½ day cluster Tutorial

Matthijs Douze (SED)

Pierre Neyron (CNRS)

Jean-François Scariot (SIC)

Objectives

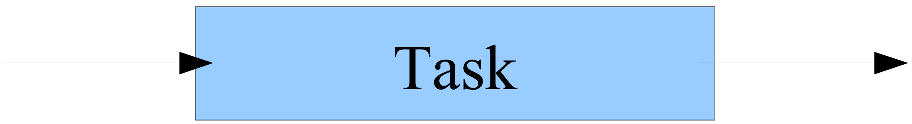
- At noon: ready to run parallel computations
  - Crash-course – simplifications
- Basics of parallelization
  - What you can expect from a cluster
- Accessible platforms
  - For INRIA members
  - Access conditions
- Exercises
  - Simple application cases
  - Main steps of the parallelization

Basics of distributed computing

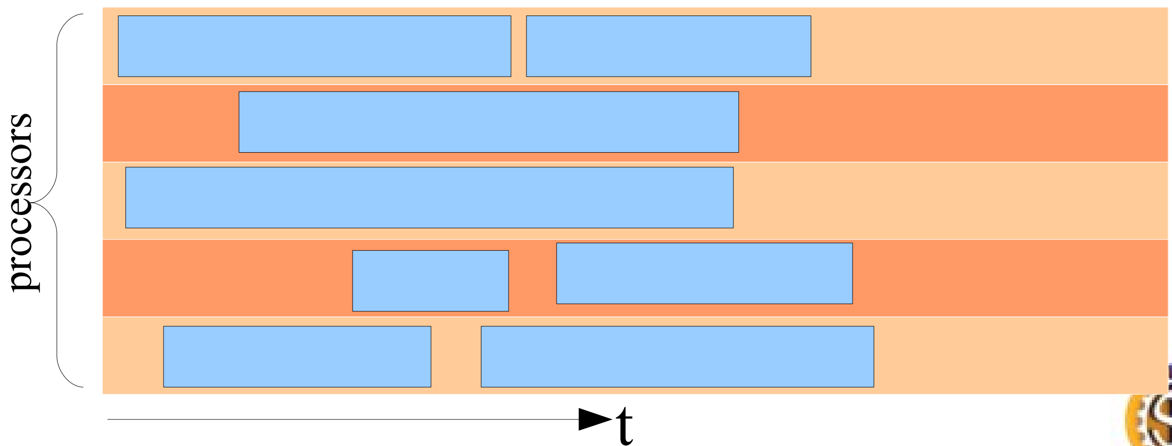
Matthijs Douze

Tasks and processors

- Task = unit of computation
  - Code
  - Inputs
  - Outputs



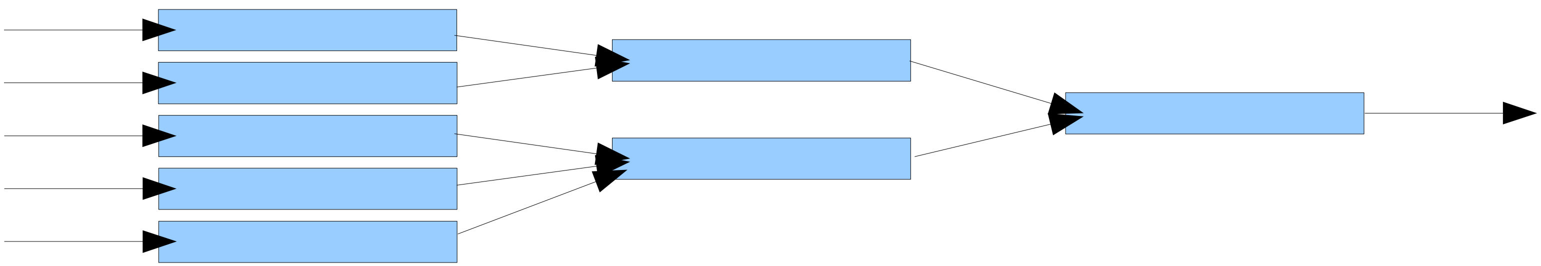
- Processor
  - Executes task code
  - We have a number of them
  - Task on a processor ->processing time
- Scheduling
  - Assignment of tasks to processors
  - Over time
  - Gantt diagram



