Multicore workshop Day 1

August 2016

Francis Giraldeau
francis.giraldeau@polymtl.ca

Prof. Michel Dagenais
michel.dagenais@polymtl.ca

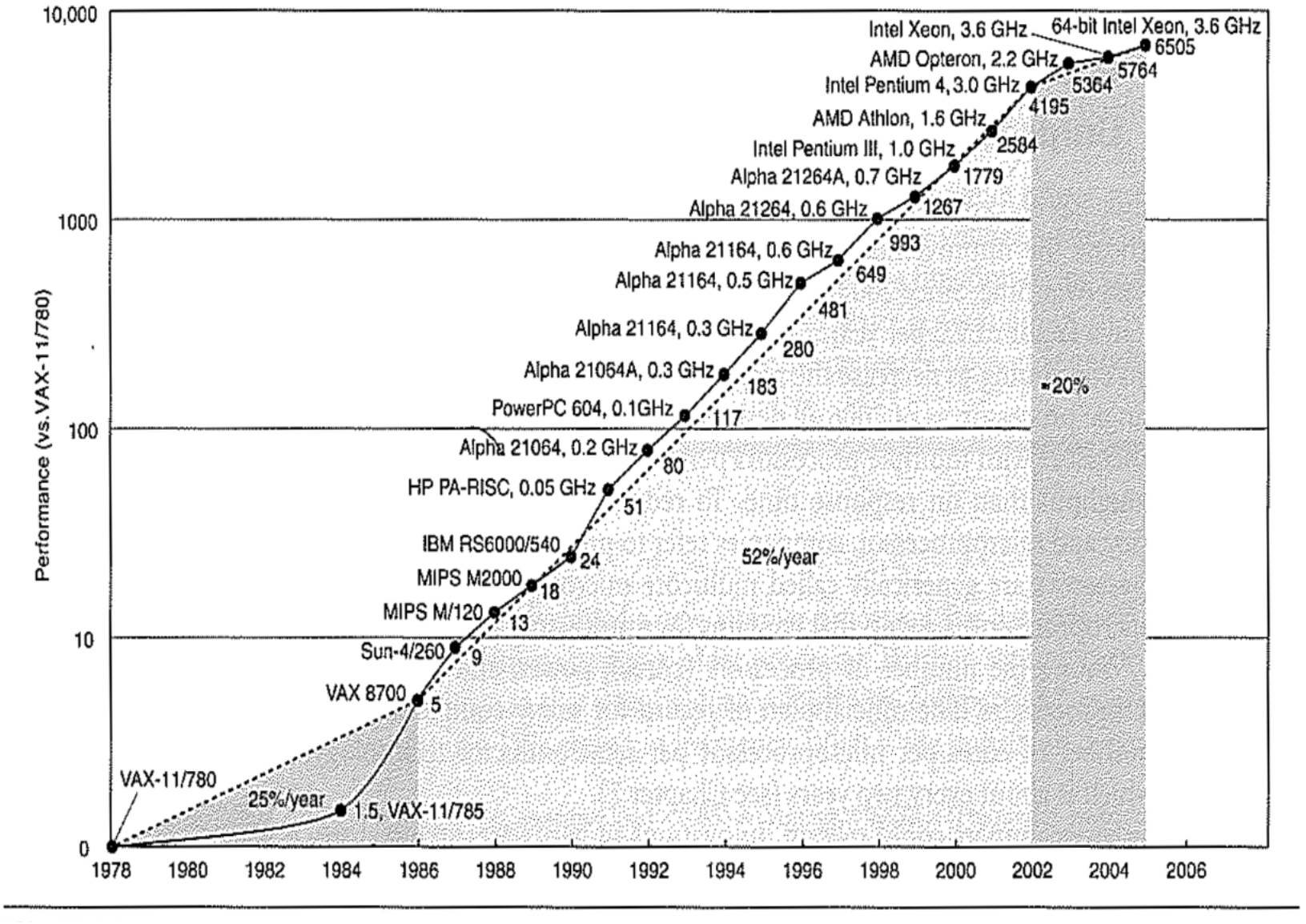
Polytechnique Montréal

Master plan

- Make program run faster!
- Design of parallel program
- Scalability analysis
- Parallel algorithms, libraries and tools
- Single Instruction Multiple Data (SIMD)
- Hardware counters
- Lock-less data structures
- Advanced cache optimizations
- Debugging race conditions

