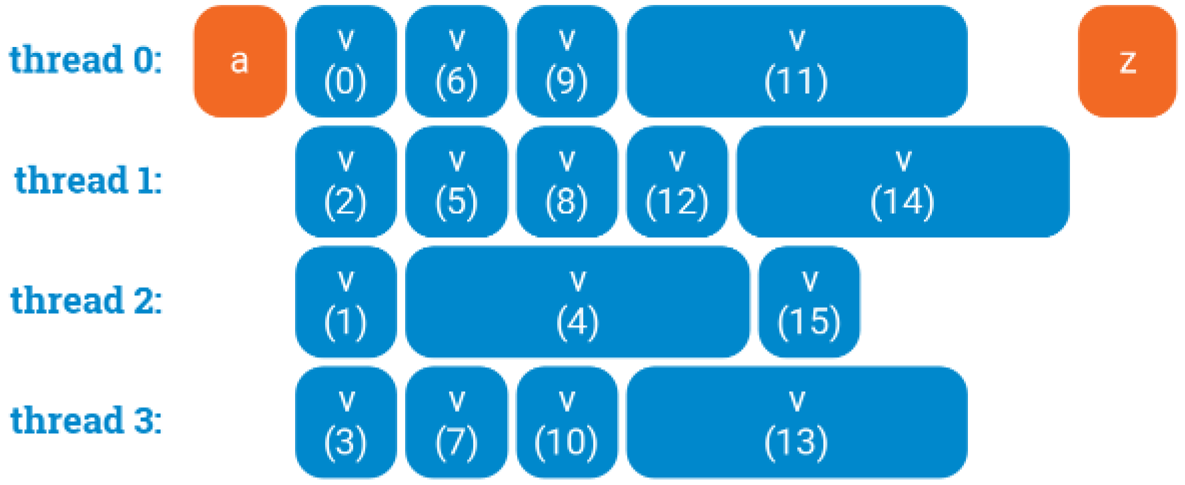
```
a();
#pragma omp parallel for schedule(static,1)
for (int i = 0; i < 16; ++i) {
    w(i);
}
z();
```



```
a();
#pragma omp parallel for schedule(static,2)
for (int i = 0; i < 16; ++i) {
    w(i);
}
z();
```



```
a();
#pragma omp parallel for schedule(dynamic,1)
for (int i = 0; i < 16; ++i) {
    v(i);
}
z();
```



However, please note that dynamic **scheduling is expensive**: there is some communication between the threads after each iteration of the loop! **Increasing the chunk size** (number "1" in the `schedule` directive) may help here to find a better trade-off between balanced workload and coordination overhead.

Lecture # 14 – Topics

- Decomposition, Task, Dependency Graph
- Dense Matric Vector Multiplication
- Granularity of Task Decompositions
- Database Query Processing
- Task Dependency Graph
- Task mapping to processors (How to save communication cost?)

Preliminaries: Decomposition, Tasks, and Dependency Graphs

- The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently

- A given problem may be decomposed into tasks in many different ways.

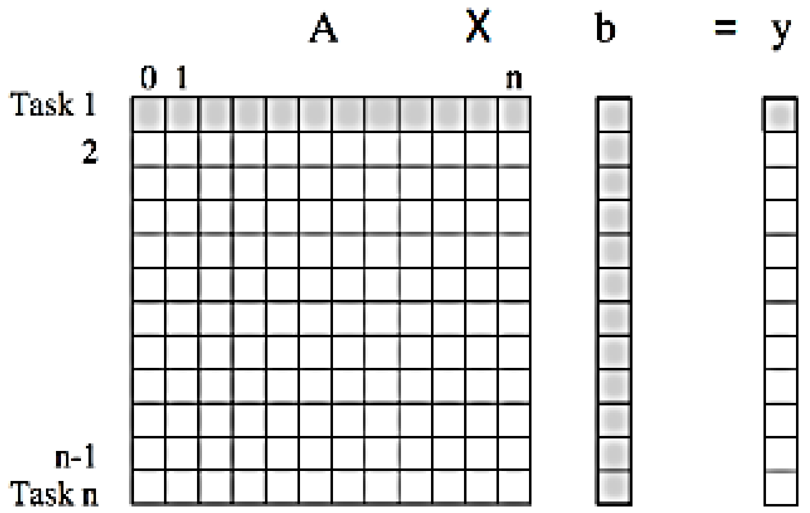
- Tasks may be of same, different, or even indeterminate sizes.

