Parallel Algorithmic

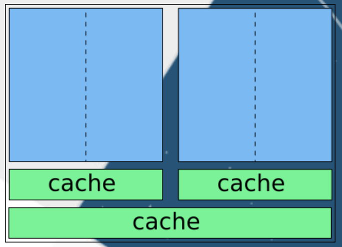
CSC5001 – Systèmes Hautes Performances

Objectives

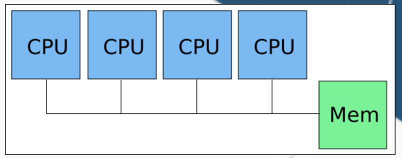
- What is the potential performance gain if I parallelize an application ?
- What is the cost of a network communication ?
- How to distribute data ?
- How to balance the workload ?

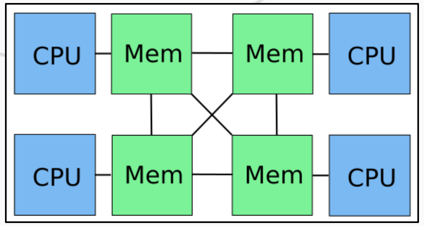
Parallel architectures

- Within a processor
  - Core, hyper-threading, superscalar CPU, vectorization

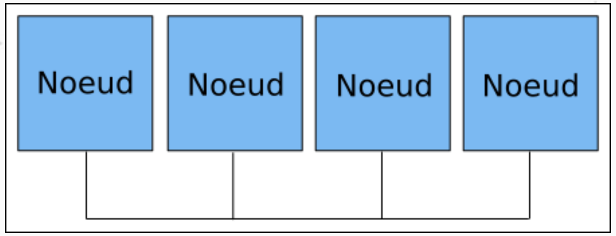


- Within a machine
  - SMP (Symmetric Multi Processor), NUMA (Non-Uniform Memory Architecture)





- Within a data center
  - Cluster of compute nodes



Programming models

Shared memory models

- Each task can access all the data
- Parallelization by distributing processing
- Bottleneck: inter-task synchronization (eg. Lock)
- Examples of shared memory models
  - OpenMP, Pthread, Intel TBB

Distributed memory models

- Each task access its own data
- Parallelization by distributing data and processing
- ‘’Owner computer’’: Each task compute the data it owns
- Bottleneck: inter-task communication (eg. network communication)
- Example of distributed memory models
  - MPI

Hybrid models

- Distributed memory model to distribute processing on several nodes
- Shared memory models within a node

- Take advantage of the cluster topology
  - 1 MPI process per NUMA node + OpenMP threads
  - 1 MPI process per machine + CUDA

Flynn taxonomy

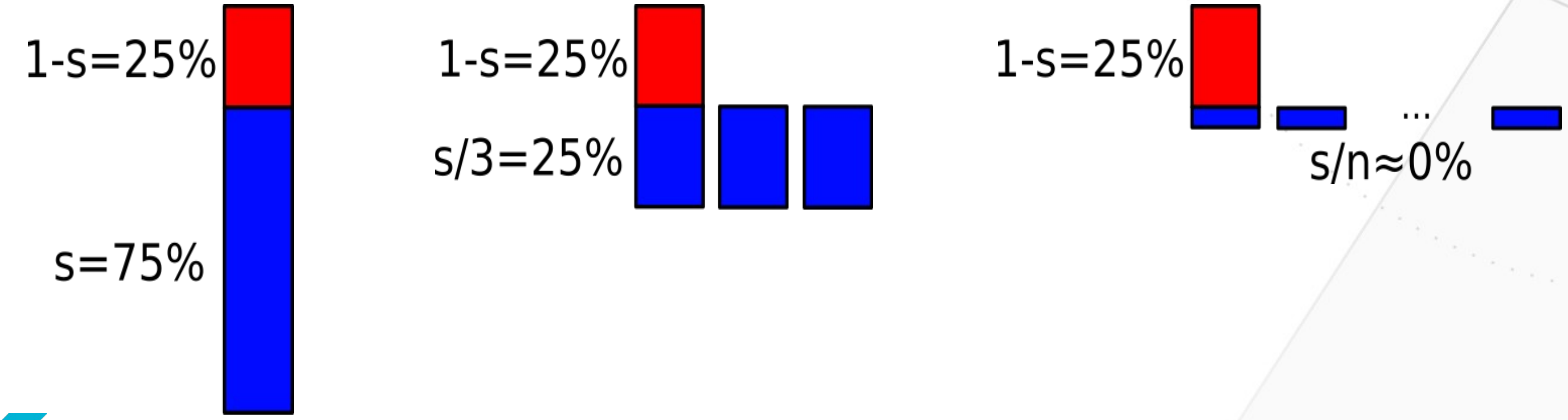
- Classification of computer architectures

	Single instruction	**Multiple instruction**	**Single program**	**Multiple programs**
Single data	SISD (sequential processorl)	MISD (aircrafts)		
Multiple data	SIMD (GPU, vector CPU)	MIMD (multicore, cluster)	SPMD (MPI)	MPMD (Cell/BE, CPU+GPU)

What is the potential performance gain if I parallelize an application ?

Theory of parallelism

- Parallelization
  - Use several processors to compute faster
  - Usually, only a part s of the program run in parallel

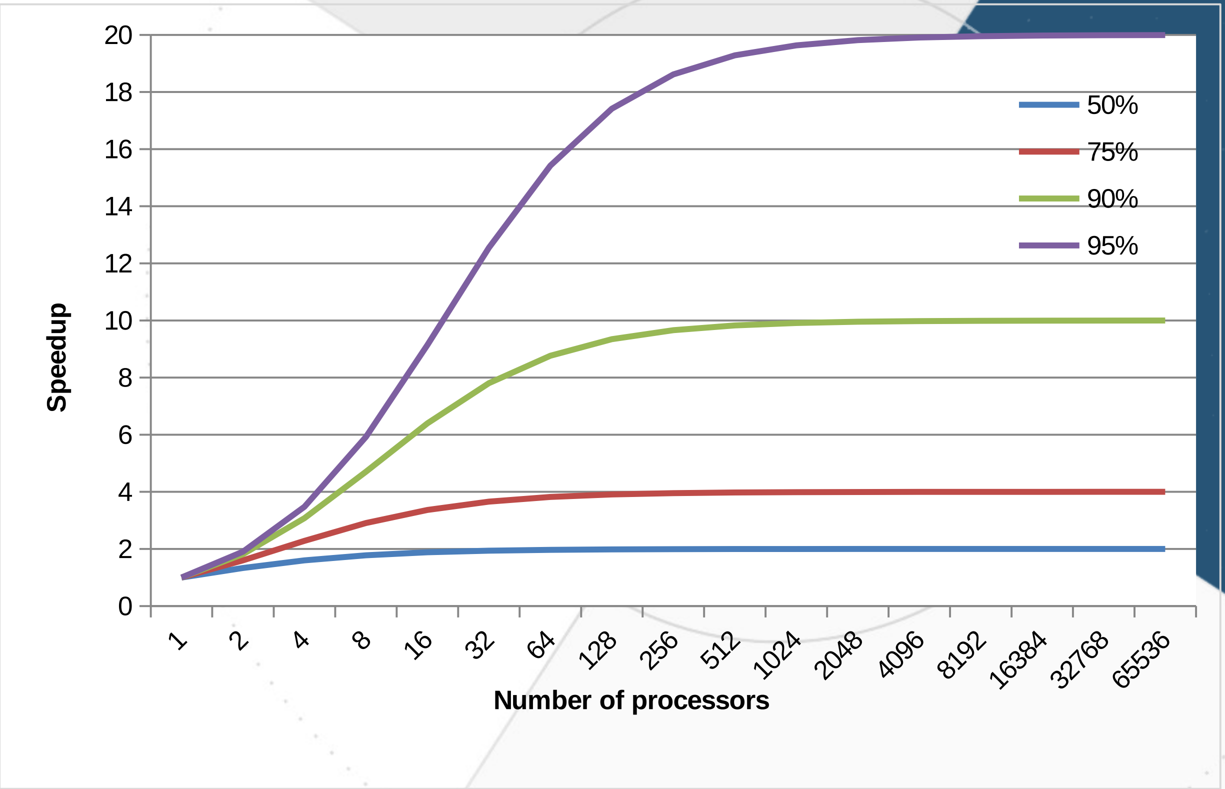


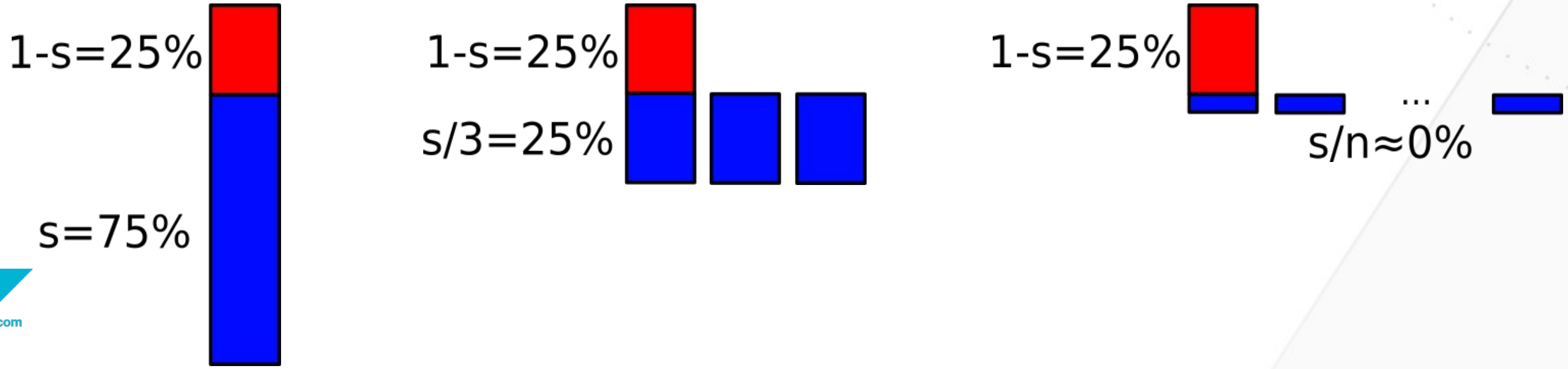
Measuring parallel performance

- Parallel performance metrics
  - **Speedup**: evolution of the execution time as the number of processors p increases
    - $S_p = T_s/T_p$
      - T_s: execution time of the best sequential algorithm
      - T_p: execution time of the parallel algorithm running on p processors
  - **Parallel efficiency**: evolution of the speedup as the number of processors p increases
    - $E_p = S_p/p$