Introduction

Processor performance evolution



Source: Computer organization and design, 4th edition

POLYTECHNIQUE MONTRÉAL

ERICSSON

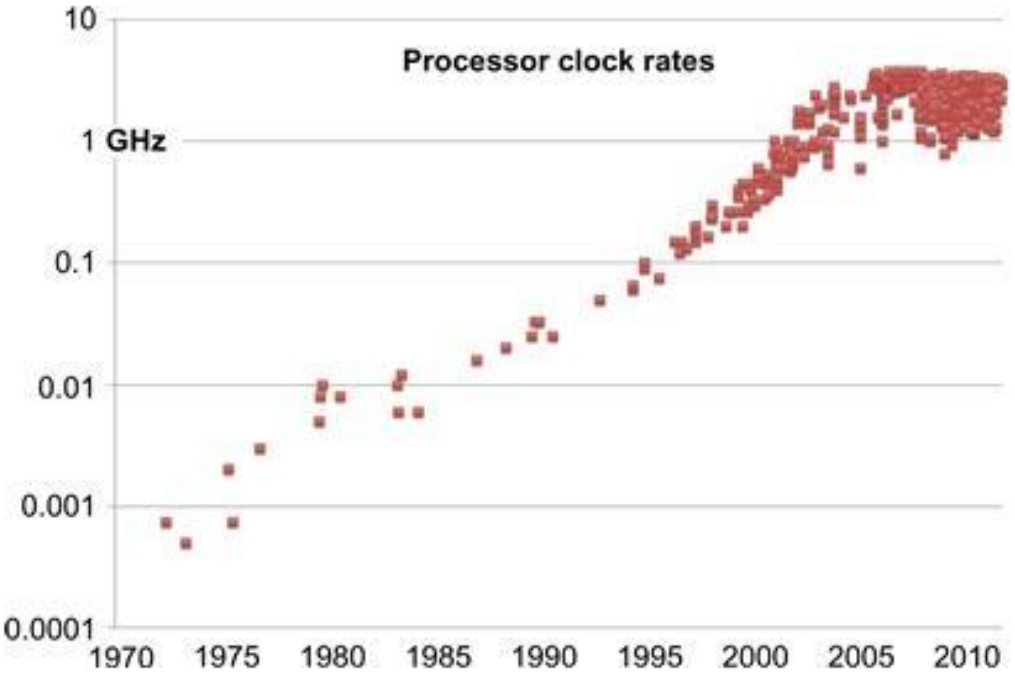


Figure 1.4 Growth of processor clock rates over time (log scale). This graph shows a dramatic halt by 2005 due to the power