- A **decomposition can be illustrated in the form of a directed graph** with nodes corresponding to tasks and edges indicating that the result of one task is required for processing the next. Such a graph is called **a *task dependency graph*.**

Example: *Dense* Matrix Vector Multiplication

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax+by+cz+dw \\ ex+fy+gz+hw \\ ix+jy+kz+lw \\ mx+ny+oz+pw \end{bmatrix}$$

```
REAL A[n][n], b[n], y[n]; int i, j;
for (i = 0; i < n; i++) {
   sum = 0.0;
   for (j = 0; j < n; j++)
      sum += A[i][j] * b[j];
   c[i] = sum;
}
```



Independent task

- Results can be generated without any interactions between tasks.
- Each task work on a portion of data.

There are *n*2 multiplications and additions in matrix-vector multiplication and the problem cannot be decomposed into more than *O*(*n*2) tasks even by using the most fine-grained decomposition.

- Computation of each element of output vector y is independent
- Decomposed into n tasks, one per element in y → Easy
- Observations
  - Each task only reads one row of A, and writes one element of y
  - All tasks share vector b (the shared data)
  - No control dependencies between tasks
  - All tasks are of the same size in terms of number of operations.

Granularity

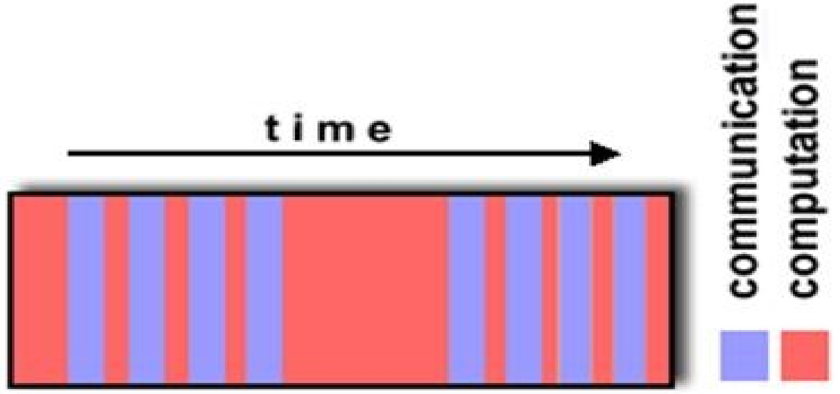
Granularity is the ratio of computation to communication.

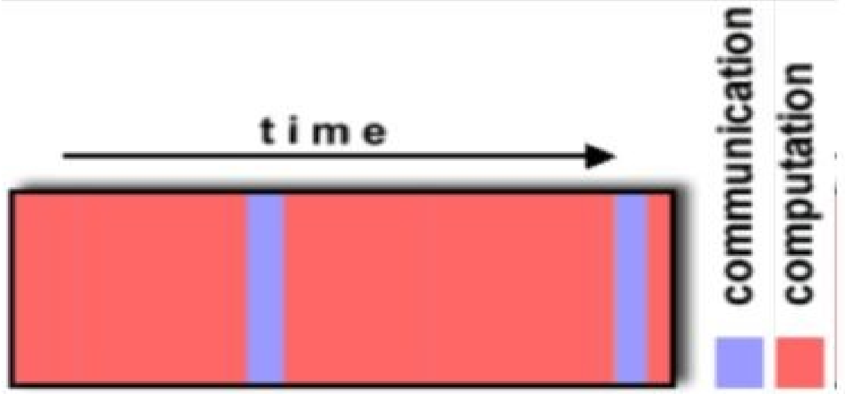
Periods of computation are typically separated from periods of communication by synchronization events.

Fine-grain Parallelism: Relatively small amounts of computational work are done between communication events.

Facilitates load balancing and Implies high communication overhead and less opportunity for performance enhancement

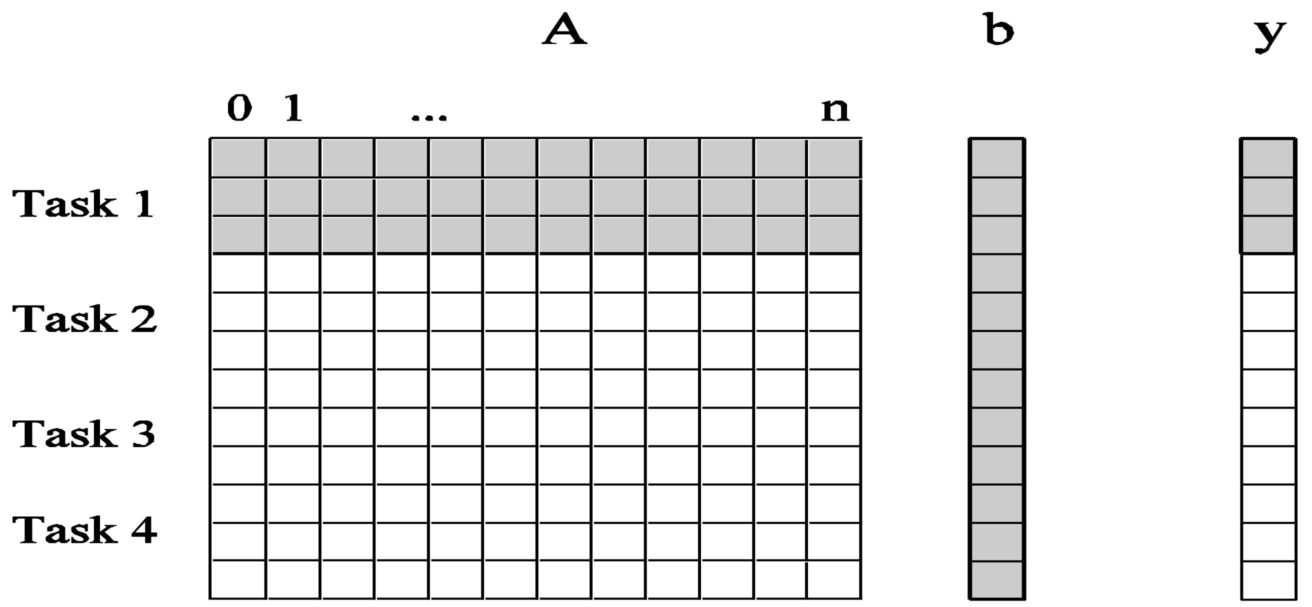
Coarse-grain Parallelism: Relatively large amounts of computational work are done between communication/synchronization events. Harder to load balance efficiently