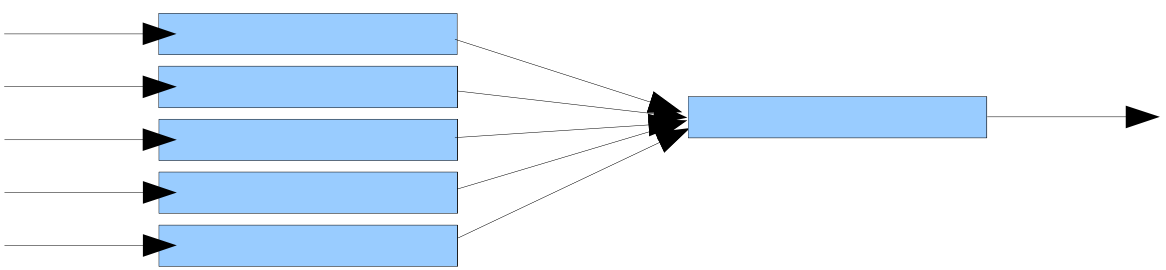
Task dependencies

- Output of one task required as input to another task
  - Dependency graph
  - Determines ordering of tasks
- Sequential
  - Cumsum

- Tree merge



- Parallel + 1 merge operation

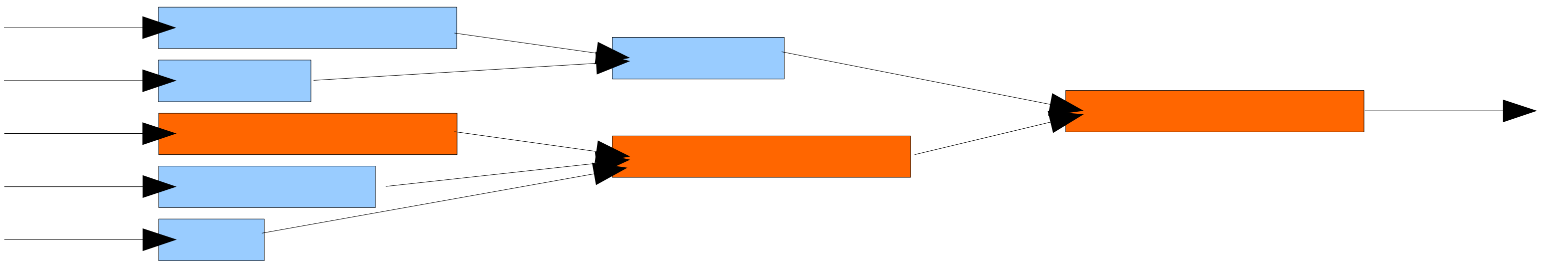


What you can expect

- Lower bounds on total processing time
- Lower bound 1:

$$\frac{\text{sequential processing time}}{\text{nb of processors}}$$

- Lower bound 2:
  - Critical path = longest path in the dependency graph
  - Bound = sum of times for tasks in critical path



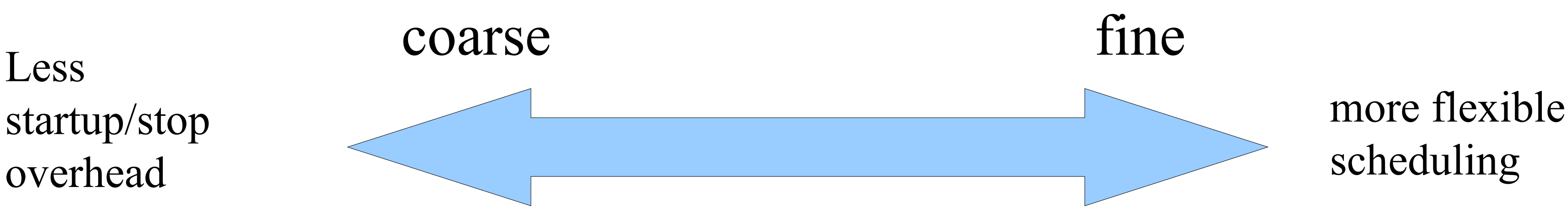
- Lower bounds are not reached in practice
  - Task startup and cleanup overhead
  - Communication overhead
  - Duplicated work between processors
  - Data loaded from a central disk (or other shared resource)

When distribute?