wall, although current processors are available over a diverse range of clock frequencies.

Source: Structured Parallel Programming, Patterns for Efficient Computation

POLYTECHNIQUE MONTRÉAL

ERICSSON

Instruction pipeline

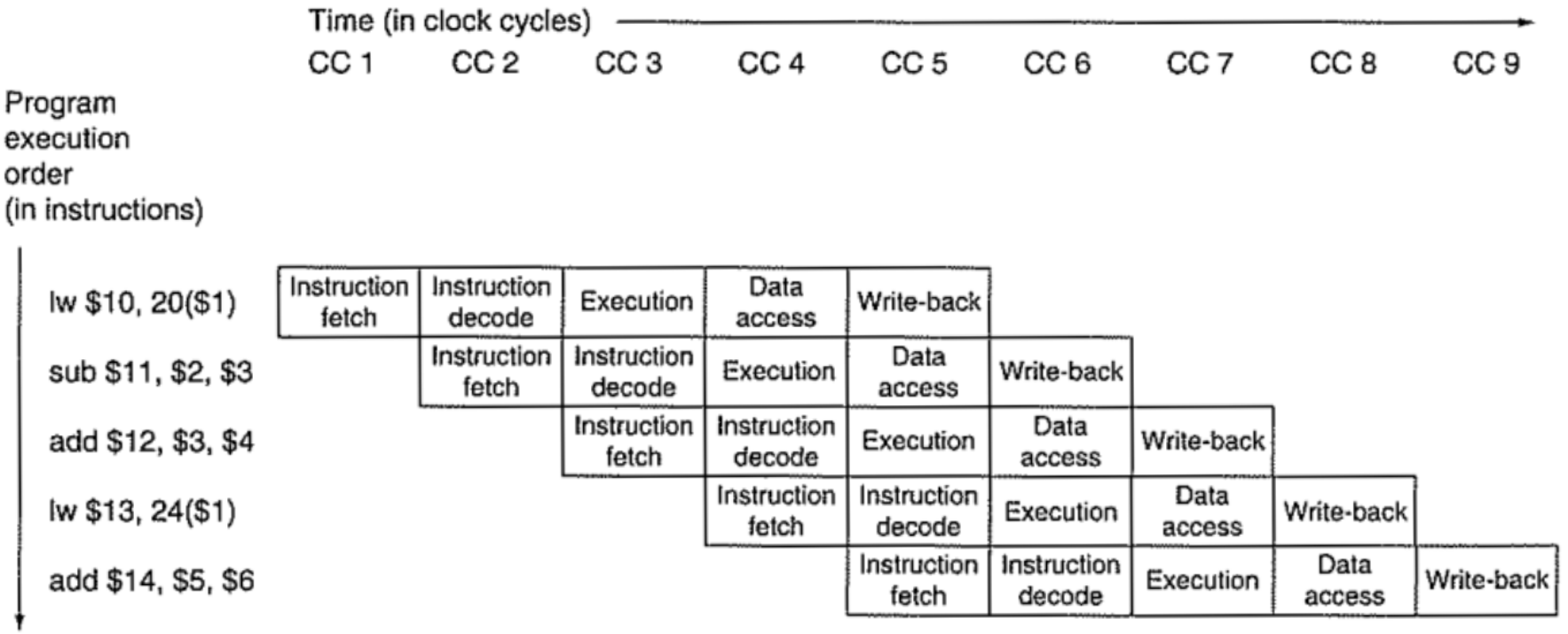
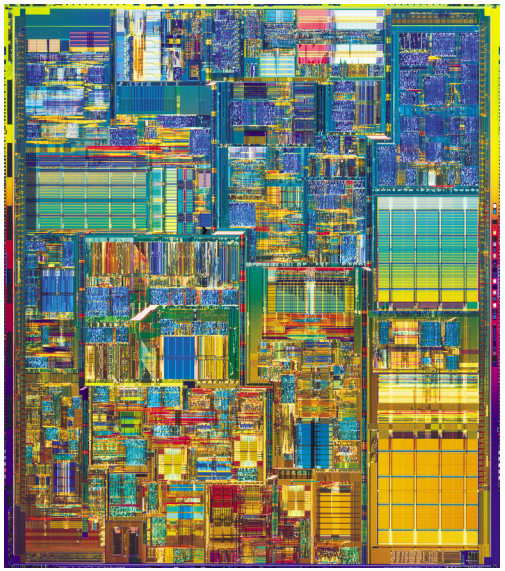