Granularity of Task Decompositions

- The number of tasks into which a problem is decomposed determines its granularity.
- Decomposition into a large number of tasks results in fine-grained decomposition and that into a small number of tasks results in a coarse grained decomposition.



Task Granularity

- Task need inputs which are generated by other tasks (in various ways).
- Dependent task reads data from shared memory or network transfer.

A coarse grained counterpart to the dense matrix-vector product example. Each task in this example corresponds to the computation of three elements of the result vector.

Example: Database Query Processing

Consider the execution of the query:

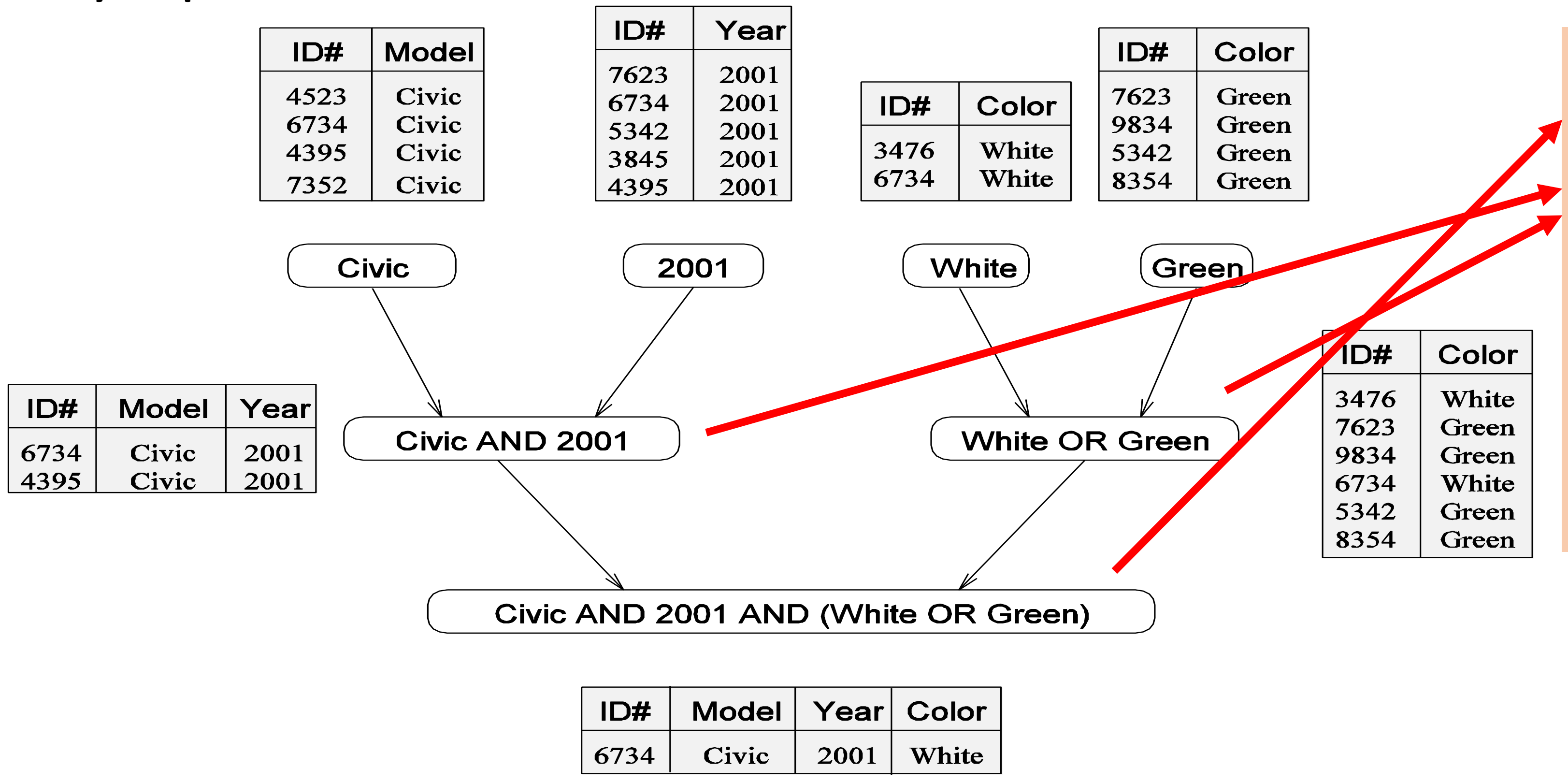
MODEL = ``CIVIC" AND YEAR = 2001 AND
(COLOR = ``GREEN" OR COLOR = ``WHITE)

