IN3200/IN4200: More about parallelization

Examples from A. Grama, A. Gupta, G. Karypis, and V. Kumar: "Introduction to Parallel Computing", Addison Wesley, 2003

Content

- More examples of identifying parallelism
- Simple design of parallel algorithms

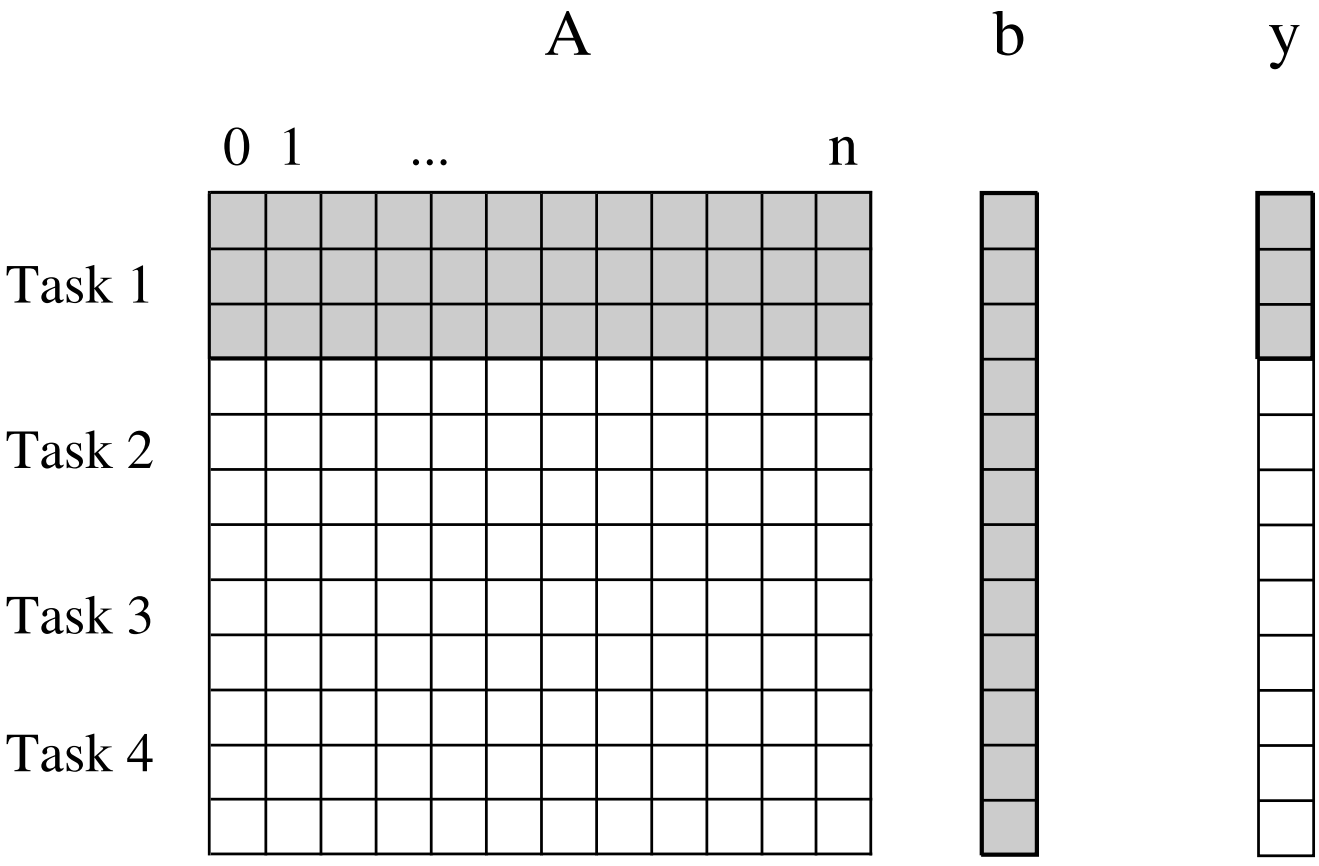
Example: dense matrix-vector multiply

Dense matrix-vector multiply

$$\mathbf{y} = \mathbf{A}\mathbf{b}$$

```
for (i=0; i<N; i++) {
  double tmp = 0.;
  for (j=0; j<N; j++)
    tmp += A[i][j]*b[j];
  y[i] = tmp;
}
```

Parallelization



Decomposition of the outer loop (index `i`) into P chunks, each as the computational task for a processor core. All the tasks are *completely independent*.