FIGURE 4.44 Traditional multiple-clock-cycle pipeline diagram of five instructions in Figure 4.43.

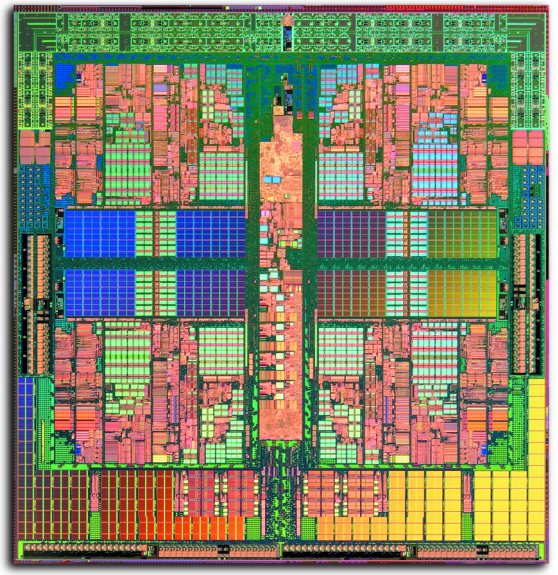
Source: Computer organization and design, 4th edition

POLYTECHNIQUE MONTRÉAL

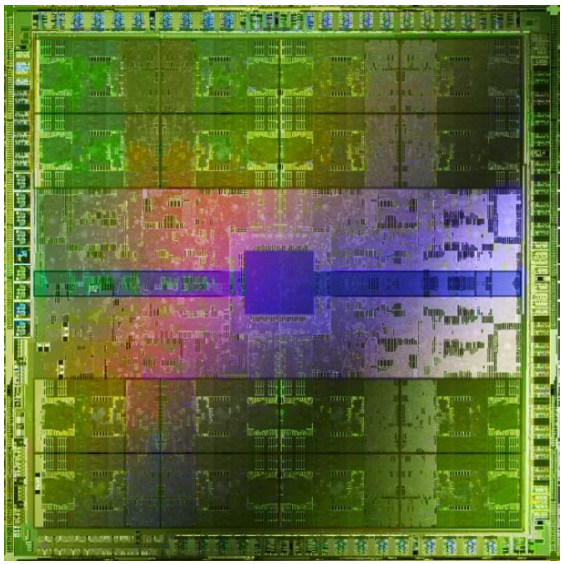
ERICSSON



Uniprocessor



Multicore



Manycore

Source des images: wikimedia.org

Processor organization

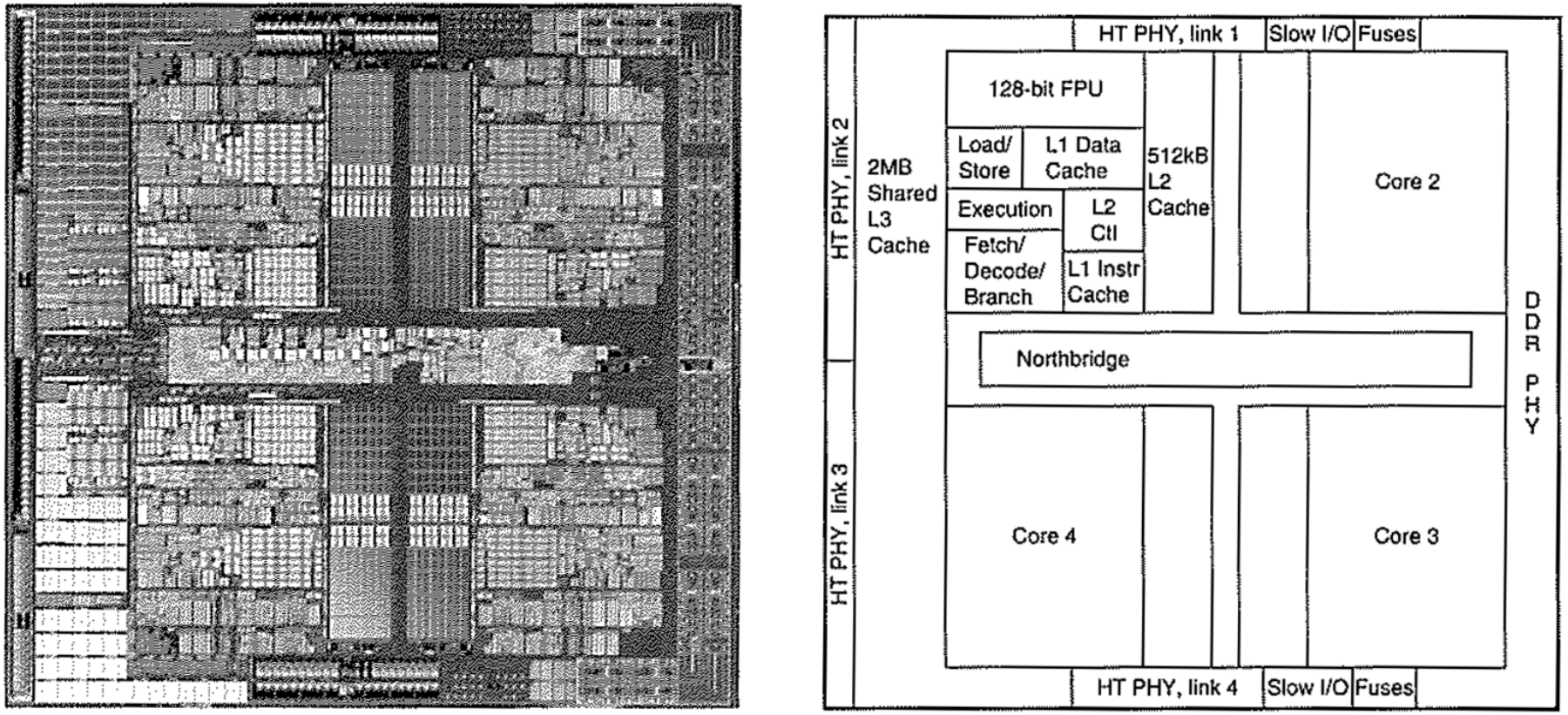
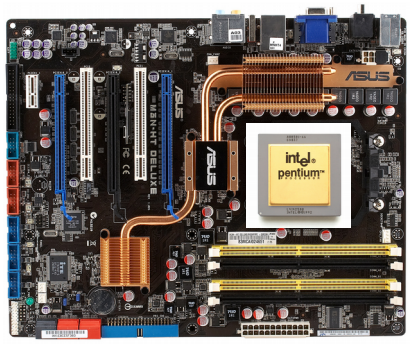


FIGURE 1.9 Inside the AMD Barcelona microprocessor. The left-hand side is a microphotograph of the AMD Barcelona processor chip, and the right-hand side shows the major blocks in the processor. This chip has four processors or "cores". The microprocessor in the laptop in Figure 1.7 has two cores per chip, called an Intel Core 2 Duo.