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

The execution of the query can be divided into subtasks in various ways. Each task can be thought of as generating an intermediate table of entries that satisfy a particular clause.



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

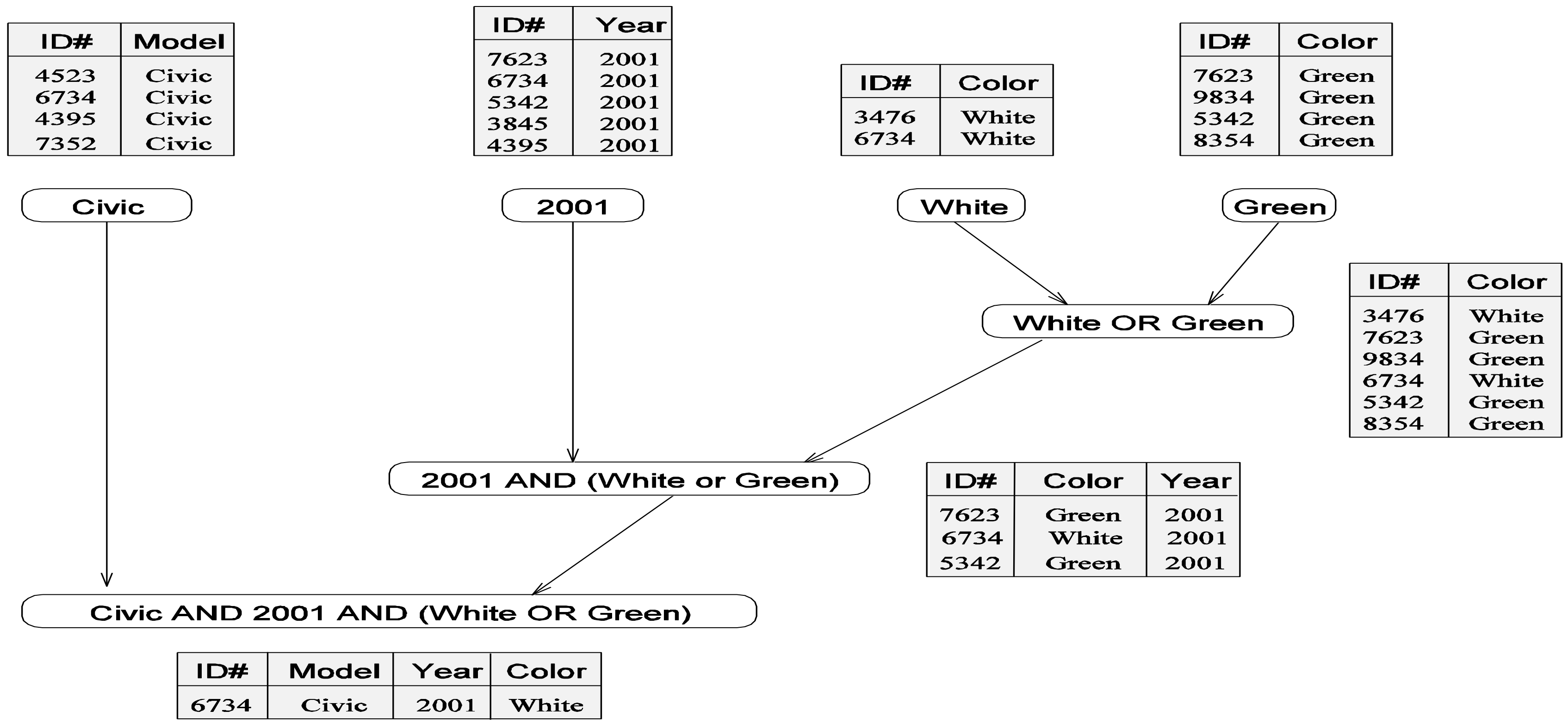
Dependent task

- Task need inputs which are generated by other tasks (in various ways).
- Dependent task reads data from shared memory or network transfer.