- A cluster is expensive: don't waste resources
- Machine cost per day = 5 euro
  - 3000 euro / 5 years = ~2 euro
  - 250W + 100W electricity = 1.5 euro
  - Sys admin (1 engineer / 100 machines) = 1.4 euro
  - Amazon cloud 10$/day/machine
- Shared resource
  - Social pressure from administrators and other users
  - Should (be able to) justify your usage
- When not distribute?
  - I/O bound tasks (example: grep, small files are worst)
  - Not parallelizable
  - Useless experiments
    - Hard to evaluate...
    - K-means on 10^9 pts for 1000 centroids...
    - Having Tflops available does not mean you should use them

Embarassingly parallel

- Parallel case with lots of small independent tasks, examples:
  - 1s processing on 10000 images
  - Evaluate a grid of 10x10x10 parameters, each evaluation is short
- Easiest to parallelize
- Choice of granularity
  - Tasks can be clustered together -> jobs
  - 100*100 or 10*1000 jobs ?



- We focus on this case
  - Inputs = files and command line params
  - Outputs = files, stdout

Parallelization on one machine

- Not vectorization
  - SIMD: SSE
  - Co-processor GPU: Cuda / OpenCL
- Exploit several cores
- Multithreading
  - OpenMP
- Multiprocessing at shell level
- Demo...
  - echo {1..10} | xargs -n 1 -P 4 ./task.sh
- Orthogonal with cluster
  - Tasks run on cluster can be multi-threaded
  - 1 cluster job =
    - 1 machine (node) or
    - 1 core
- We concentrate on 1-thread tasks

APIs for distributed programming

- MPI (Message Passing Interface)
  - Transfers blocks of data between processes
  - High level of synchronization
  - All started simultaneously
- Map-reduce
  - Map: process input with mapping function, output a dictionary
  - Reduce: data for each dict key is combined by a reduction function
  - Hadoop: focus on robustness to failures
- + tons of others
  - Everybody has his collection of scripts / abstraction layers...
- This tutorial's approach:
  - Start from most basic tools
  - Enough for our scales and types of clusters...
    - 10-100 machines
    - Data central
    - No hardware failures (recover by hand)

Parallelization on machines in your neighborhood

- With a set of machines
  - ssh to them
  - cd to the correct directory
  - Run the task
- Automate this with a tiny script
  - Uses a lock file
  - Demo...

Parallelization on a cluster

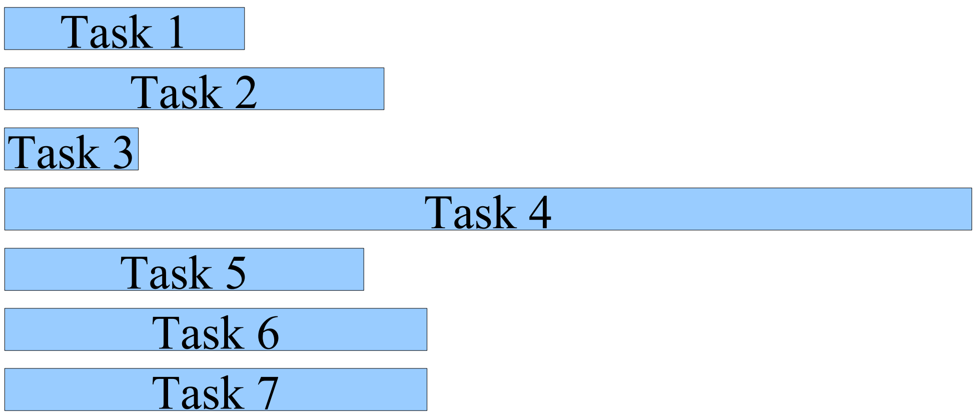
- Cluster = set of computers
  - Similar to desktop machines
  - Uniform: same OS, centralized storage
  - Intel + 64 bit Linux
- Batch scheduler
  - Maintains a database of **jobs**
  - Decides what jobs are running
  - Starts and kills the jobs
  - Knows the state of the processors: alive, dead, suspected....
- demo...