Amdahl’s law

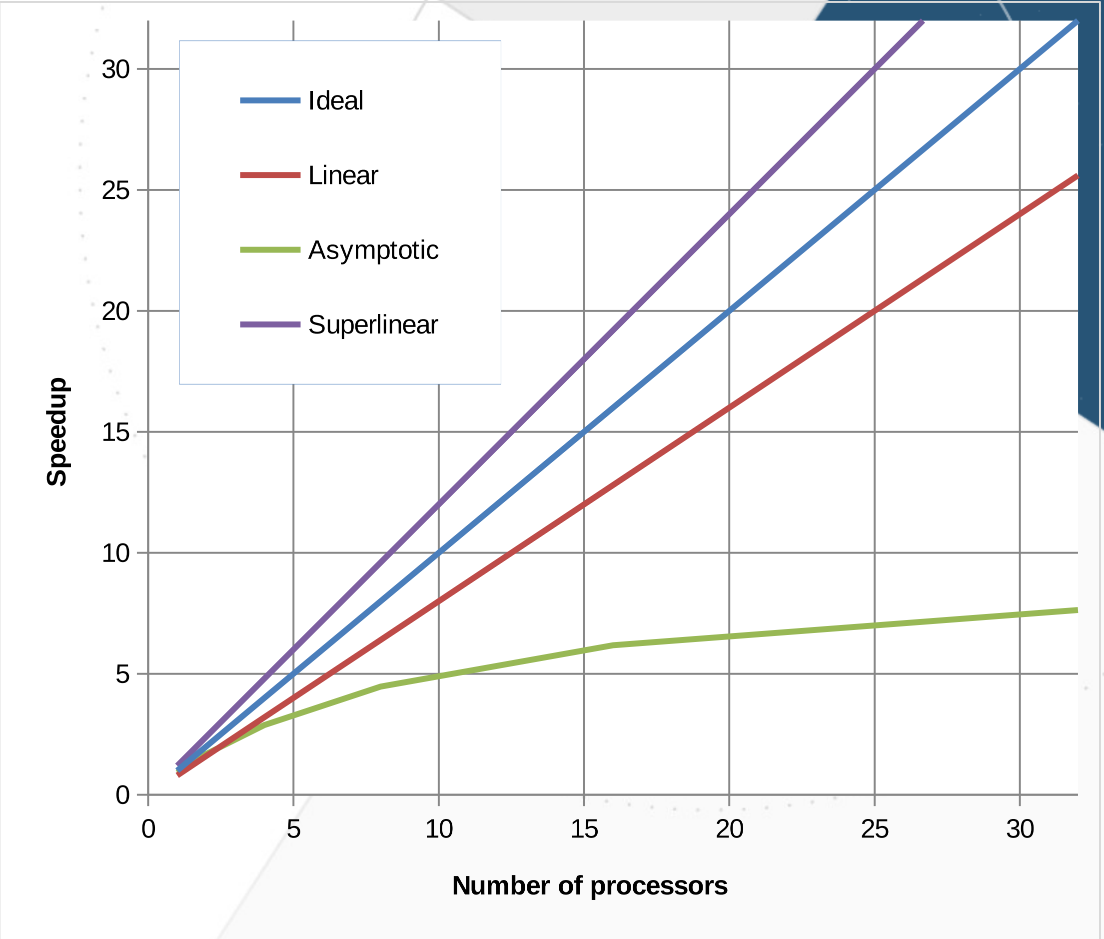
- Theoretical maximum speedup
- s = part of the program that is parallel
- 1-s = part of the program that is sequential
- r = 1 / (1-s) + (s/p)





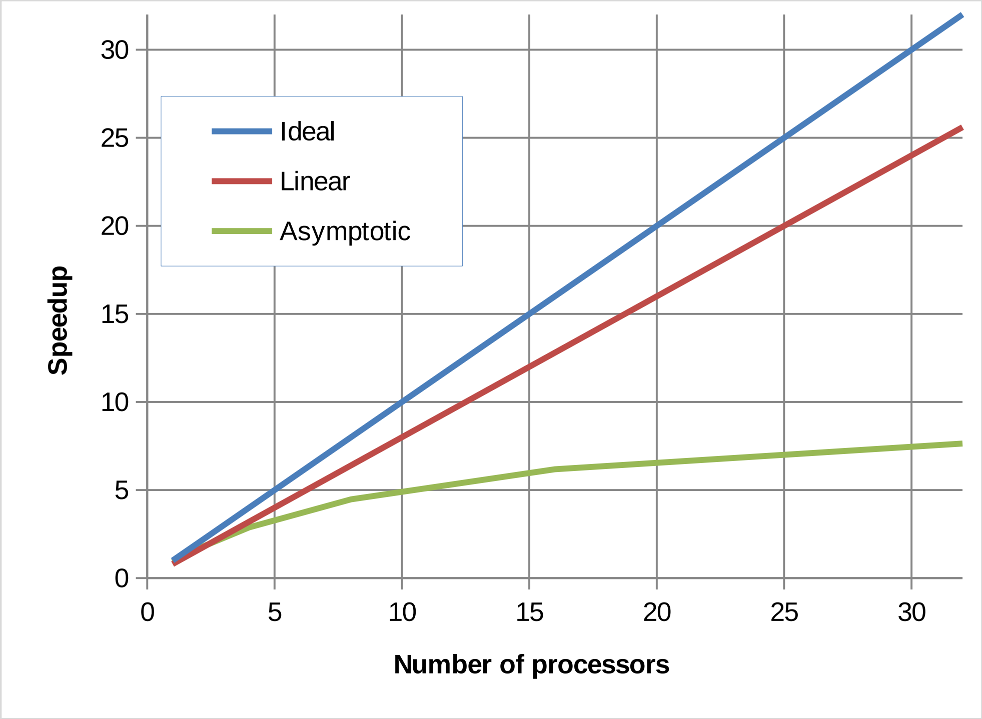
Speedup plots

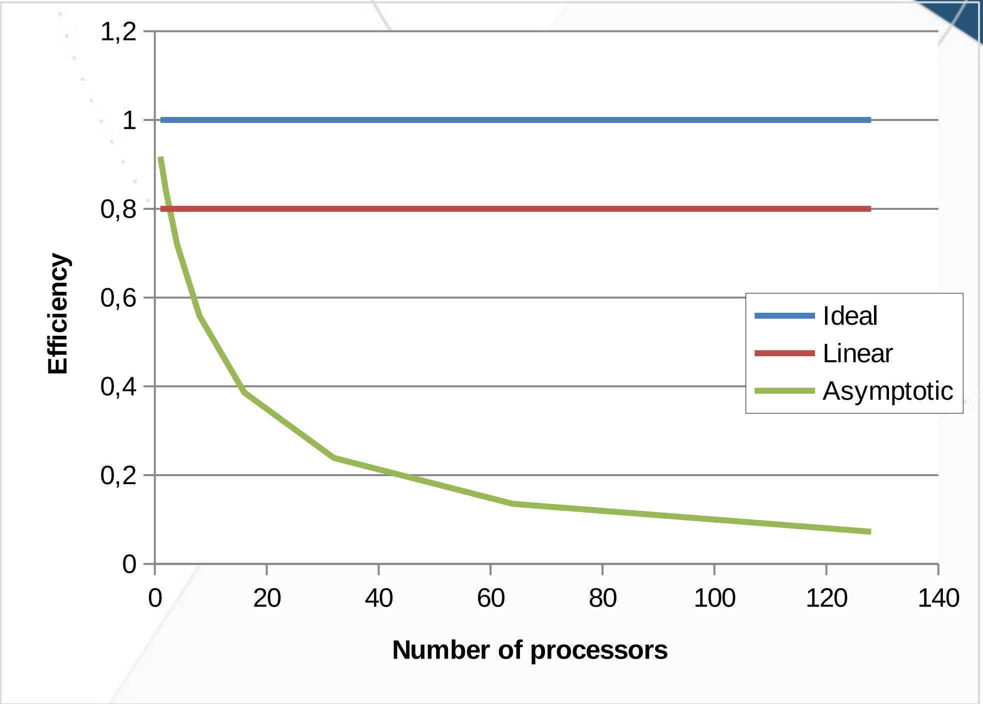
- Several classes of speedup exist
  - Ideal : $T_p = T_s/p$
  - Linear: $S_p = \alpha.S_i\ (\alpha<1)$
  - Asymptotic: $S_p < \beta$
  - Superlinear: $S_p > S_i$
    - Because of the architecture (eg. cache effects)
    - Because of the algorithme (eg. search algorithm)