Decomposing the given query into a number of tasks. Edges in this graph denote that the output of one task is needed to accomplish the next.

Example: Database Query Processing

Note that the same problem can be decomposed into subtasks in other ways as well.


Civic
2001
White
Green
White OR Green
2001 AND (White or Green)
Civic AND 2001 AND (White OR Green)

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

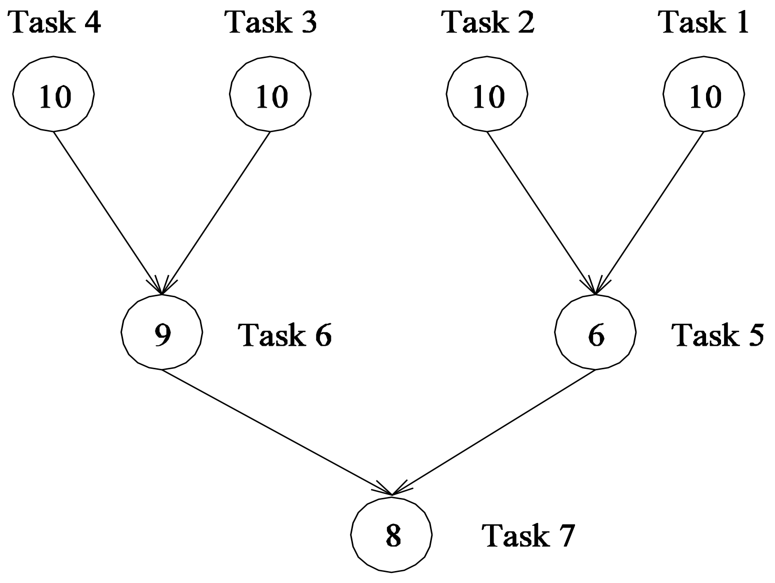
ID#	Model	Year	Color
6734	Civic	2001	White

An alternate decomposition of the given problem into subtasks, along with their data dependencies. Different task decompositions may lead to significant differences with respect to their eventual parallel performance.

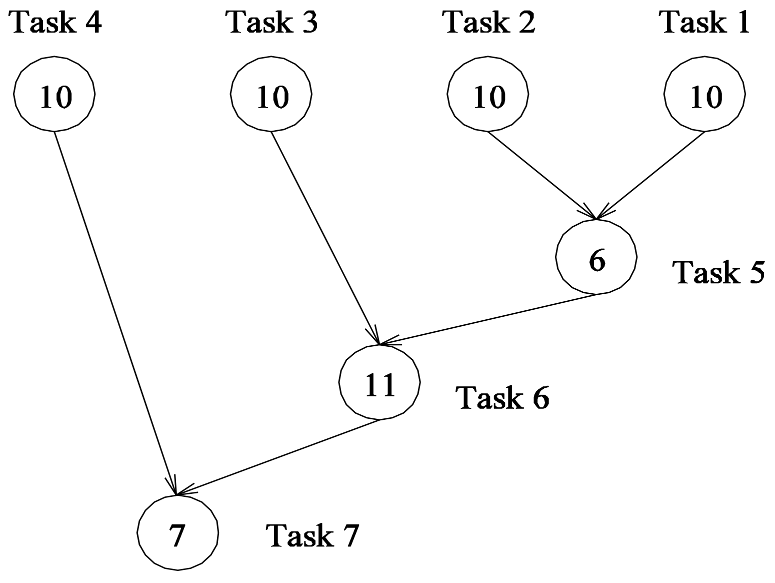
Task Dependency Graph

* The weight of a node is the size or the amount of work associated with the corresponding task.

- **Decomposition can be illustrated in the form of a directed graph** with nodes corresponding to tasks and edges indicating that the result of one task is required for processing the next. Such a graph is called ***a task dependency graph*.**



(a)



(b)

Task Dependency Graph

- For example, the decomposition of matrix-vector multiplication shown in Figure 3.1 has a fairly small granularity and a large degree of concurrency. The decomposition for the same problem shown in Figure 3.4 has a larger granularity and a smaller degree of concurrency.

Figure 3.1

Figure 3.4

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.