Work decomposition

Let N denote the number of entries in vector $\mathbf{y}$ (same as the number of rows in matrix $\mathbf{A}$). If N is divisible by the number of processor cores P, then work decomposition will be perfectly even.

For example: processor core number k ($0 \leq k < P$) can be responsible for computing the following entries of vector $\mathbf{y}$:

```
y[k*chunk_size],
y[k*chunk_size+1],
...
y[(k+1)*chunk_size-1]
```

where `chunk_size=N/P`

Danger for severe load imbalance

What if N is not divisible by P?

Integer division `chunk_size=N/P` will result in

$$\texttt{chunk_size} = \lfloor \frac{N}{P} \rfloor = \frac{N - \text{modulo}(N, P)}{P}$$

That can easily lead to that $P - 1$ processor cores compute each `chunk_size` entries of vector $\mathbf{y}$, whereas one processor core computes modulo(N, P) entries **extra**.

An extreme case of load imbalance arises when $N = 2P - 1$. It will mean that the amount of work for the "heavy-load" processor core is P times of the other processor cores!

Remedy for load balance

The following work decomposition will guarantee that the maximum difference between "heavy-load" and "light-load" tasks is at most 1.

Processor core number k computes

```
y[start_k],
y[start_k+1],
...
y[stop_k-1]
```

where `start_k=k*N/P` and `stop_k=(k+1)*N/P` (integer divisions are used to compute both values).

Example: summing an array of values

```
sum=0.;
for (i=0; i<N; i++)
  sum += y[i];
```

Basic strategy of parallelization:

- Divide the entries of array `y` into as equal-sized chunks as possible

  `start_k=k*N/P` *and* `stop_k=(k+1)*N/P`

- Each processor core *independently* computes a partial sum as

  `y[start_k]+ y[start_k+1]+...+y[stop_k-1]`

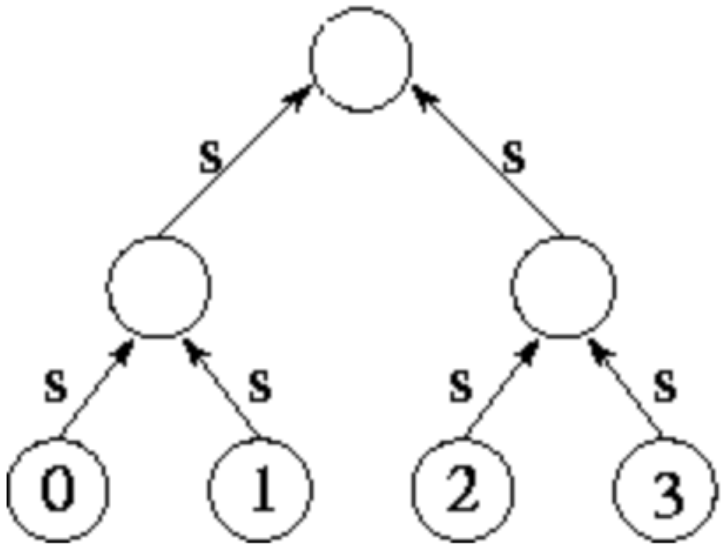
- When all the P partial sums are computed, they are added up to produce the correct value of `sum`

How to sum up P values from P processor cores?

Approach 1: Pick a "master" processor core, and let the master add the P values together.

Downside of this approach: The master core can become a bottleneck if P is large.