Parallel efficiency

- Efficiency: $E = S/p$

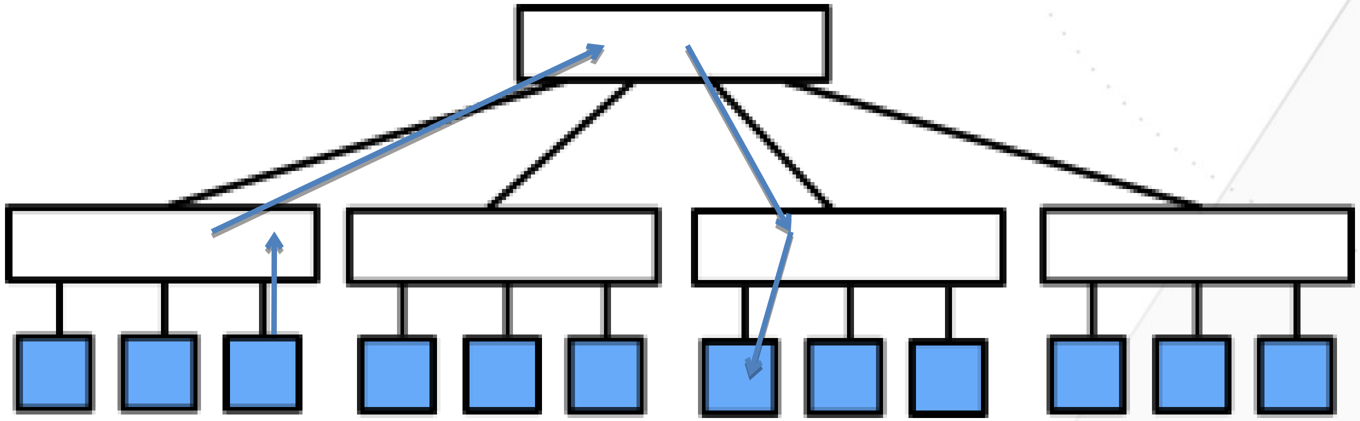




What is the cost of a network communication ?

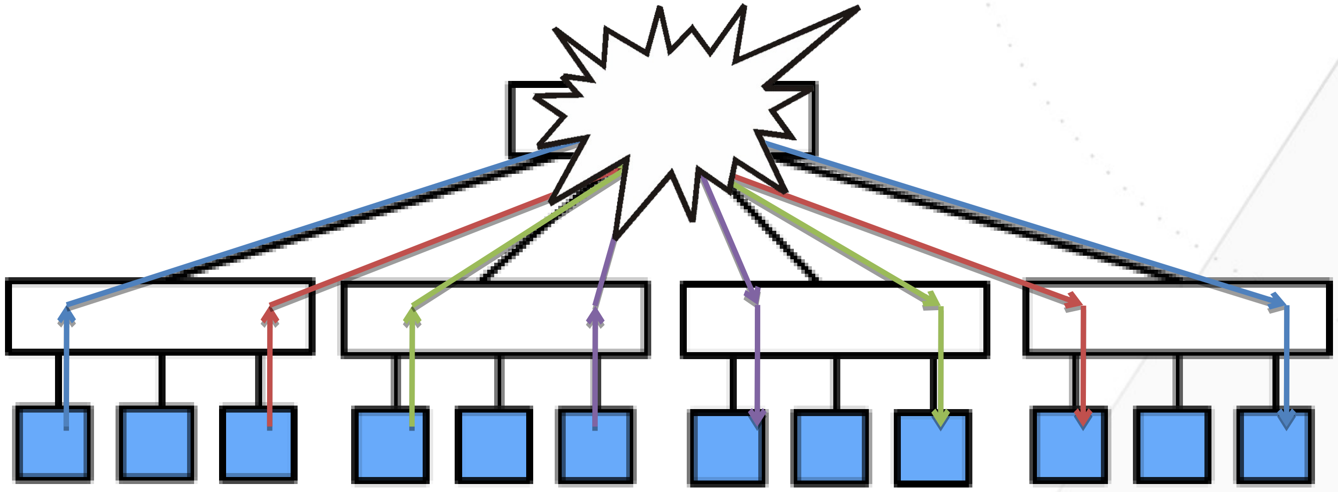
Network topologies

- How to connect N machines with 4-ports switches ?
  - Tree of switches



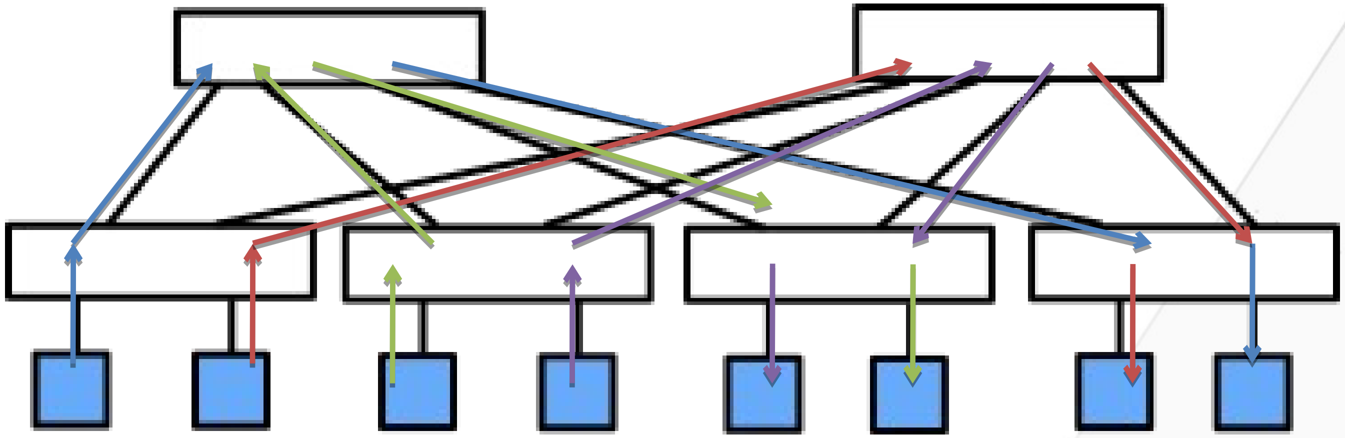
Network topologies

- How to connect N machines with 4-ports switches ?
  - Tree of switches



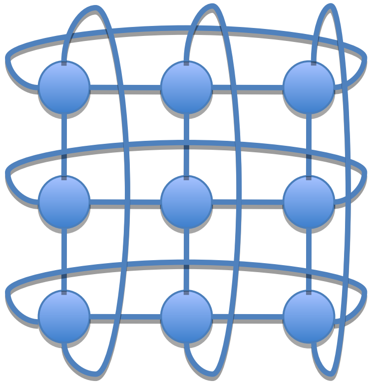
Fat tree

- How to connect N machines with 4-ports switches ?
  - Fat Tree

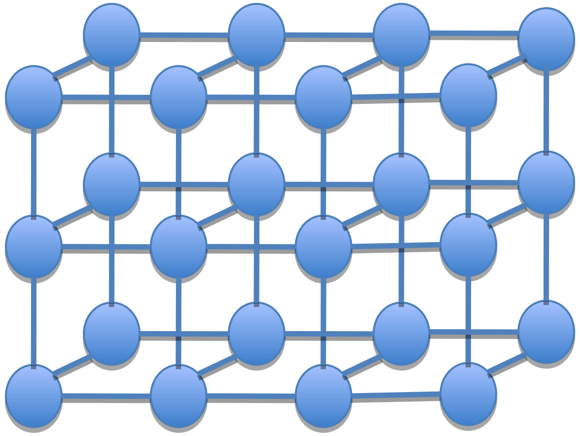


Other topologies

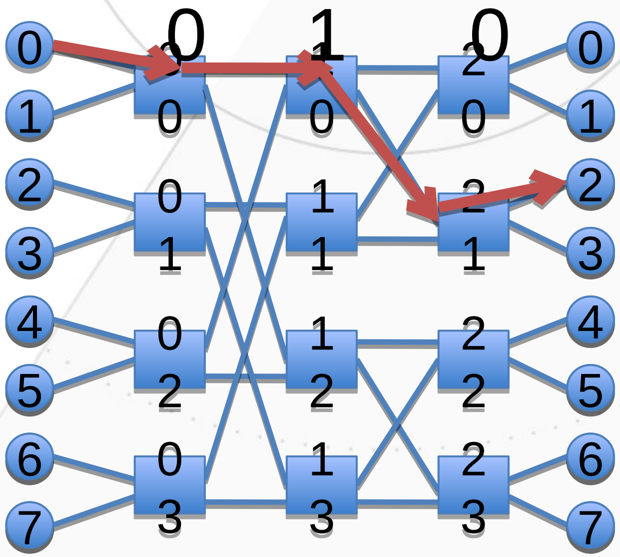
- Goal:
  - Minimize the number of hops (~latency)
  - Maximize throughput



Tore 2D



Tore 3D



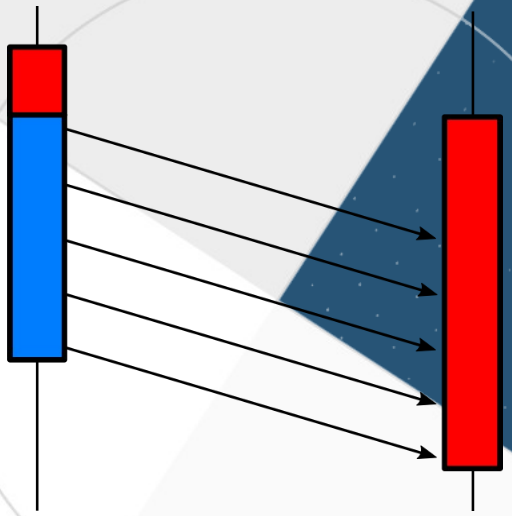
Butterfly

Communication models

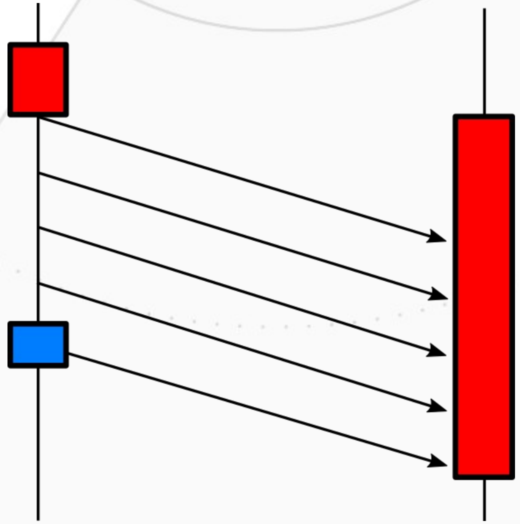
- Hypothesis
  - Communication cost (almost) constant for each pair of nodes
  - 1-port communication model
  - Full-duplex links
- Communication cost for a m-bytes word: $t_s + m\ T_w$
  - t_s: startup time
  - t_w: transfer time per word