Basics of scheduling

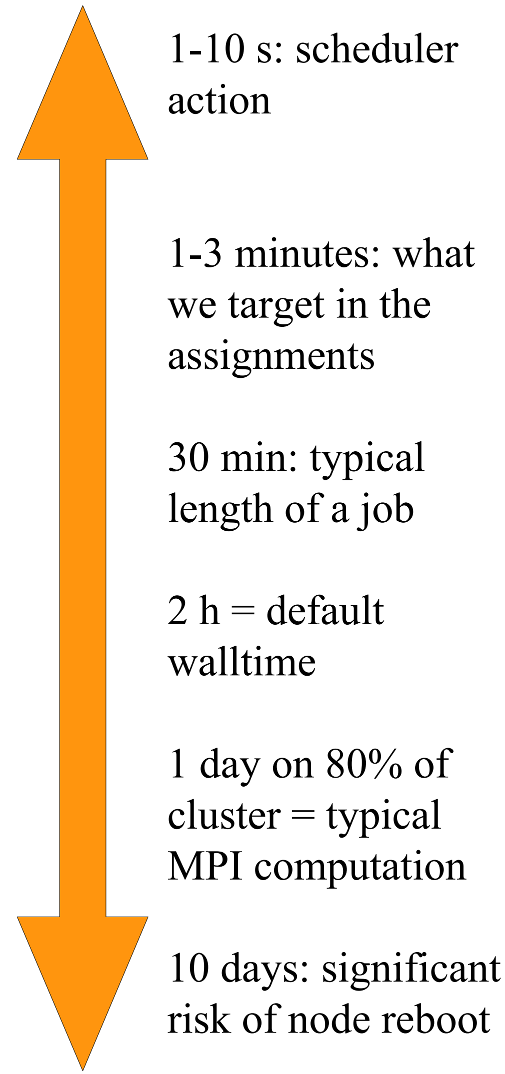
- OAR scheduler (others work similarly)
- Scheduling decisions based on:
  - Default = FIFO
  - Dependencies
  - fair usage
  - Walltime = max time a job is allowed to run
  - Ressources required by job (nb of cores or nb of processors)
- Interactions
  - oarsub: submitting a job = command line
  - oarstat: query state of job
  - oardel: cancels submitted or running job
- Besteffort jobs
  - Submit with oarsub -tbesteffort -tidempotent
  - Killed when normal job submitted, restarted afterwards
  - Flood the cluster without feeling guilty

Length of a job

- Execution time limited by walltime
  - Try to set realistic walltime...
- Run this on 3 processors with FIFO scheduler:



- Checkpointing:
  - Be able to recover from a crash (mem overflow, maintenance, hardware failure, ...)
  - Store state at time intervals or on signal
  - OAR can be instructed to send a signal before kill
- For embarassingly parallel:
  - short and 1-core or 1-processor
  - Do not submit more than 500 processes at a time



Job babysitting

- Always know what your job is doing
  - First few minutes: did my jobs launch?
  - Then every few hours: how are my jobs doing? Did they give partial results?
- On the frontal node
  - Oarstat -f -j <job id>
  - tail -f OAR.*
    - Make sure your program says what it does
- On the node
  - oarsub -C: connect to it
  - top: what's running on the node
  - strace, ls /proc/**pid**/fd: what I/O is a process doing?
  - gdb –pid XXX: connect to running process