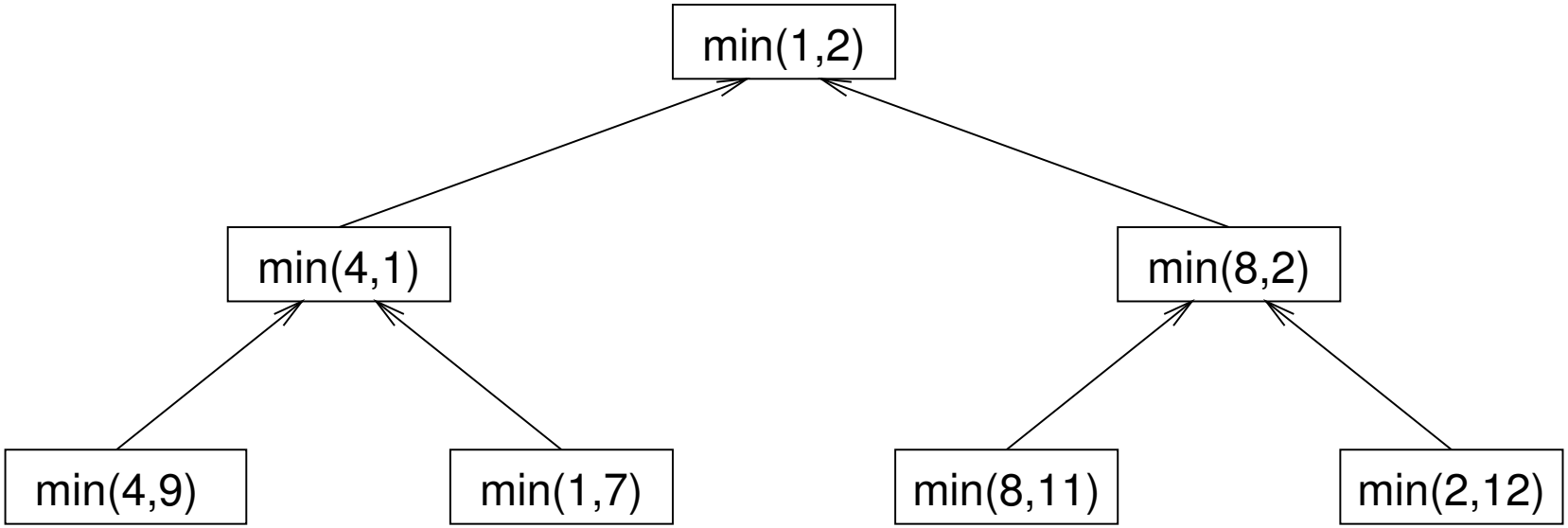
Approach 2: *reverse recursive decomposition*



The "bottom" tasks represent individual partial sums on the processor cores, the other tasks are pair-wise additions until `sum` is computed at the "top".

Another example of reverse recursive decomposition

Suppose we want to find the minimum value in the set $\{4, 9, 1, 7, 8, 11, 2, 12\}$.



Example: Database Query Processing

Consider the execution of the query:

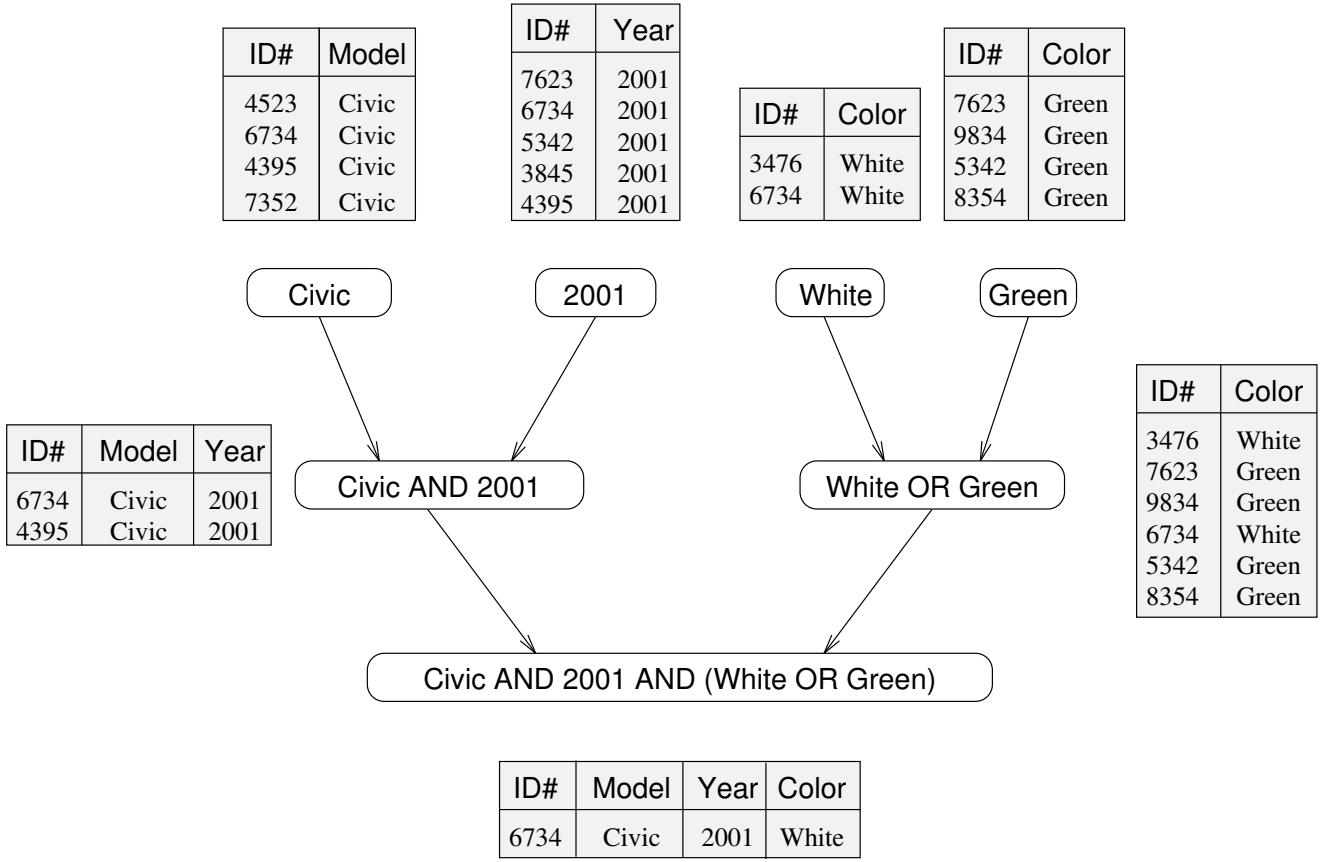
```
MODEL = ‘‘CIVIC’’ AND YEAR = 2001 AND
(COLOR = ‘‘GREEN’’ OR COLOR = ‘‘WHITE)
```

