Unit 2.1: Understanding Parallelism I

Video lesson 2 – Expressing tasks

Eduard Ayguadé, Josep Ramon Herrero,
Daniel Jiménez and Gladys Utrera

Barcelona Supercomputing Center
Universitat Politècnica de Catalunya

Motivation – I

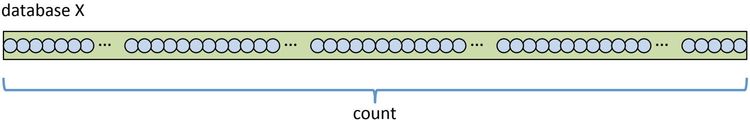
Let's go back to the car retail database example.

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

And assume that we want to count how many `Green` cars are available to sell.

Motivation – I

- One could traverse all the records `X[0] ... X[n-1]` in the database `X` and check if the `Color` field matches the required value `Green`, storing the number of matches in variable `count`



Motivation – I

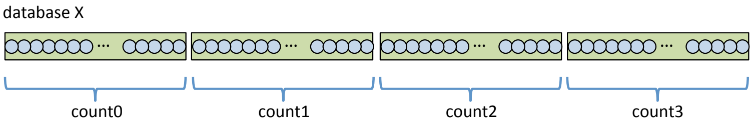
- A possible sequential program could be:

```
count = 0;
for ( i = 0 ; i < n ; i++ )
    if (X[i].Color == "Green") count++;
```

whose computation time on a single processor would be proportional to the number of records in the database $T_1 \propto n$

Tasks

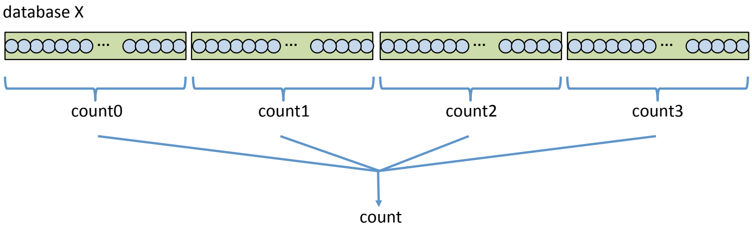
- One could divide the traversal in P groups (tasks), for example for $P = 4$:



checking the `Color` field for a subset of $n \div P$ consecutive records, and counting on a per–task "private" copy of variable `count`

Tasks

- However, we still need to "globally" count the number of records found that match the condition by combining the individual "private" counts into the original `count` variable



Tasks

- Up to this point you could anticipate that the computation time would be divided by the number of tasks P if P workers are used to do the computation

$$T_P = T_1 \div P$$

with an additional "overhead" to perform this global reduction

$$T_p = T_1 \div P + T_{ovh}(P)$$

probably proportional to the number of workers P

Motivation – II

Consider now that we want to execute the following query:

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

on our car dealer 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

Tasks

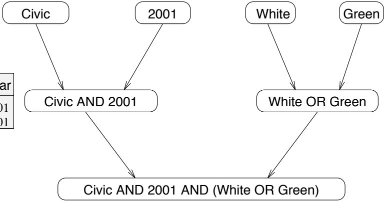
- A possible query plan could be:

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

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

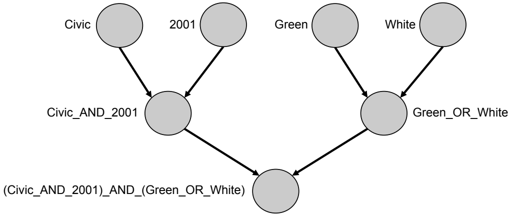
Barcelona Supercomputing Center Centro Nacional de Supercomputación

Tasks and dependences

- Each of these operations in the query plan could be a task, each computing an intermediate table of entries that satisfy particular conditions
- Are they independent?
  - Some of them are, for example tasks *"Civic"*, *"2001"*, *"Green"* and *"White"*
  - Others are not independent. For example, task *"Civic_AND_2001"* can not start its execution until both tasks *"Civic"* and *"2001"* complete

Tasks and dependences

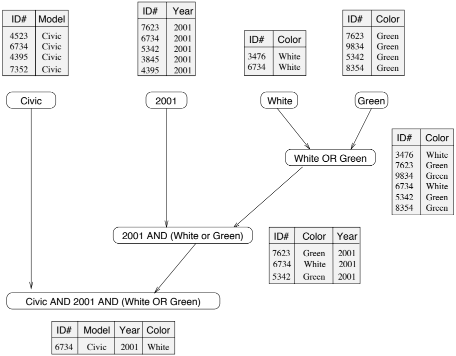
- Dependences impose task execution ordering constraints that need to be fulfilled in order to guarantee correct results



- **Task dependence graph**: graphical representation of the task decomposition

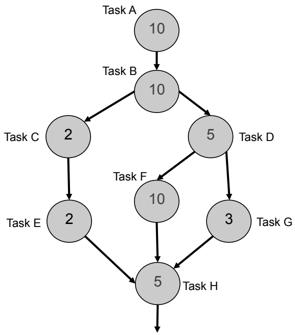
Tasks and dependences

- Other query plans are possible, for example



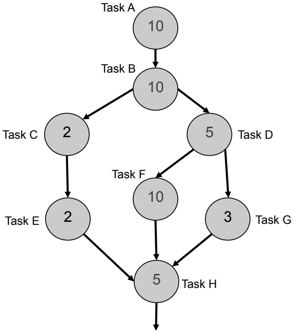
... with different task dependence graphs and potential to execute tasks in parallel