Point to point communications

- Blocking communications
  - The sending thread blocks until the buffer can be modified
    - After the data is copied to another buffer,
    - Or after the end of the data transmission

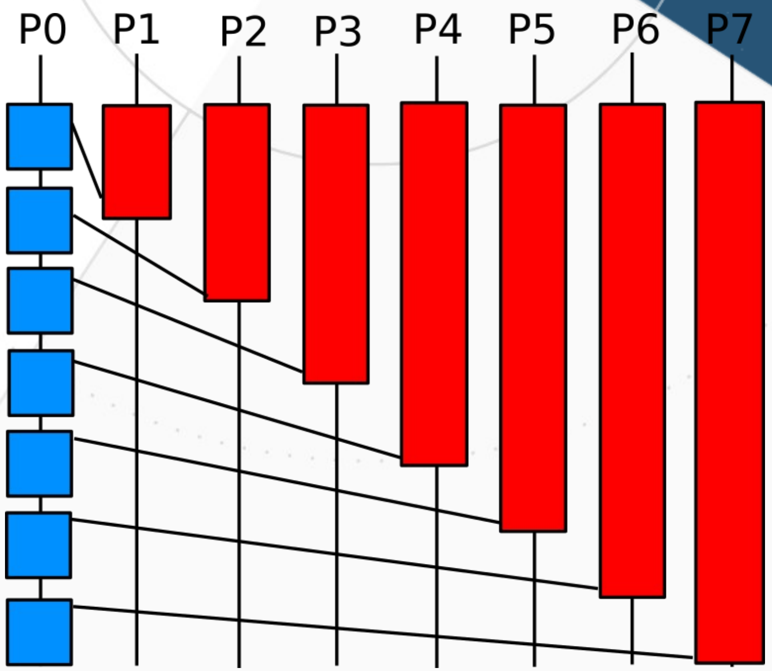


- Non-blocking communcations
  - The sending thread does not block while sending
  - The buffer can be modified after checking for the end of the data transfer
  - != asynchronous communication



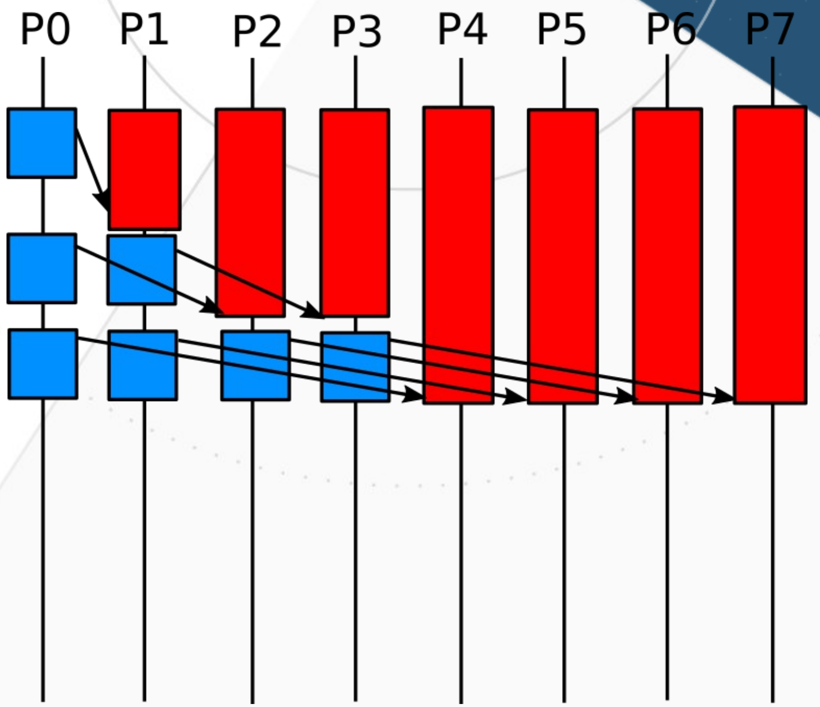
Collective communications

- Communication operation that involve a set of nodes
- Example: *1-to-n broadcast*
  - A *root* process broadcasts a m-bytes messages to the others
  - Naive algorithm:
    - The root process send the message to the other processes one by one
    - *n-1* steps
    - Execution time: $(n-1) \cdot (t_s + t_w.m)$



Collective communications

- Communication operation that involve a set of nodes
- Example: *1-to-n broadcast*
  - A *root* process broadcasts a m-bytes messages to the others
  - Other algorithms:
    - *log n* step
    - The optimal algorithm depend on the network topology
    - Execution time: *log n . (t_s + t_w.m)*



Exercise: all-to-all

- *All-to-all broadcast*
  - Every process broadcasts a m-bytes messages to the other processes of the group
- Exercise:
  - Write the all-to-all algorithm in pseudo-code

```
void all_to_all(int my_rank, message m, int m_size) {


}
```

  - Compute its execution time

Exercise : all-to-all

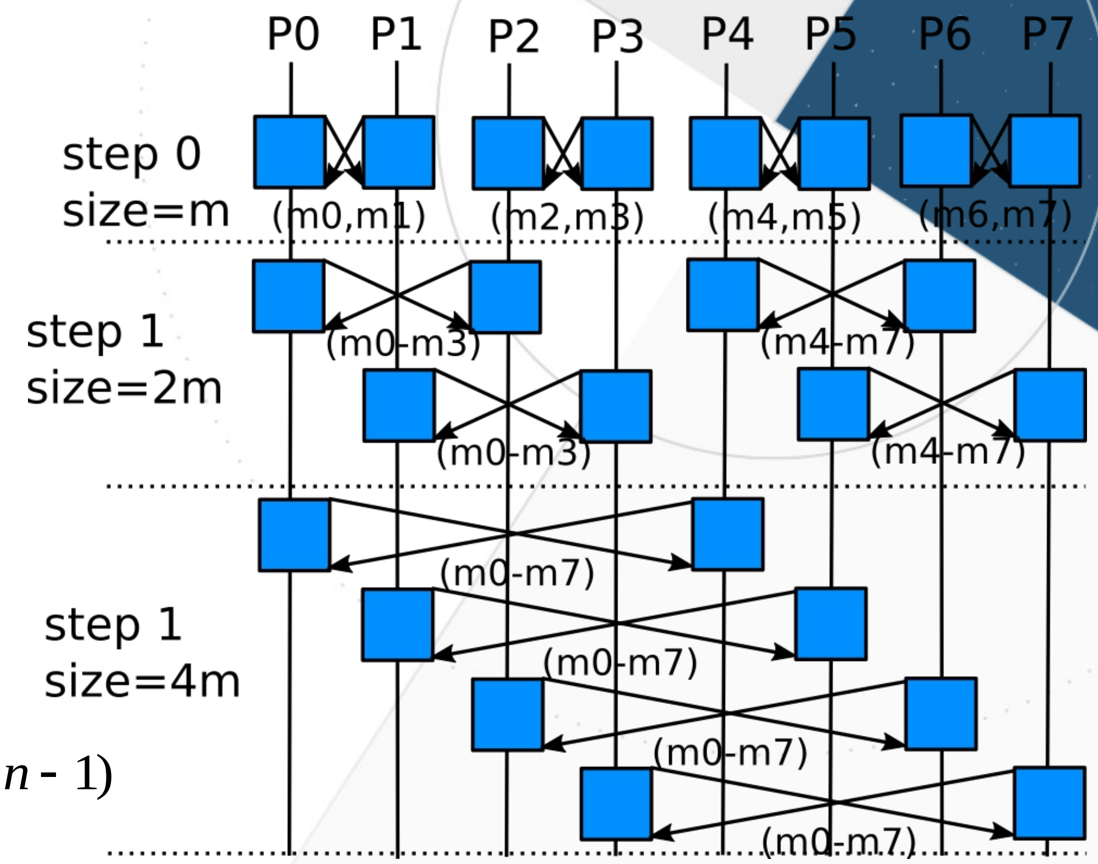
Solution

```
void all_to_all(int my_rank, message m, int m_size) {
  for(int i=0; i<log(n); i++) {
    int offset = 1<<i;
    int direction = my_rank & offset;
    int dest;

    if(direction == 0) {
      dest = my_rank + offset;
    } else {
      dest = my_rank - offset;
    }
    send(m, m_size, dest);
    recv(&m[m_size], m_size, dest);
    m_size *=2;
  }
}
```

