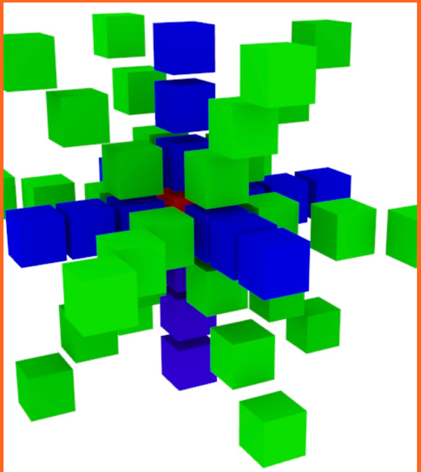
CS-E4690 – Programming parallel supercomputers

Designing parallel algorithms (EXTRA)

Maarit Korpi-Lagg

maarit.korpi-lagg@aalto.fi

Aalto University
School of Science

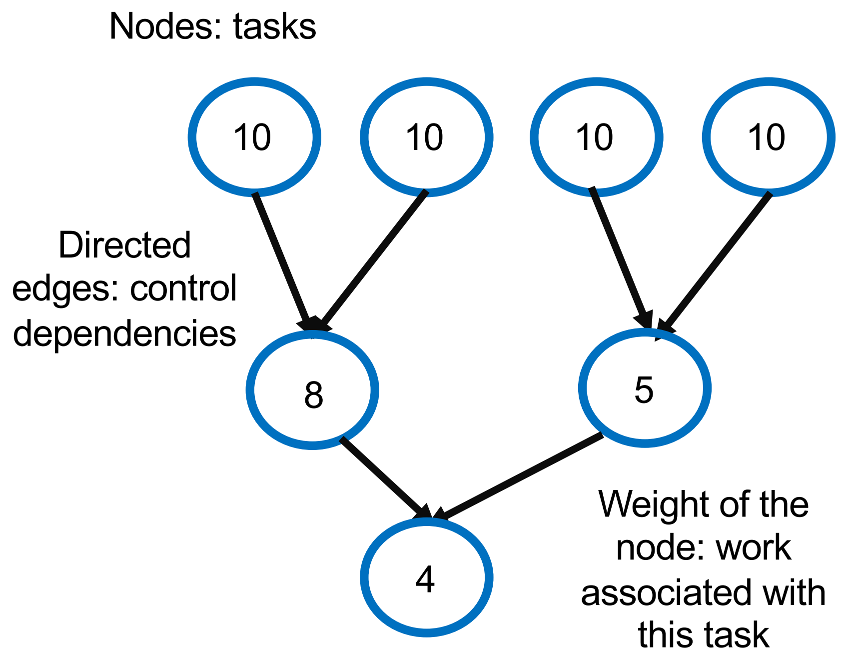


How to design a parallel algorithm?

- Determine **which parts** of your code can be computed **concurrently**
- **Decompose** these parts to **smaller pieces** that can be computed concurrently== ***tasks***
- **Map** the obtained tasks to a "virtual" topology of ***processes***, and **optimize** configuration
  - Maximize concurrency (Task dependency graphs) by mapping independent tasks onto different processes
  - Minimize interactions (Task interaction graphs) by mapping tasks with high degree of mutual interactions onto the same process
  - Make sure that there are processes to execute the next task when a previous task completes.

Task dependency graph (TDG)

- **Optimum decomposition of the tasks for concurrency**



Critical path:

The longest directed path between any pair of start (no incoming edge) and finish (no outgoing edge) node

Critical path length:

Sum of weigths along critical path

Average degree of concurrency (to be maximized)