Assignments

Assignments

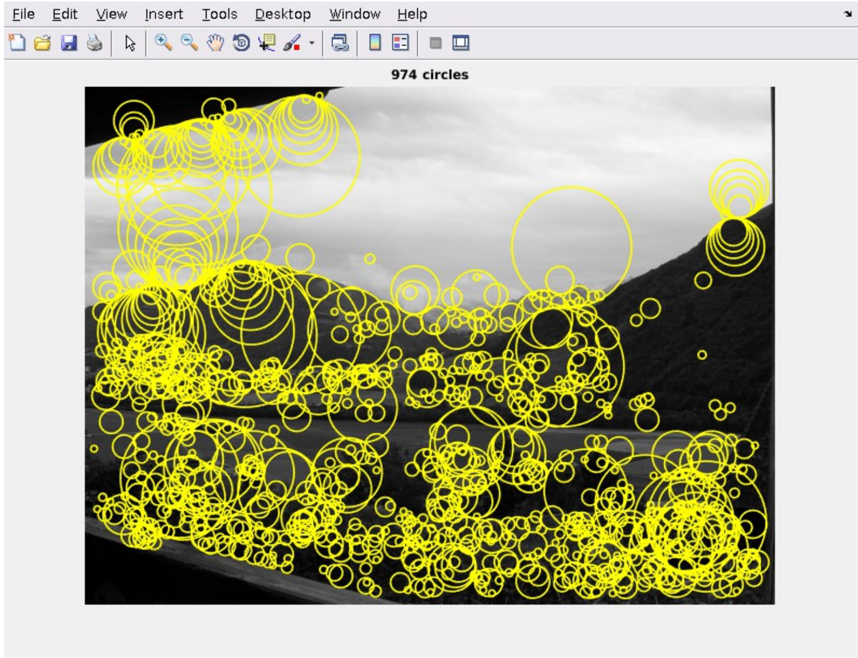
- 3 embarassingly parallel computations
  - I/O via files & parameters
- Get the code & data to the cluster
- Sequential code provided
  - Small run in a few minutes
  - Large run must be distributed
- Evaluate runtime
  - Interactive session: oarsub -I
  - Measure runtime on small case
  - Extrapolate to larger case using complexity
- Split into tasks that last 1 to 3 minutes (should be ~30 min for real application case)
  - Write code for a task that can be launched with oarsub
  - Job's command line argument = what fraction of the work to do
  - Job output = file with partial result
  - Write merging code to get the same output as the sequential code
- Run and monitor the jobs...

C assignment

- Compute the multiplication between 2 square matrices
  - C-storage
  - Triple loop (never do this in reality, use BLAS!)
- Versions:
  - Small: 1000x1000 matrices
  - Large: 5000x5000 matrices
- Split the computation in slices
  - Each task computes slices of lines of the result
- Merging code
  - Stack the slices

- Harder: combine with multithreading
  - #pragma omp parallel for
  - Reserve required # cores

Matlab assignment



- Use a circle detector on a set of images
  - Extremely slow
- Matlab not available on cluster:
  - Would consume too many licenses anyway
- Solutions:
  - Run with octave (what we do here)
  - Compile with the matlab compiler
- Make the script dependent on command-line parameters
  - Matlab: make a function with string parameters
  - Octave: argv()
- Write merging code
- Bonus: mcc
  - Compile, use isdeployed
  - Copy matlab runtime to cluster

Python assignment

- Program that
  - Process a set of text files extracted from PDF
  - Construct the document-word matrix (sparse matrix)
  - Three passes:
    - Collect all words (pass 1)
    - Select words to make a dictionary (remove too frequent and infrequent words)
    - Build matrix (pass 2)
- Cases
  - Small: 2700 files
  - Large: 17000 files
- Parallelize only matrix build
  - Just reuse the dictionary from the small case (pass 1)

- Bonus: how can we avoid small file I/O
  - Unzip on-the-fly to temp directory

Conclusion

- Cluster = little more than many machines piled up
- Basic usage:
  - Easy
  - Mostly standard tools + batch scheduler
- Advanced usage:
  - You may never need it... (I did not)

The central tool : ssh

- All communication goes via ssh
- Ssh tunnels through bastion
  - Tunnel to connect directly to a machine via another
    - ssh -o ProxyCommand="ssh douze@bastion.inrialpes.fr -W access1-cp:22 " douze@localhost -o StrictHostKeyChecking=no
  - scp: copy data
    - scp -o ProxyCommand="ssh douze@bastion.inrialpes.fr -W access1-cp:22 " .bashrc douze@localhost:/tmp
  - sshfs: mount directory (linux and mac)
    - sshfs -o ProxyCommand="ssh douze@bastion.inrialpes.fr -W access1-cp:22 " douze@localhost:/services/scratch/lear/douze /mnt/cluster_scratch
- OAR's ssh wrappers
  - Some black magic to isolate the jobs on a node
  - oarsub -C (frontal -> node)
  - oarsh (node -> node)
    - When reserving several nodes for 1 job