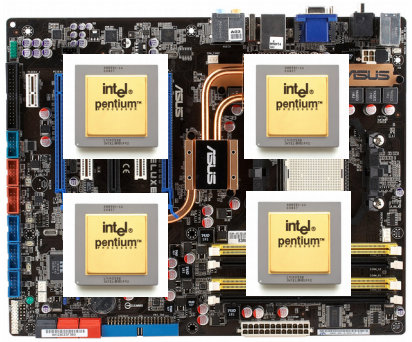
Source: Computer organization and design, 4th edition

POLYTECHNIQUE MONTRÉAL

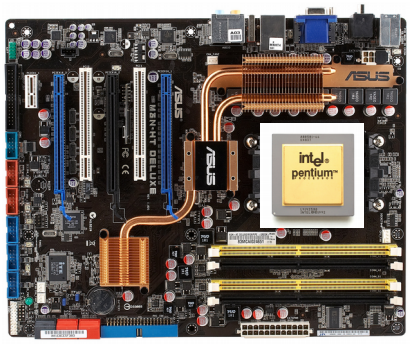
ERICSSON



- Uniprocessor

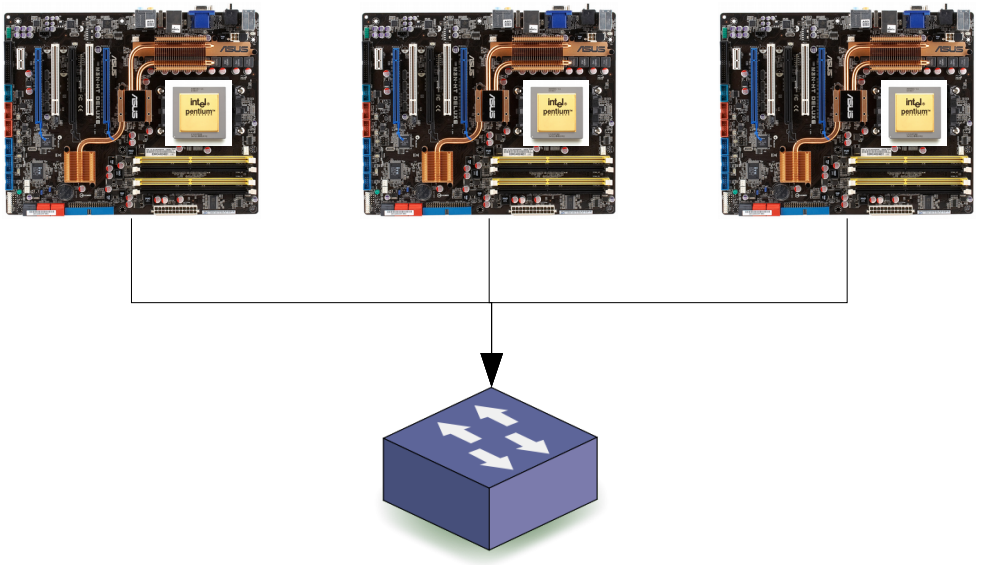


- Multiprocessor
- Shared memory
- Common clock (synchrone)
- NUMA



+

- Host and coprocessor
- Heterogeneous computing
- Memory is partitionned



- Cluster of computers
- Communication using network

Mainframe

Shared memory
Easy programmation
Good scalability
Cost: $$$$$

Beowulf cluster

Distributed memory
Program require modification
Scalacility limited by the network
Cost: $

Source: images.google.com

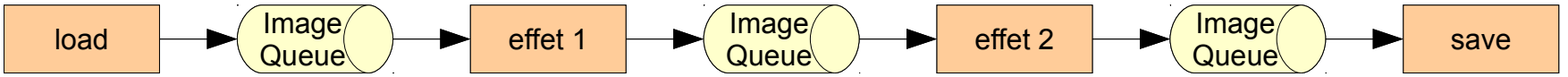
Types of parallel systems

Parallel tasks

- Decompose processing in stages
- Example : pipeline, batch processing
- Goal is to increase throughput and mask disk latency
- Does not reduce latency of one task
- Linear scalability for independent tasks