Total amount of work/critical path length

In the example: 57/22=2.59

Examples

Data base query; imaginary phone sales catalogue

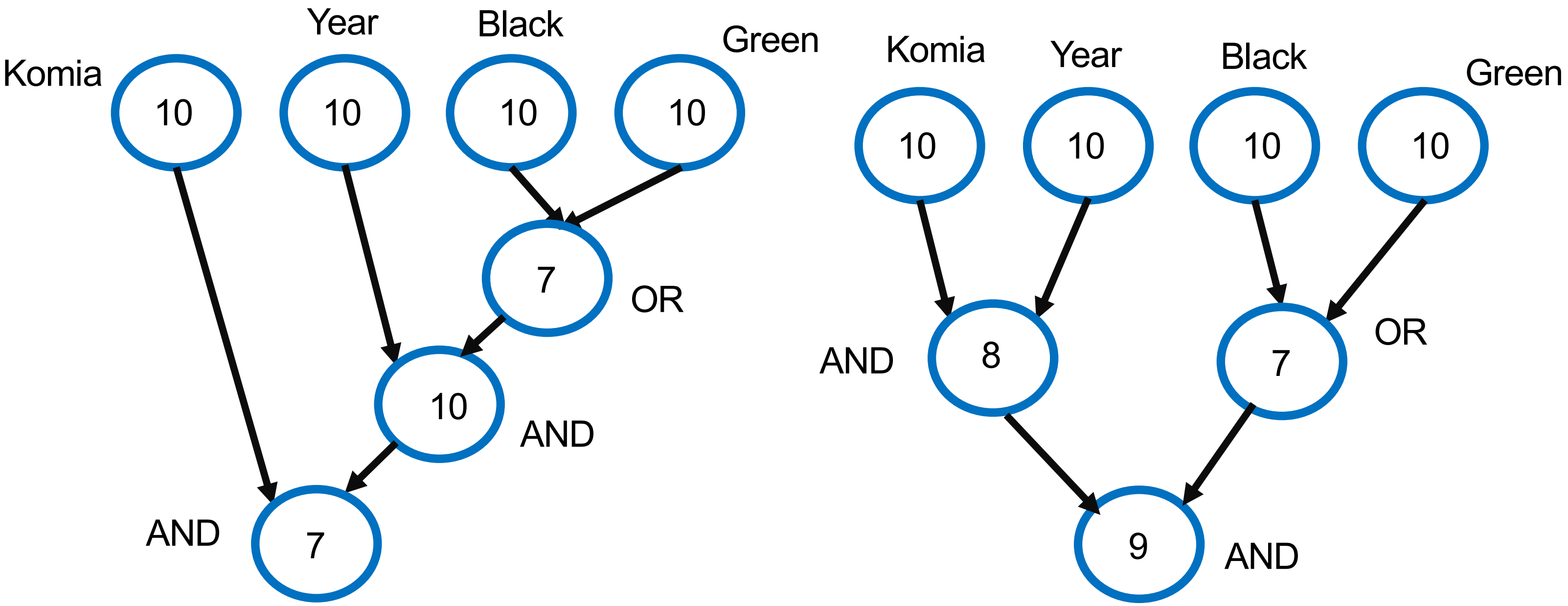
ID#	Year	Manufacturer	Model	Color	Retailer
23498	2018	Komia	Pulikka	Black	Kikantti
8734568	2019	OneMinus	6	Blue	Elise
265341	2017	Orange	10	Green	NDA
6743345	2019	Komia	Palikka	Black	Kikantti
3265	2016	OneMinus	6	Green	Elise
534876	2017	OneMinus	7	Red	NDA
762345	2019	Komia	Palikka	Green	Elise
34567	2020	Orange	11	Black	Kikantti
123867	2020	Komia	Pulikka	Blue	NDA
46556	2017	Komia	Palikka	Black	Elise

Query: Manufacturer="Komia" AND Year="2019" AND (Color="Black" OR Color="Green")

Examples

Data base query; imaginary phone sales catalogue

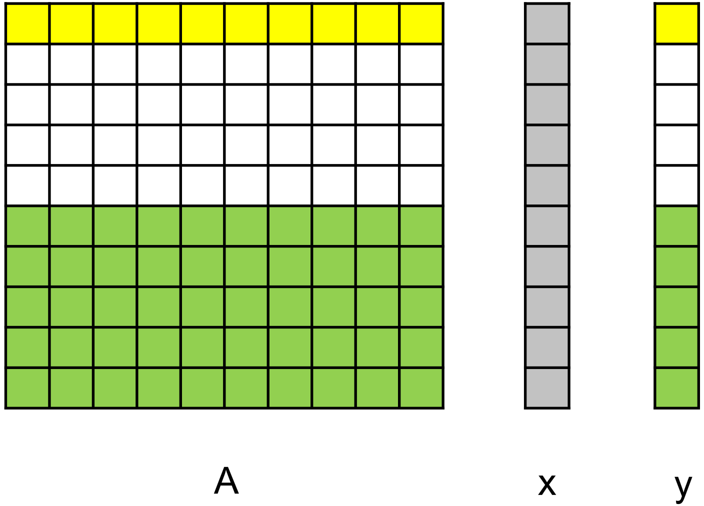
Query: Manufacturer=“Komia” AND Year=“2019” AND (Color=“Black” OR Color=“Green”)



Question; which is better?