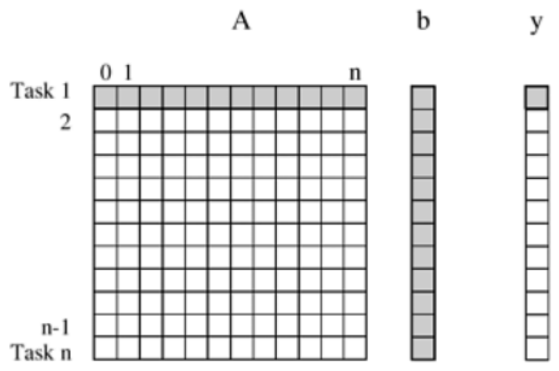
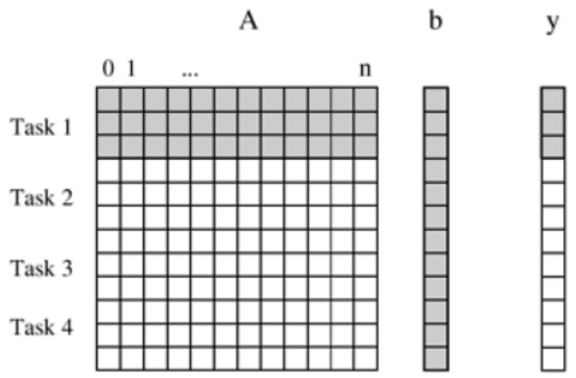


Figure 3.4. Decomposition of dense matrix-vector multiplication into four tasks. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.



Lecture # 15 – Topics

- Degree of Concurrency
    - Maximum, Average,
- Critical Path and Critical Path Length
- Calculation of Critical Path Length
- Calculation of Average degree of concurrency

Degree of Concurrency

- The number of tasks that can be executed in parallel is the *degree of concurrency* of a decomposition.
- Since the number of tasks that can be executed in parallel may change over program execution, the *maximum degree of concurrency* is the maximum number of such tasks at any point during execution. *What is the maximum degree of concurrency of the database query examples?*
  - In general, for task dependency graphs that are trees, the maximum degree of concurrency is always equal to the number of leaves in the tree.
- The *average degree of concurrency* is the average number of tasks that can be processed in parallel over the execution of the program. *Assuming that each tasks in the database example takes identical processing time, what is the average degree of concurrency in each decomposition?*
- The degree of concurrency increases as the decomposition becomes finer in granularity and vice versa.

Critical Path Length

- A directed path in the task dependency graph represents a sequence of tasks that must be processed one after the other.
- A shorter critical path favors a higher degree of concurrency.
- The length of the longest path in a task dependency graph is called the critical path length.