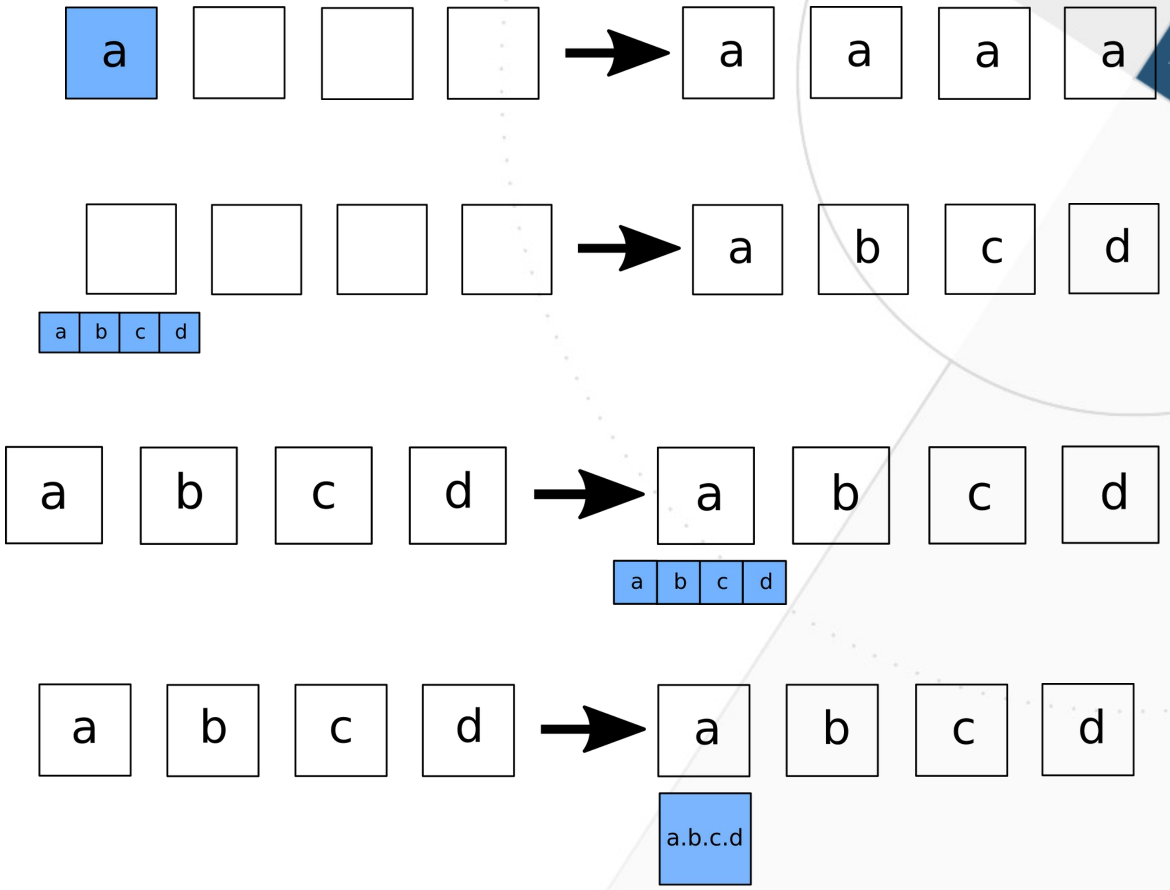
Execution time:

$$\sum_{i=0}^{\log n-1}(t_s + 2^i t_w m) = t_s \log n + t_w m(n-1)$$



Other collective communications

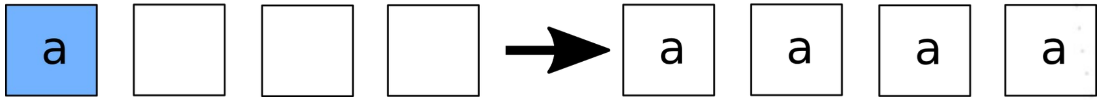
- *Broadcast*
- *Scatter*
- *Gather*
- *Reduce*



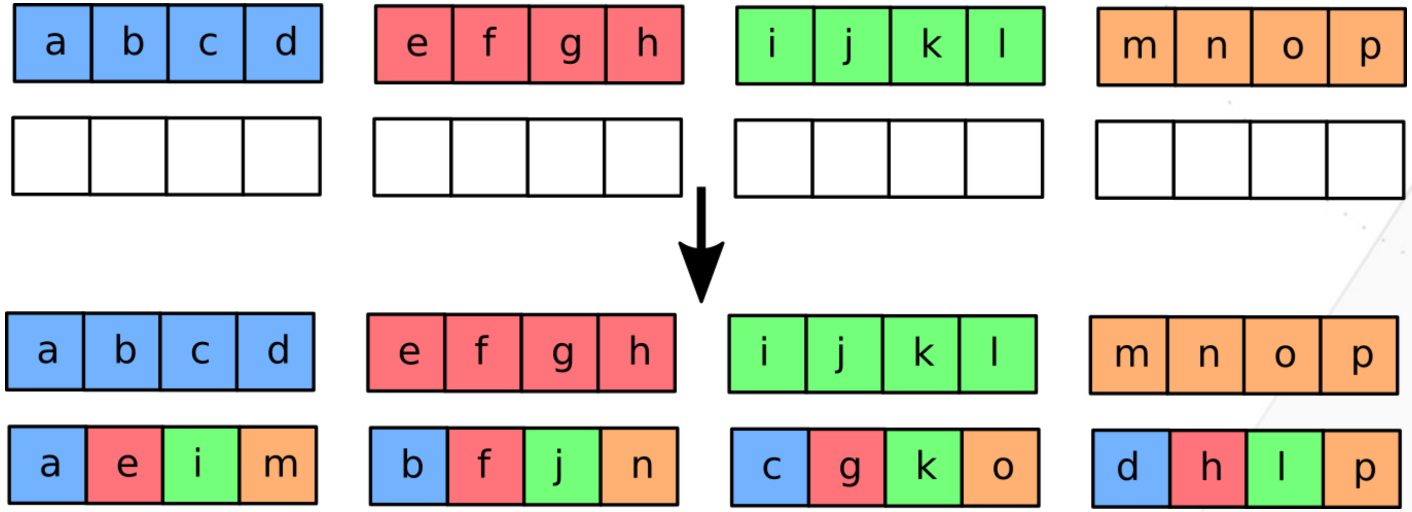
Other collective communications

all-to-all

- 1 to n *broadcast*



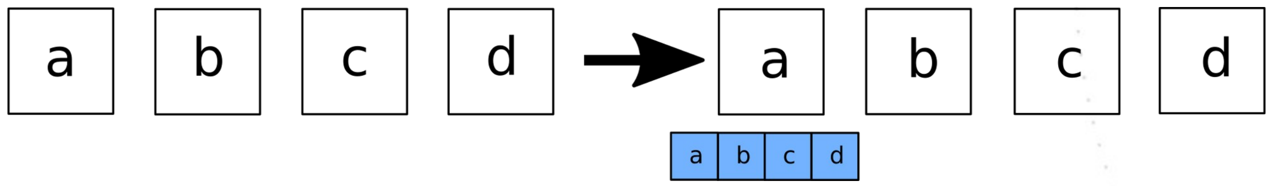
- n to n broadcast (AllToAll)



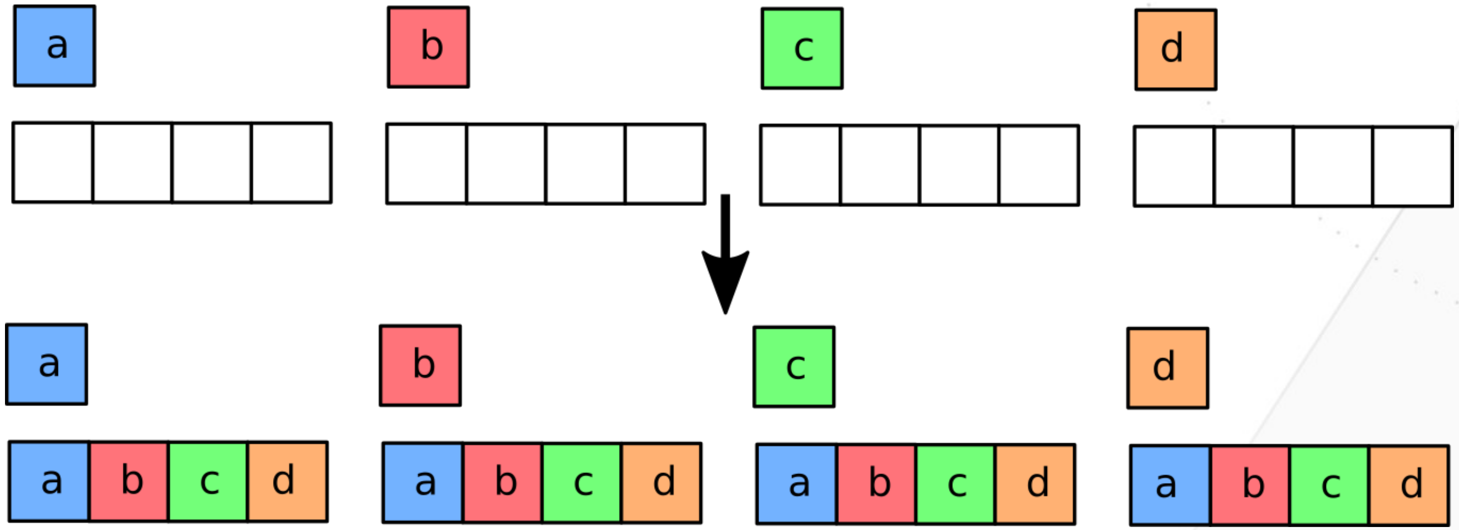
Other collective communications

all-to-all gather

- N to 1 *gather*



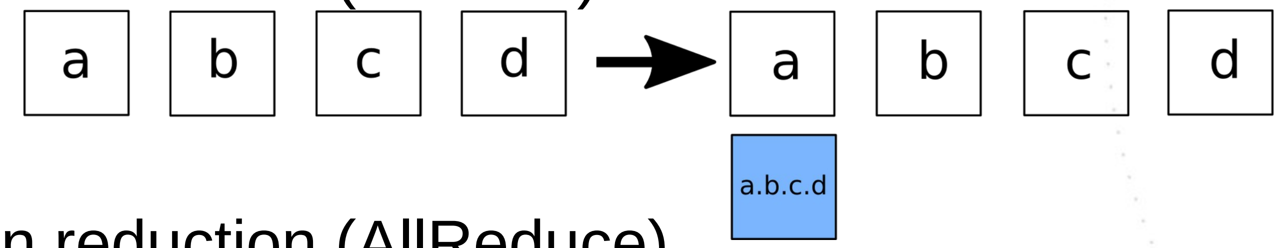
- n to n gather (AllGather)