Examples

Matrix-vector multiplication; y=Ax

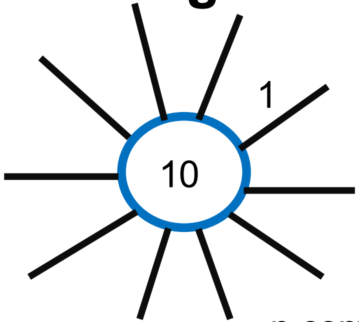


- All tasks are independent (no directed edges from nodes)
- Maximum concurrency according to TDG would be obtained by dividing to the smallest possible entity (one cell)
- Possibility to divide the work based on data in many different ways (for example to yellow or green blocks)
- No matter how you divide the work, **you will need totality of x** for all tasks to update an element of y

Task interaction graph (TIG)

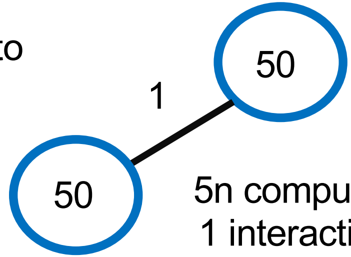
- Optimize data dependencies (minimize interactions)
- To decide what is the optimum granulation level of the decomposition
- Nodes represent tasks and their computation times
- (Un)directed edges the interactions in between them

Dense matrix case: task decomposition to one row; one task



n computations, n interactions of x

Decomposition to green blocks



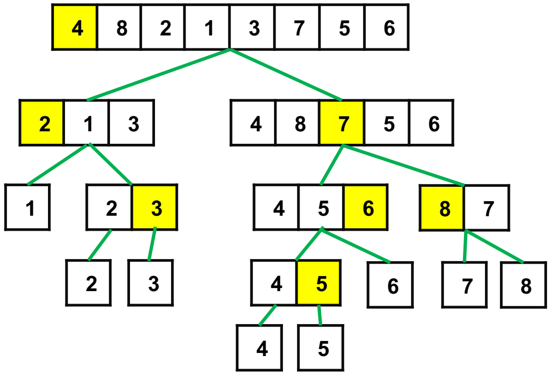
5n computations, 1 interaction of x

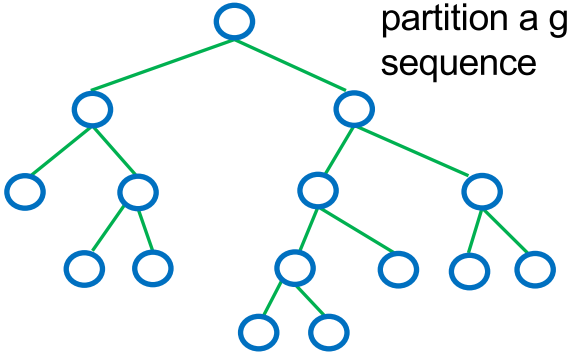
Decomposition

- **Task decomposition**
  - Recursive decomposition: “Divide and conquer”
- **Data decomposition (Input/Output/Intermediate/Hybrid)**
  - Input/Output: “Owner computes” model
- **(Exploratory)**
- **(Speculative)**

Recursive decomposition

- Decompose a problem into independent sub-problems
- Decompose sub-problems similarly using recursion
- Stop decomposing, when the granularity becomes sub-optimal or result is obtained
- Typical example: Quicksort

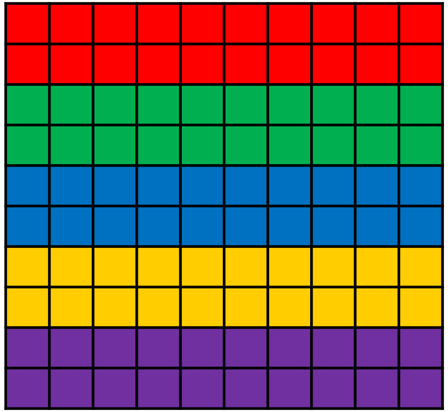


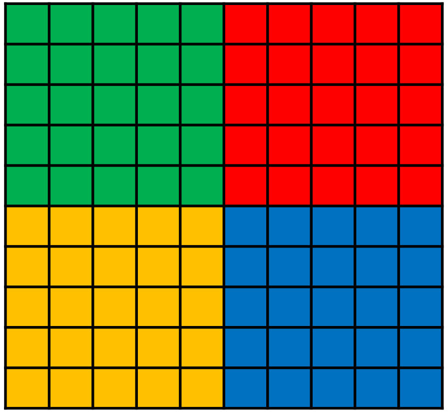


Task==work to partition a given sub-sequence

Data decomposition

- Manipulation of large data sets; matrix-vector multiplication was one good example
- Define tasks based on partitioning the data
- Output/Input/Intermediate/Hybrid