Task graph



- **Task dependence graph** abstraction
  - Directed Acyclic Graph
  - Node = task, its weight represents the amount of work to be done
  - Edge = dependence, i.e. successor node can only execute after predecessor node has completed

Computing the parallelism

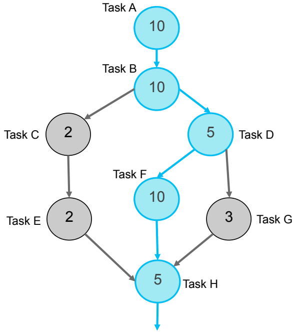


- Parallel machine abstraction
  - P identical processors
  - Each processor executes a node at a time
- $T_1 = \sum_{i=1}^{nodes}(work\_node_i)$

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

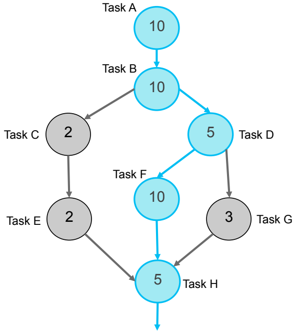
Barcelona Supercomputing Center Centro Nacional de Supercomputación

Computing the parallelism



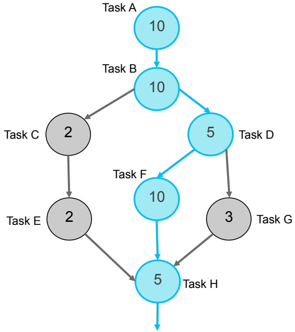
- Critical path: path in the task graph with the highest accumulated work
- $T_\infty = \sum_{i \in criticalpath} (work\_node_i)$, assuming sufficient processors

Computing the parallelism



- $Parallelism = T_1/T_\infty$, if sufficient processors were available
- P_{min} is the minimum number of processors necessary to achieve $Parallelism$

Computing the parallelism



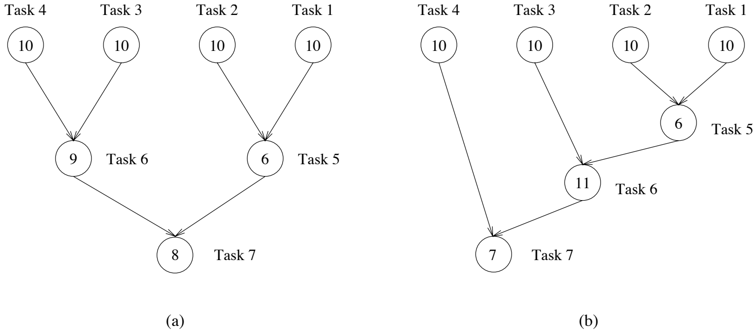
- $T_1 = 47$ including tasks {ABCDEFGH}
- Possible paths:

  {ABCEH} with total cost 29
  {ABDFH} with total cost 40
  {ABDGH} with total cost 33

- $T_\infty = 40$ for critical path {ABDFH}
- $Parallelism = 47/40 = 1.175$
- $P_{min} = 2$

Going back to our database query example

Consider the task dependency graphs for the two database queries, assuming *work_node* is proportional to the number of inputs to be processed



Which are T_1, T_∞ and $Parallelism$ in each case?

Going back to our database query example

- For graph (a) the critical path is determined by any of the two task sequences {3, 6, 7} or {4, 6, 7}

$$T_1^{(a)} = 63, T_\infty^{(a)} = 27, Parallelism^{(a)} = 63/27 = 2.33$$

- However, for graph (b) the critical path is determined by any of the two task sequences {1, 5, 6, 7} or {2, 5, 6, 7}

$$T_1^{(b)} = 64, T_\infty^{(b)} = 34, Parallelism^{(b)} = 64/34 = 1.88$$

- Do both query plans require the same minimum number of processors P_{min} to achieve this potential $Parallelism$?

Granularity and parallelism

- The granularity of the task decomposition is determined by the computational size of the nodes (tasks) in the task graph

- Example: counting matches in our car dealer database

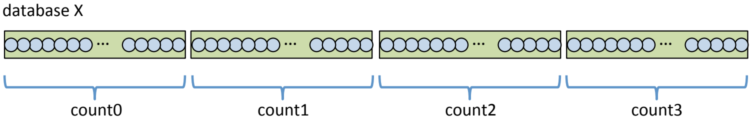
```
count = 0;
for ( i=0 ; i< n ; i++ )
    if (X[i].Color == "Green") count++;
```

Coarse-grain decomposition:
The whole loop is a task
Parallelism = 1

Fine-grain decomposition:
Each iteration of the loop is a task
Parallelism = n

Granularity and parallelism

- Example: A task could be in charge of checking a number of consecutive elements m of the database:



with a potential *parallelism* $= n \div m$

$$1 < (n \div m) < n$$

Granularity and parallelism

- It would appear that the parallelism is higher when going to fine-grain task decompositions

	Coarse grain	Fine grain	Medium grain
Number of tasks	1	n	n / m
Iterations per task	n	1	m
Parallelism	1	n	n / m

- However, tradeoff between potential parallelism and overheads related with its exploitation (e.g. creation of tasks, synchronization, exchange of data, ...)

Unit 2.1: Understanding Parallelism I

Video lesson 2 – Expressing tasks

Eduard Ayguadé, Josep Ramon Herrero,
Daniel Jiménez and Gladys Utrera

Barcelona Supercomputing Center
Universitat Politècnica de Catalunya