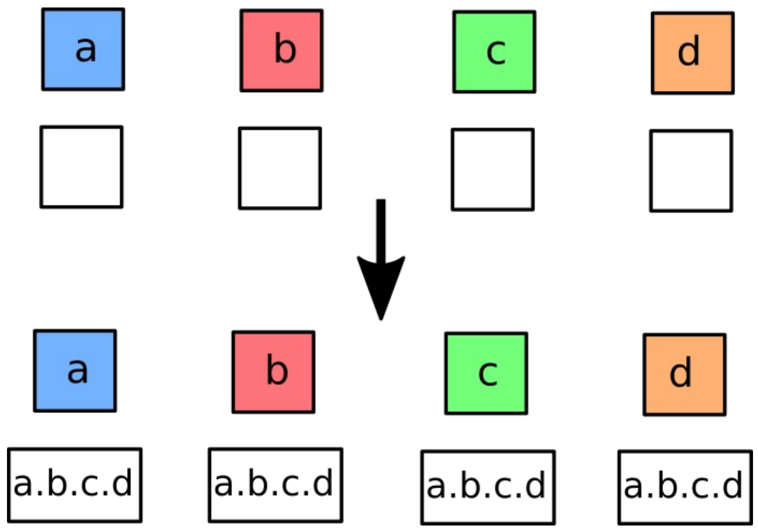
Other collective communications

all-to-all reduction

- n to 1 reduction (*Reduce*)



- n to n reduction (AllReduce)



Institut Mines-Télécom

How to distribute data ?

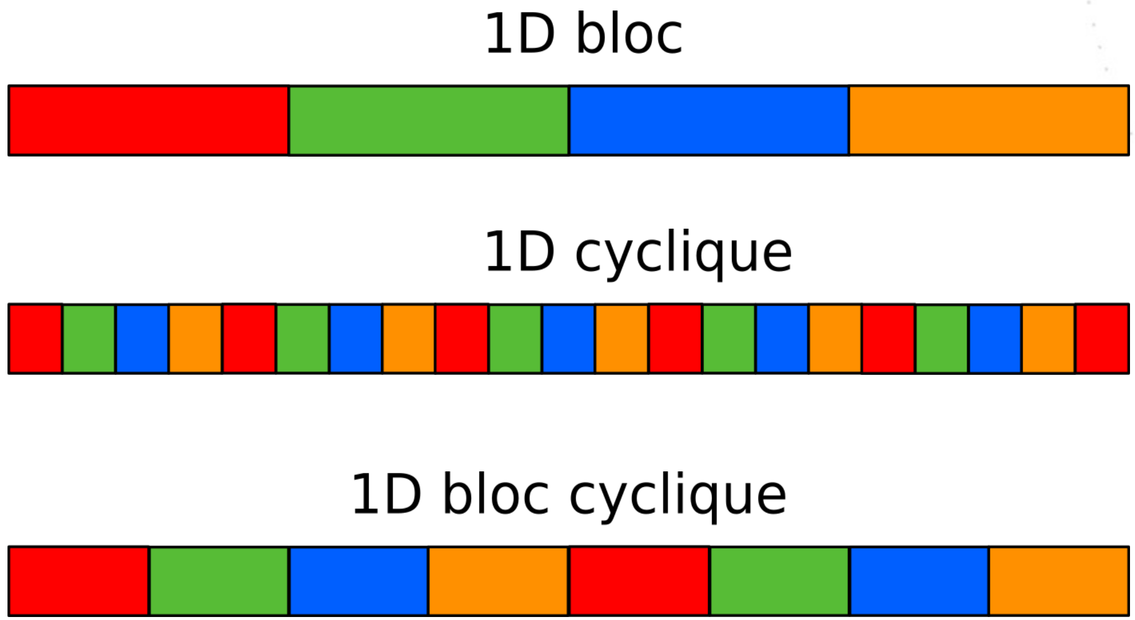
Data parallelism

- Parallelization based on data distribution
  - *Owner computes*

- A buffer can be distributed in several ways
  - A bad data distribution may generate spurious data transfers

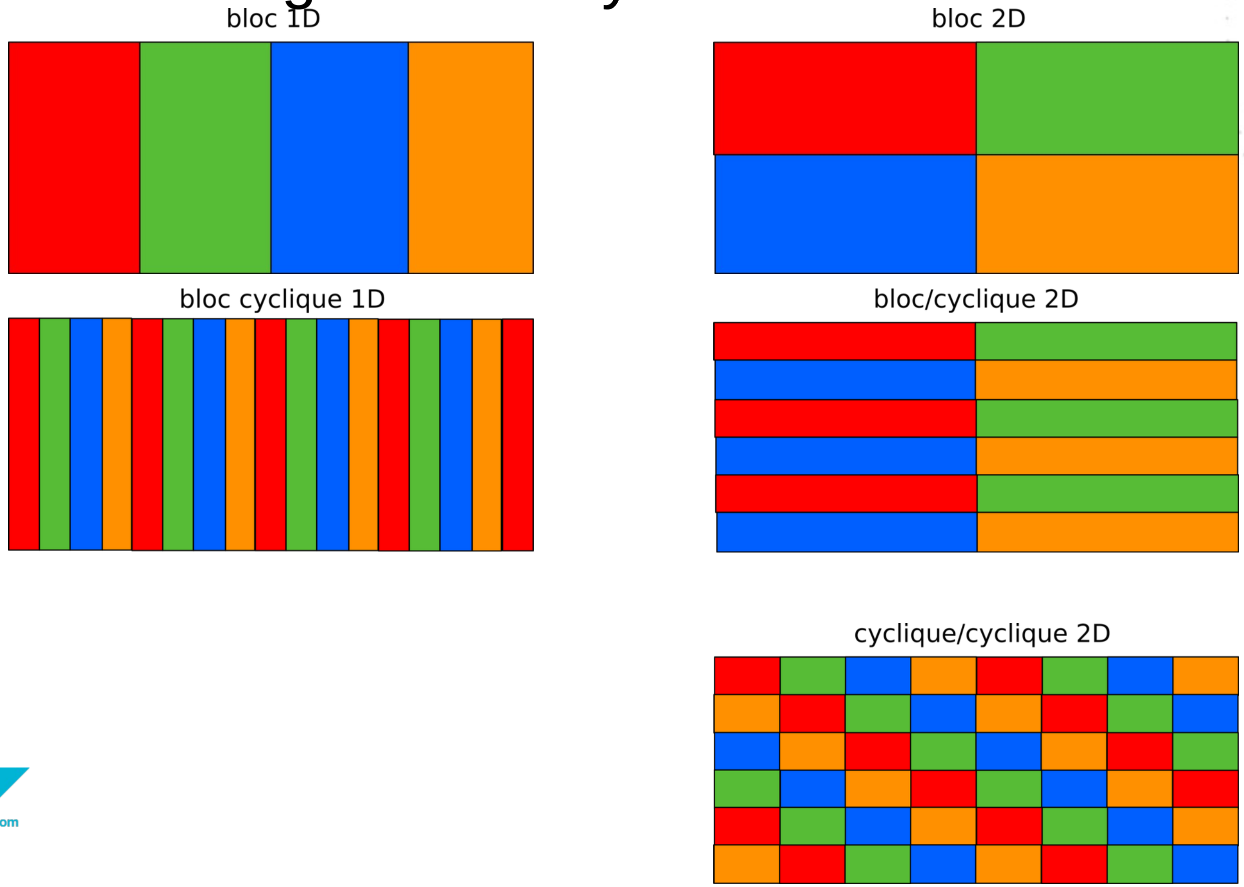
Distributing dense arrays

- Distributing a 1D array
  - block, cyclic, or block-cyclic distribution



Distributing dense arrays

- Distributing a 2D array



Exercise

- Multiplying NxN matrices
  - A x B = C
  - How to distribute matrices other 4 processes ?
  - Compute the memory footprint of matrices for each process