Example of parallel task

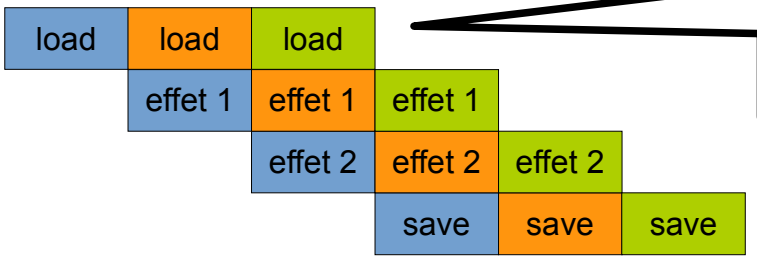
Image processing pipeline



Each image takes 4 units of time

Traitement en série

Traitement en pipeline



Keep disk busy

Total duration reduced, but each image still takes 4 units of time

POLYTECHNIQUE MONTRÉAL

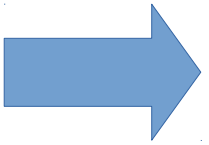
ERICSSON

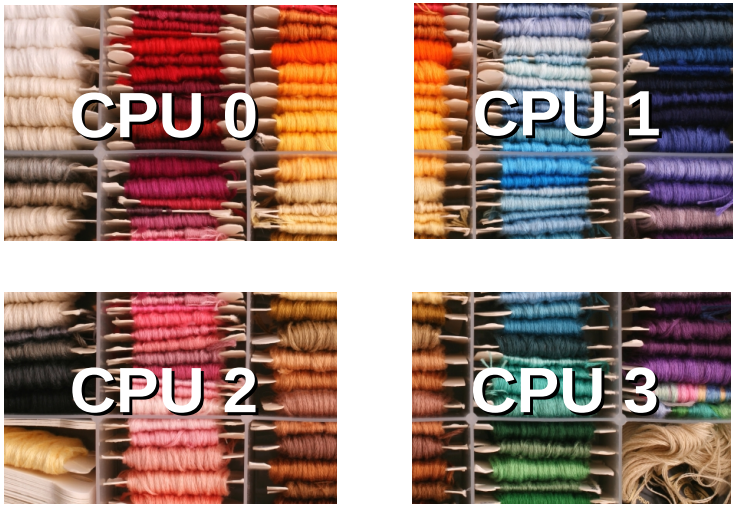
Data parallel algorithm

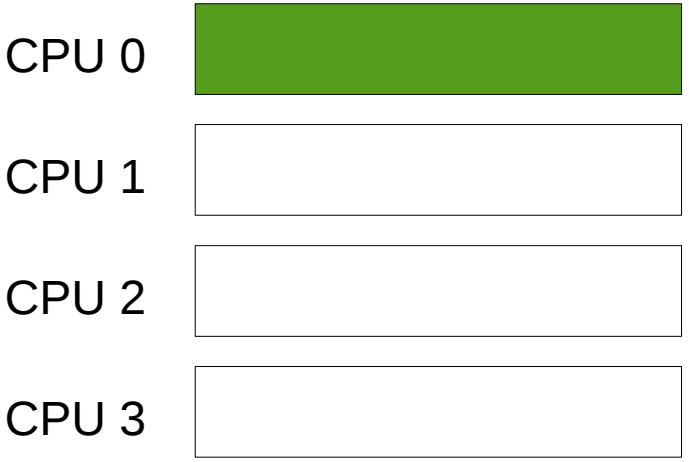
- Split the data in chunks
- Simultaneous execution of the same function on different data sections
- Goal : reduce latency and/or power