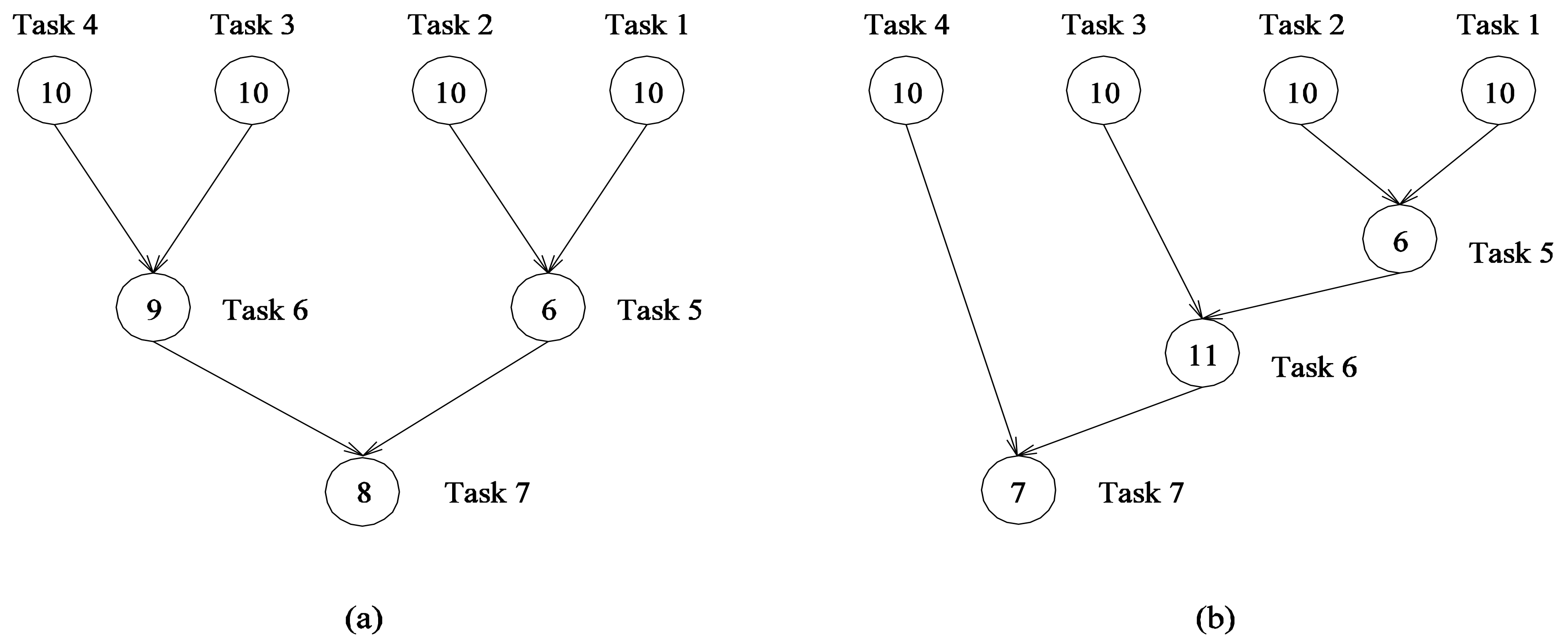
Critical Path Length

- The longest directed path between any pair of start and finish nodes is known as the critical path.
- The sum of the weights of nodes along this path is known as the ***critical path length***, where the weight of a node is the size or the amount of work associated with the corresponding task.
- The ratio of the total amount of work to the critical-path length is the average degree of concurrency. Therefore, a shorter critical path favors a higher degree of concurrency.
- For example, the critical path length is 27 in the task-dependency graph Figure 3.5(a) and is 34 in the task-dependency graph Figure 3.5(b).
- Since the total amount of work required to solve the problems using the two decompositions is 63 and 64, respectively, the average degree of concurrency of the two task-dependency graphs is 2.33 and 1.88, respectively.

Critical Path Length

Textbook Example.

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

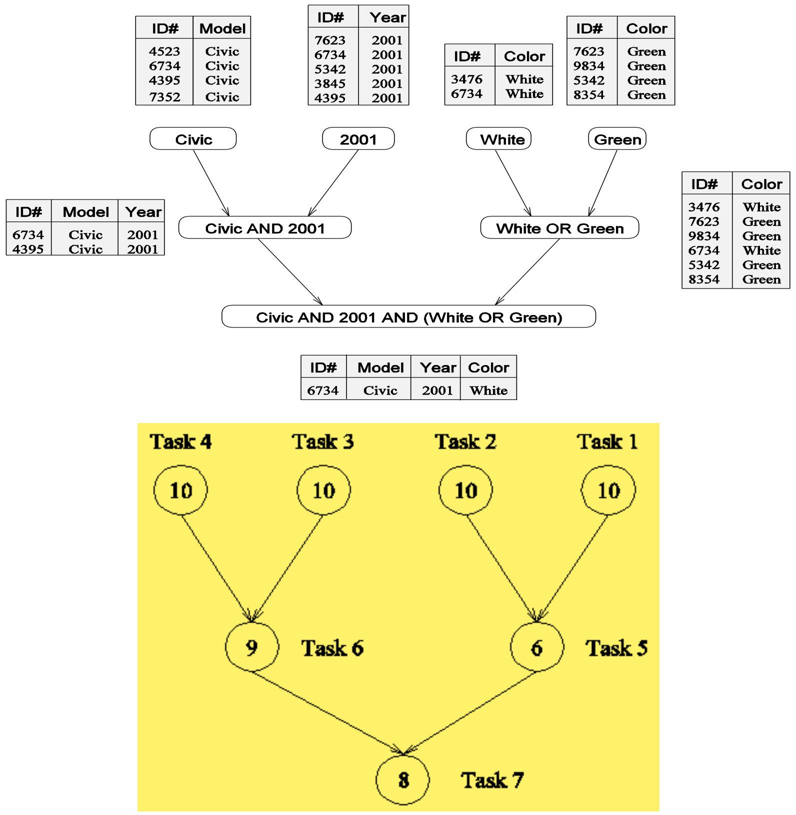


- The **critical path length** in (a) 27 and (b) 34
- **Total amount of work** required to solve the problems in (a) 63 and (b) 64
- **Average degree of concurrency** of the two task-dependency graphs is (a) 2.33 and (b) 1.88

Task Dependency Graph

- For example, the maximum degree of concurrency in the task-graphs of Figures 3.2 and 3.3 is four.