Exercise

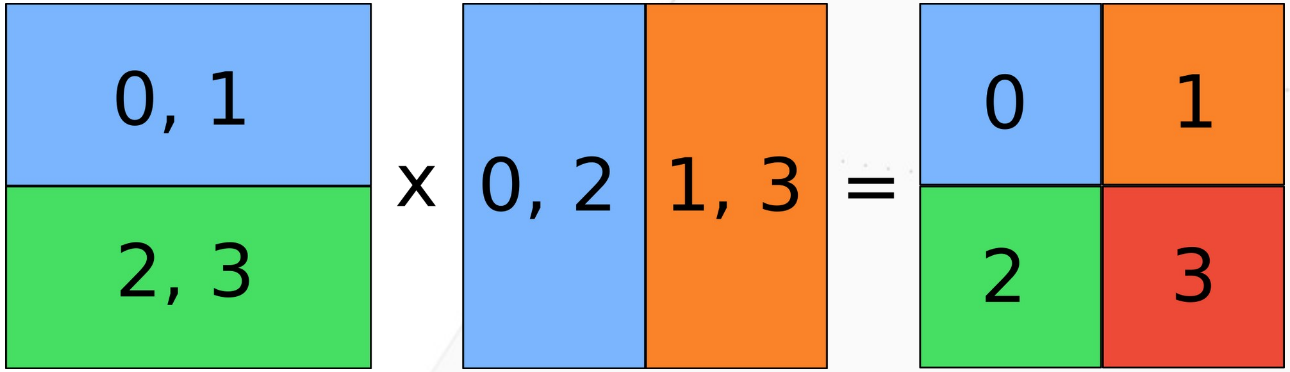
Naive solution

- Memory footprint

$$2.N.\frac{N}{\sqrt{p}} + \left(\frac{N}{\sqrt{p}}\right)^2$$

→ memory scaling problem

- Communication : 0



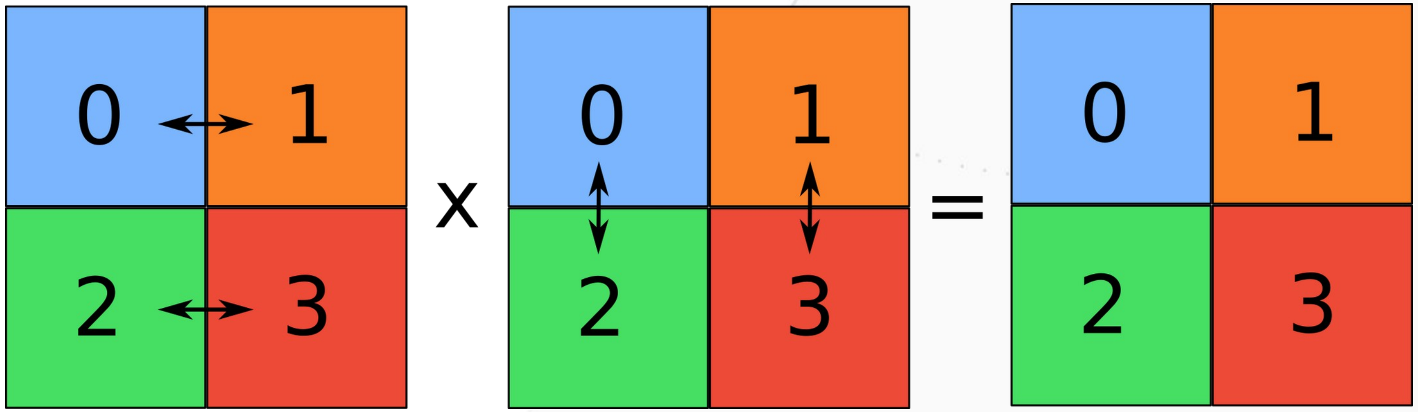
Exercise

Other solutions

- Memory footprint

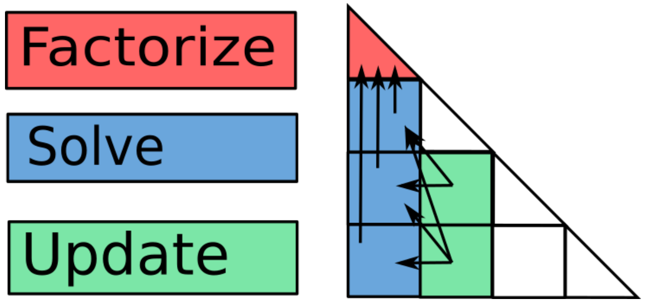
$$3.\left(\frac{N}{\sqrt{p}}\right)^2$$

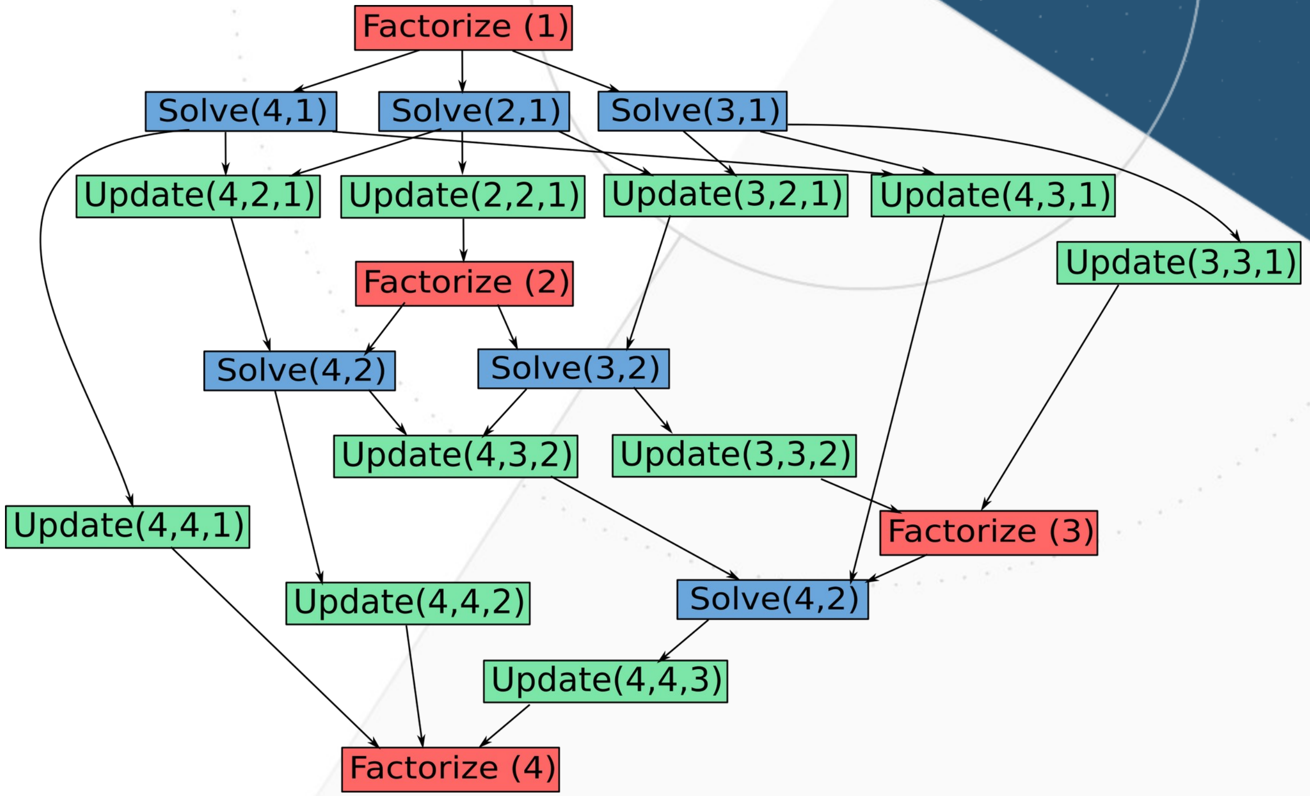
- Communication: $\sqrt{p}$ phases
- Several algorithms exist: Cannon, Fox, Snyder



Task parallelism

- Decompose a program as a Direct Acyclic Graph (DAG) of tasks
  - Nodes = tasks (functions)
  - Edges = data dependencies
- Example: *Choleski factorization*

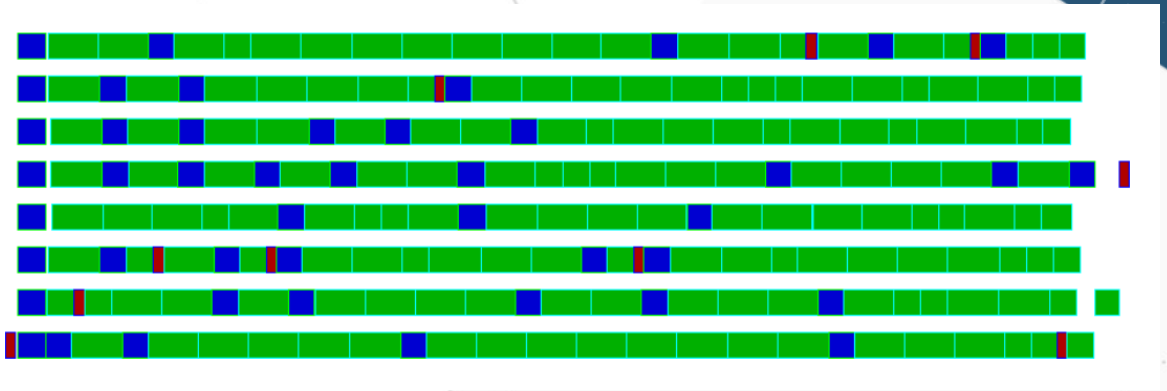




Data parallelism vs Task parallelism

- Choleski parallellized with #pragma omp parallel for

- Choleski parallelized with tasks



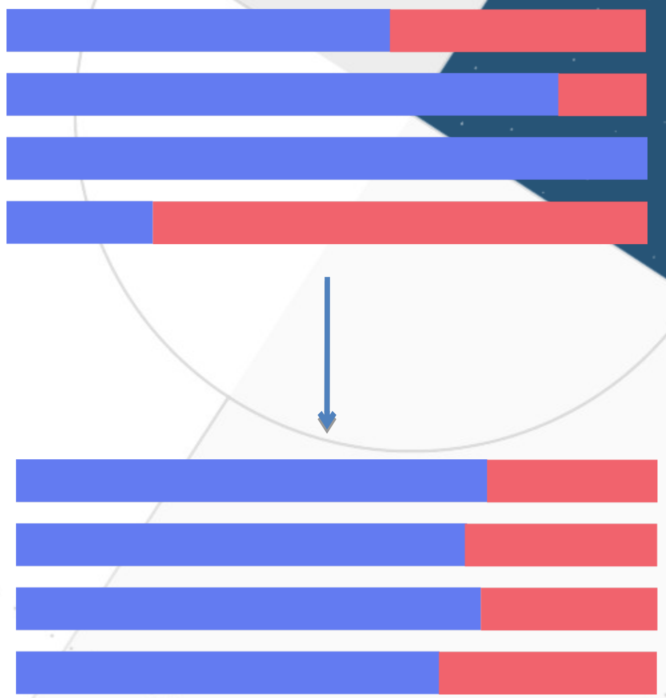
How to balance the workload ?