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

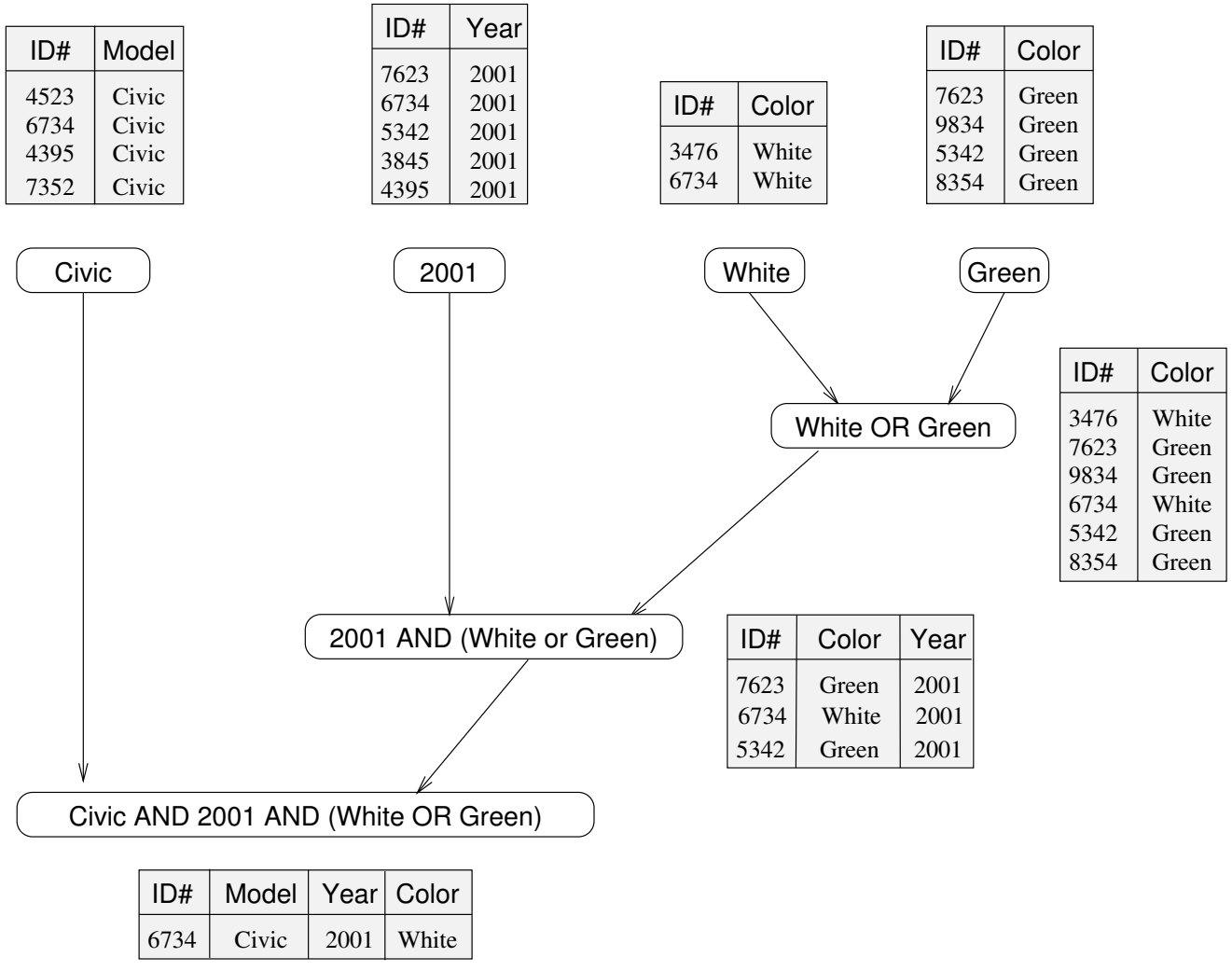
Decomposition into tasks

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



Decomposing the given query into several tasks. Edges denote that the output of one task is needed to accomplish the next.

Another decomposition

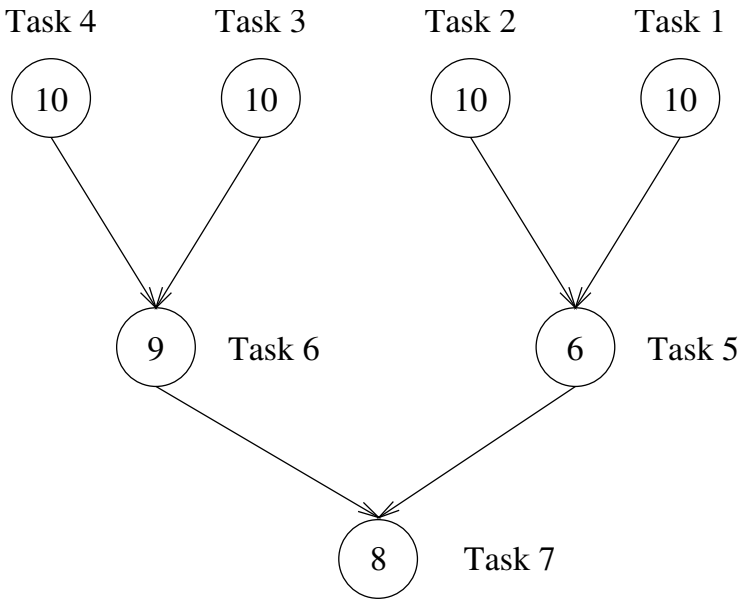


Different task decompositions may lead to significant differences with respect to their eventual parallel performance.