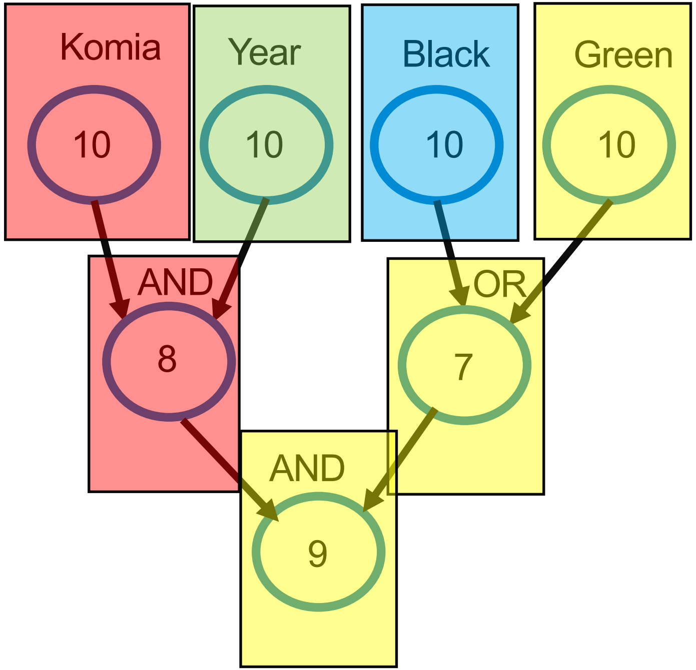
......

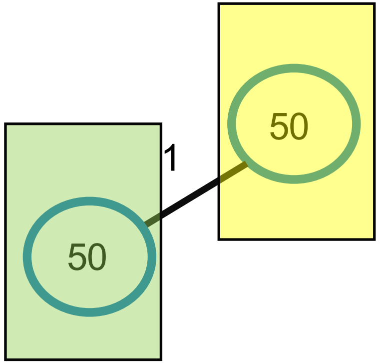
How to map tasks to processes?

- Process is a logic entity performing the defined tasks
- Let us look at our example cases

How to map to processes?

Data base query; best case concurrency-wise

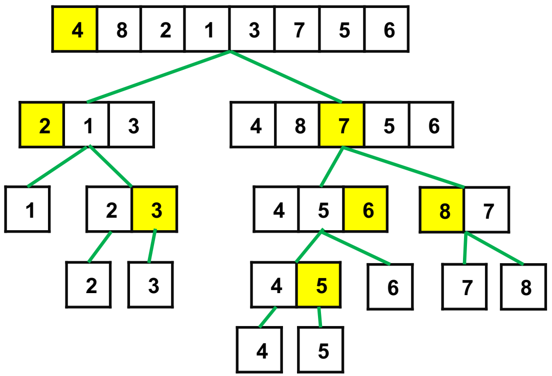


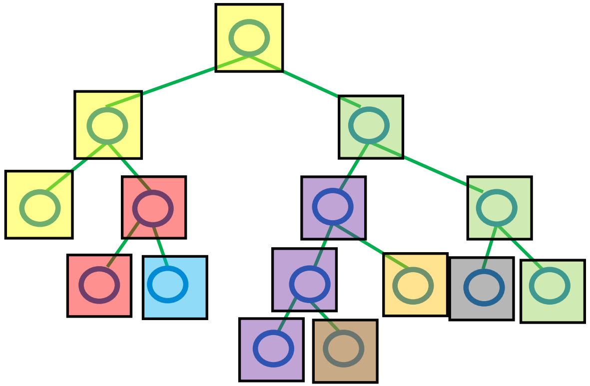


Dense matrix-vector multiplication; If we decompose the data into two row-wise blocks, we can map them to two concurrent processes

How to map to processes?

Quicksort; tree-like mapping





Static versus dynamic tasks and mapping

- Dense matrix multiplication is suited for *static* task generation and mapping (no need to change them when repeating the operation for different data sets)

- Database query and sorting, for example, are suited for *dynamic* task generation and mapping (with a changing query, the optimal graphs will change)

- Task depency graph is fixed for static, not known a priori for the dynamic case

Regular versus Irregular interactions

- Dense matrix multiplication has *regular* interactions (communication pattern between tasks repeats)

- If the matrix was sparse but did not possess any symmetry properties, then its communication pattern would become *irregular* (communication pattern would become dependent on where the zeros are in the matrix).

- Task interaction graph is fixed for regular, not known a priori for the irregular case
- Interactions can also be static and dynamic.

What to do in practice?

- **Static and regular mappings are “trivial” cases for MPI.**
- **Dynamic and irregular mappings are the challenge**
  - MPI can handle dynamicism with spawning more processes when needed (MPI_Comm_spawn and related functions); tedious
    - Also the way for implementing fault tolerance in MPI-4 standard; not ubiquituously available, hence we skip this year
  - MPI + openMP programming model; less tedious