Figure 3.2



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

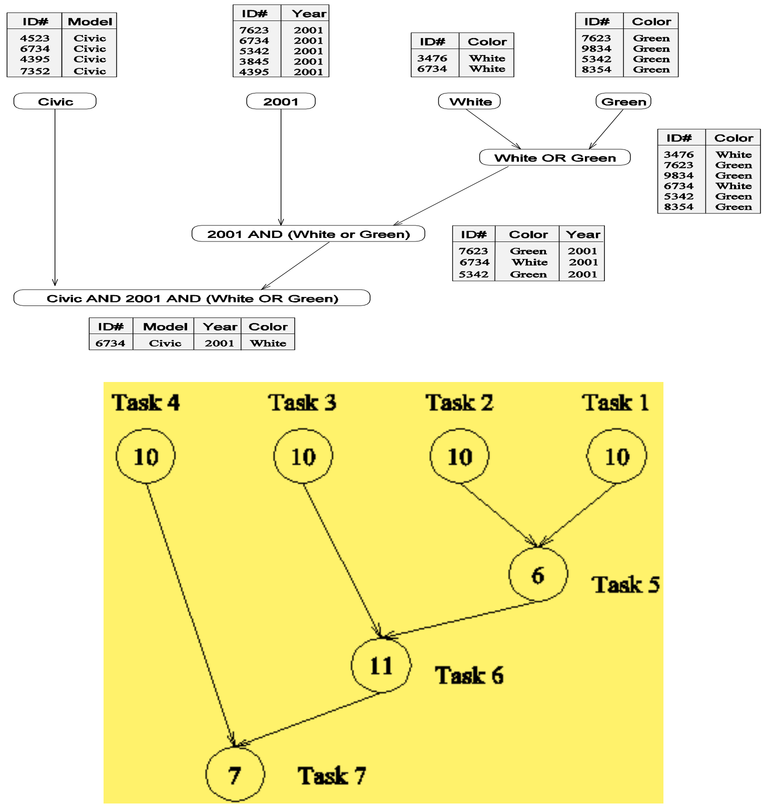
ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

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 3.3



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

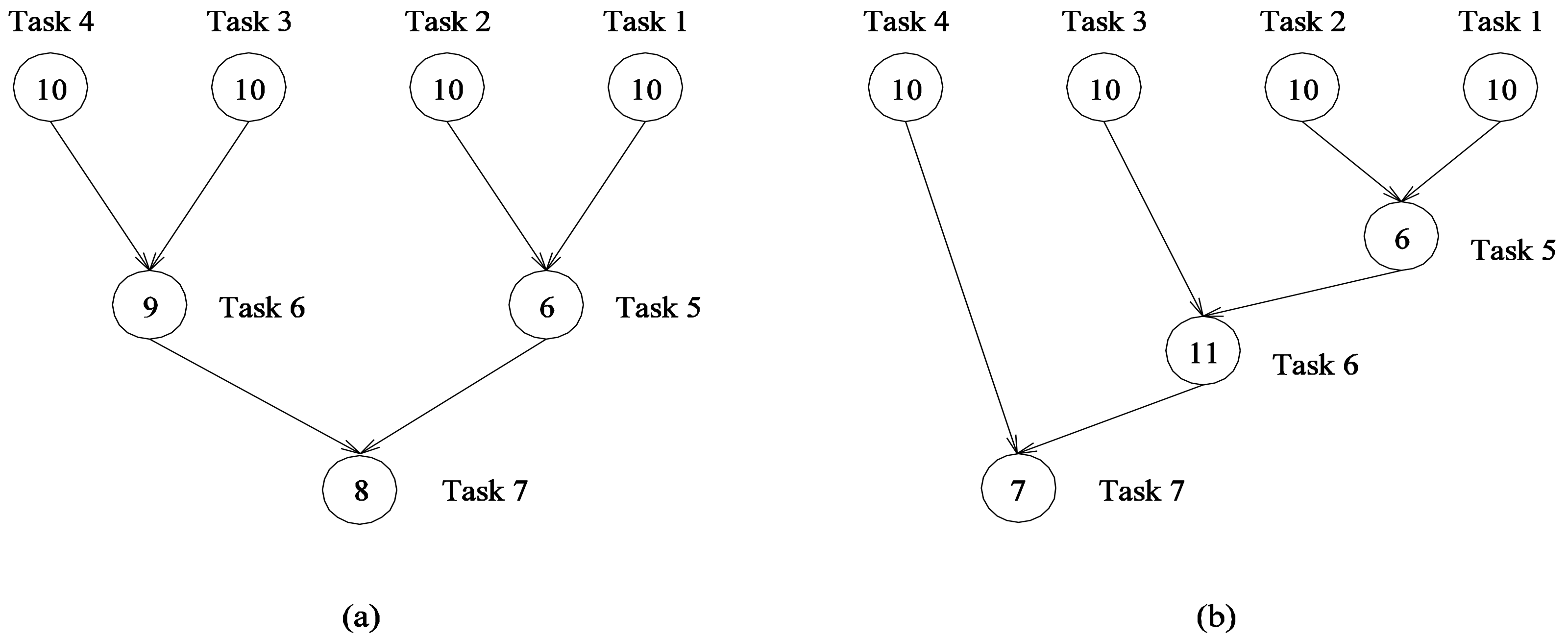
Critical Path Length

Consider the task dependency graphs of the two database query decompositions:

Textbook Example.

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

Please attempt it yourself.



a) What are the critical path lengths for the two task dependency graphs?

b) If each task takes 10 time units, what is the shortest parallel execution time for each decomposition?

c) How many processors are needed in each case to achieve this minimum parallel execution time?

d) What is the maximum degree of concurrency?