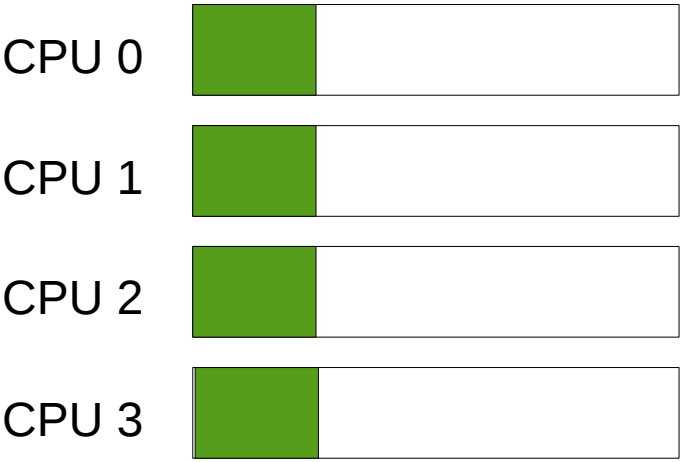
Example of data parallelism

Apply a filter on an image









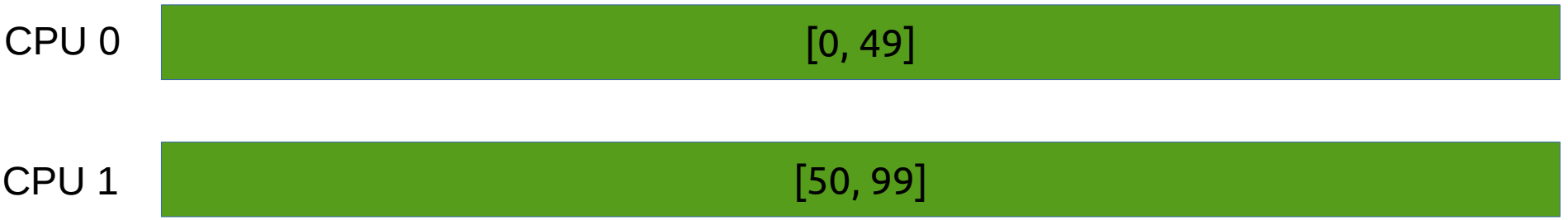
POLYTECHNIQUE MONTRÉAL

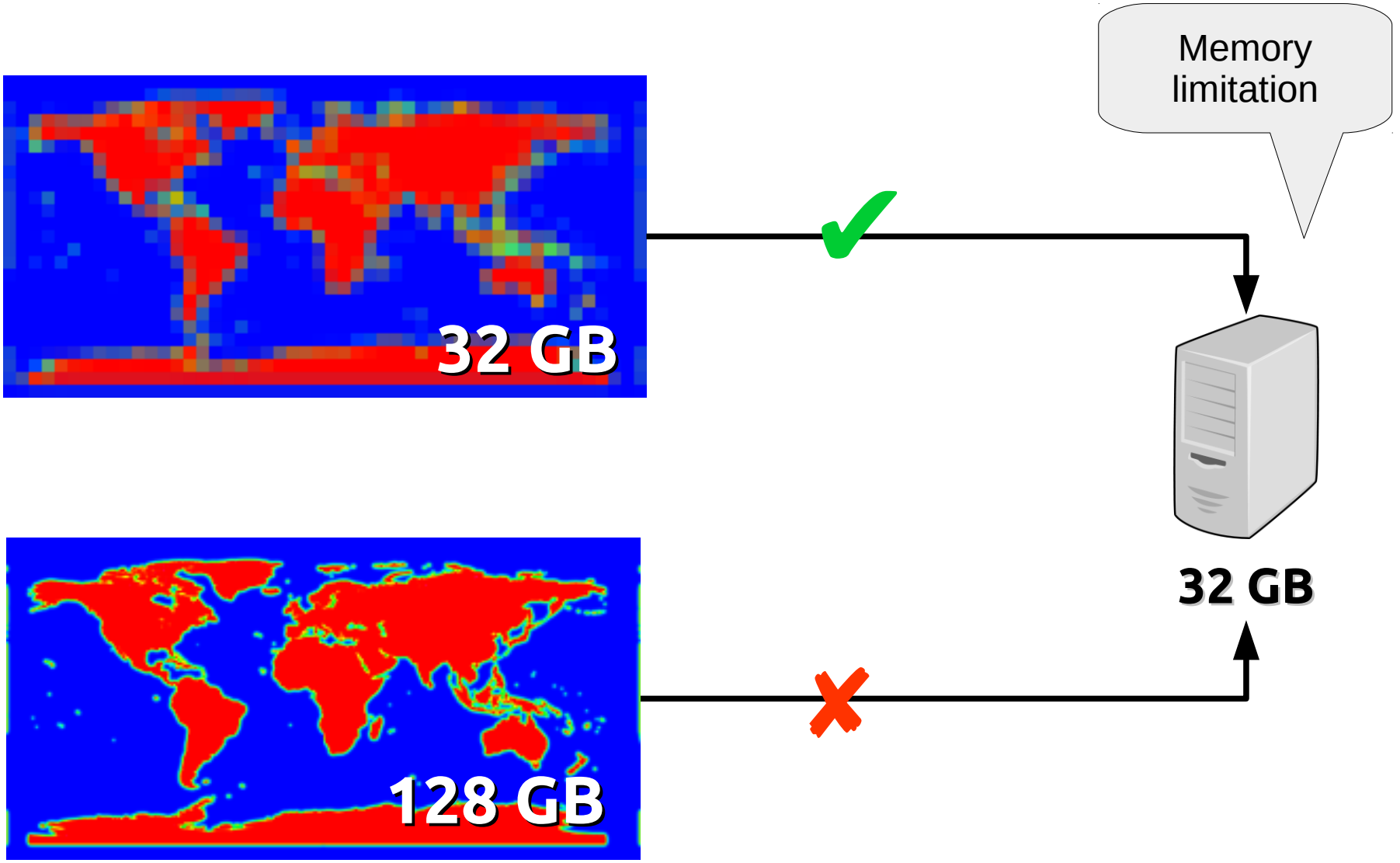
ERICSSON

Serial computation (1 GHz):