Load balancing

- Goal of parallelism: reducing the execution time

  ~ each thread has the same execution time

  → Load balancing



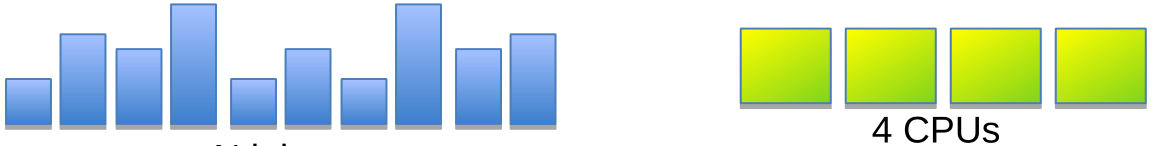
Load balancing

- 3 levels of difficulty:
  - Easy: *n* homogeneous jobs

N jobs

4 CPUs

  - Hard: *n* heterogeneous jobs



N jobs

4 CPUs

  - Harder: the cost of jobs is unknown

Stencils, dense matrices, etc.

Sparse MxV, etc.

Searching, etc.

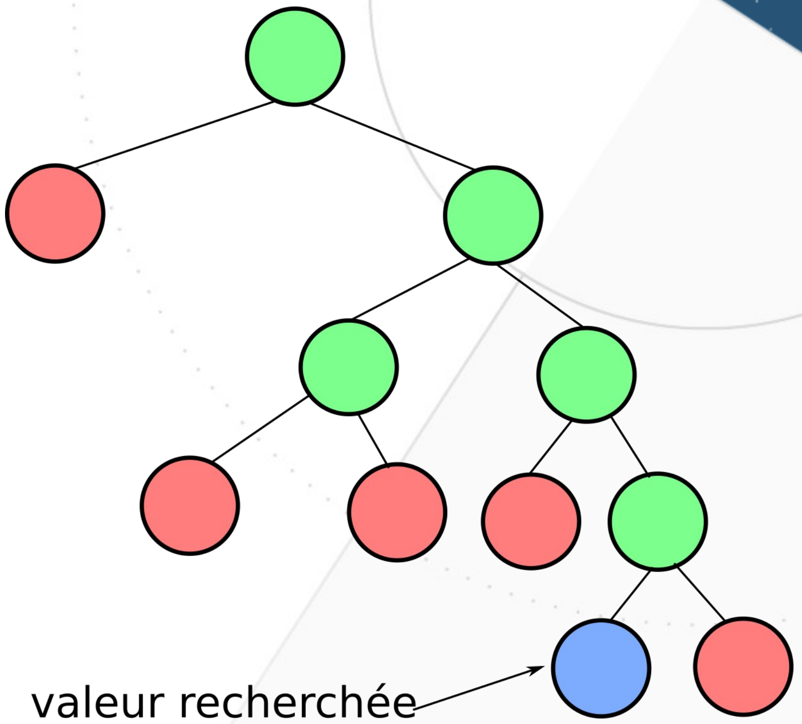
Static scheduling

- Static distribution of the workload
  - Equally split the data and distribute it
  - No communication at runtime
  - Example with OpenMP: `schedule(static)`
- Efficient for homogeneous cases
- Not efficient if
  - CPUs are heterogeneous
  - The workload is irregular

Dynamic data distribution

example: searching in a graph

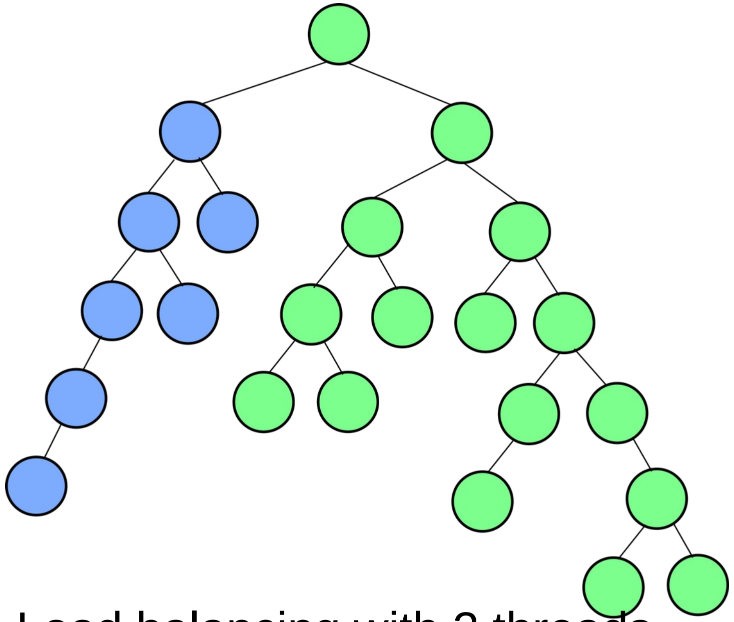
- Searching for a value in a graph/tree



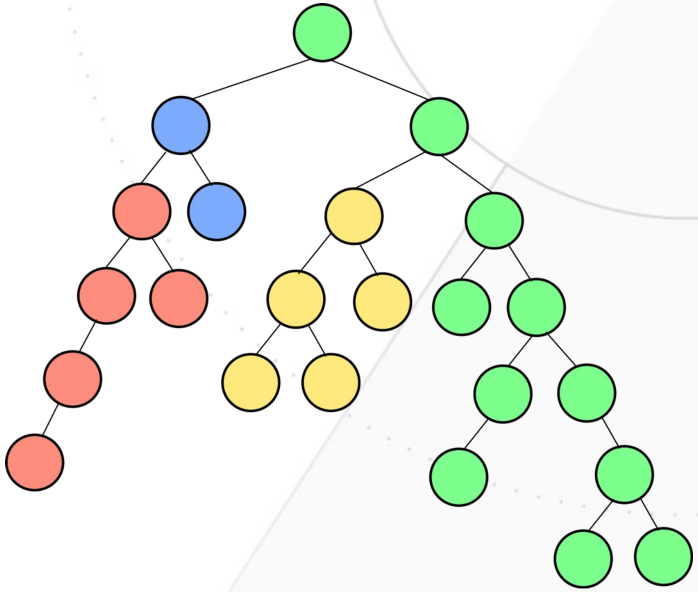
Dynamic data distribution

example: searching in a graph

- Static distribution
  - Each new node is assigned to an idle CPU



Load balancing with 2 threads

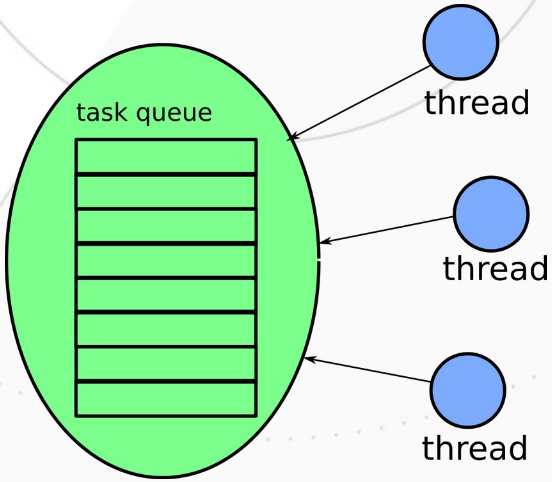


Load balancing with 4 threads

Tasks queues

Master/slave scheme

- A list of task to be executed
  - Managed by a master thread
  - Or in a protected data structure
    - ex: `schedule(dynamic)` d'OpenMP
- Problems
  - Task granularity
    - Many small tasks → contention
    - Few large tasks → load inbalance
  - No data locality

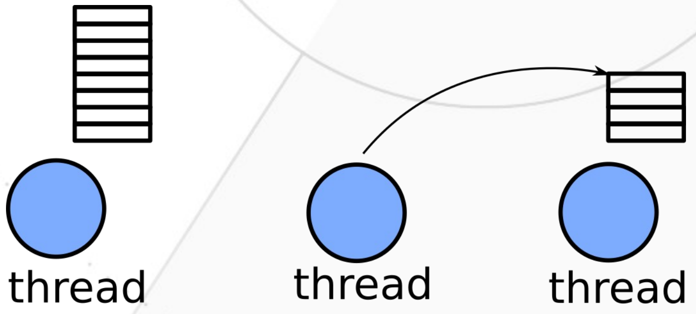


Multiple tasks queues

Work stealing

- One list of tasks per thread
  - Maintain data locality
  - Little contention
  - When a local task queue is empty : work stealing
    - Who’s the victim ?
    - Should I steal a large tasks ?

      → *Deque (Double-ended queue)*



7 dwarfs of HPC

A dwarf is an algorithmic method that captures a pattern of computation and communication.

- Dense Linear Algebra
- Sparse Linear Algebra
- Spectral Methods
- N-Body Methods
- Structured Grids
- Unstructured Grids
- MapReduce

Complete list: *Asanovic, Krste, et al. "The landscape of parallel computing research: A view from berkeley." (2006)*

Exercise: Mandelbrot

- mandelbrot_seq.c computes the Mandelbrot set
  - For each pixel, a computation is required
  - The number of iteration of this computation results in a color
    - White ↔ lots of computation
    - Black ↔ little computation
- Measure the application current speedup
- Modify the application to improve load balancing
  - Dynamically
  - Statically
- Measure the modified application speedup