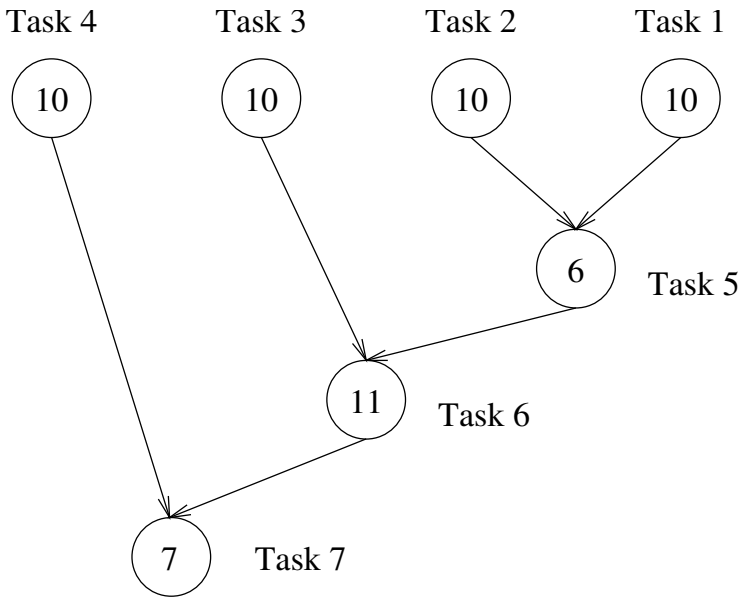
Task dependency graph & critical path

Task dependency graph: A directed path in the task dependency graph represents a sequence of tasks that must be processed one after the other.

The length of the longest path in a task dependency graph is called the critical path length. It also gives the minimum time needed by parallel execution.



(a)



(b)