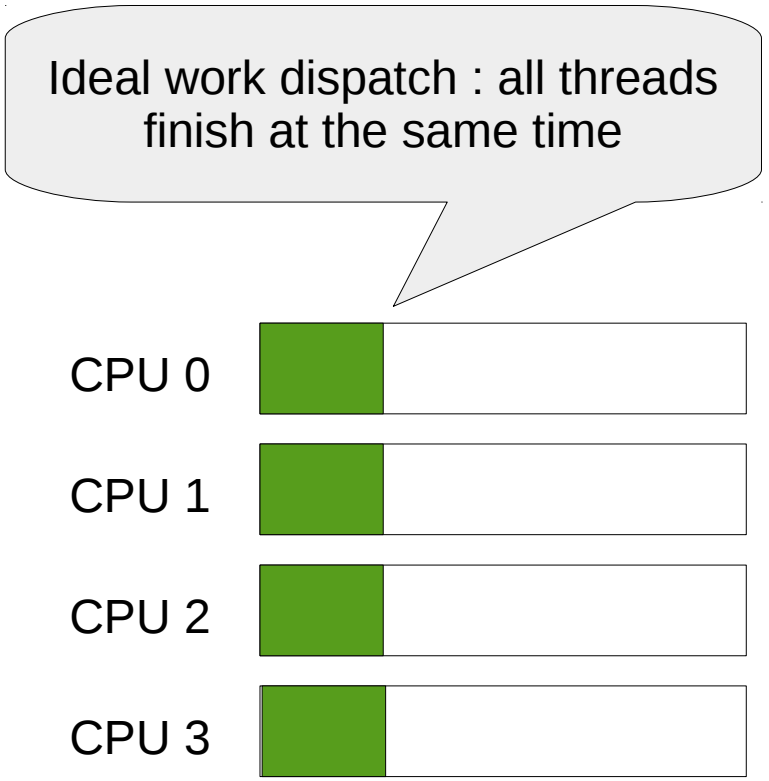
Parallel processing (1 GHz):

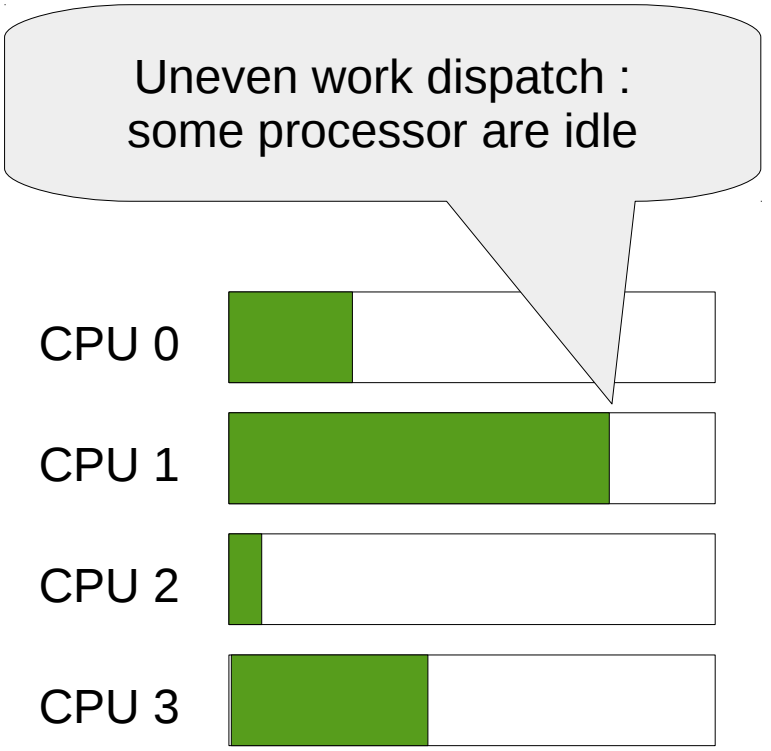
Parallel processing (500 MHz):




Memory limitation
32 GB
32 GB
128 GB

Dispatching the work





Activity: parallel image processing

- 01-pthread-image
  - ./01-pthread-image --input <image>
  - Each thread colorize a portion of the image
  - Your goal: dispatch the work between threads
  - Trace the program and check the end time jitter
  - What is the overhead of spawning the threads?
  - Test with small image res/bsod.png
  - Measure the speedup with image res/world.jpg

Parallel program process

- Implement serial algorithm and input data
- Measure the serial elapsed time
- Profile the program to identify the hotspots
- Implement the parallel algorithm
- Measure the speedup
- Compare theoretical and actual speedup
- Improve until objective is attained

Metrics for parallel programs

- Predicted, theoretical speedup
  - Amdahl's law
  -
- Effective speedup
- Efficiency

Measure effective speedup

$$Speedup = \frac{t_{serial}}{t_{\text{parallel}}}$$

Example: The serial execution time of a program is 5 seconds. The parallel version of the same program takes 1.25 secondes. The speedup is therefore:

$$\frac{5}{1.25} = 4$$

Max theoretical speedup

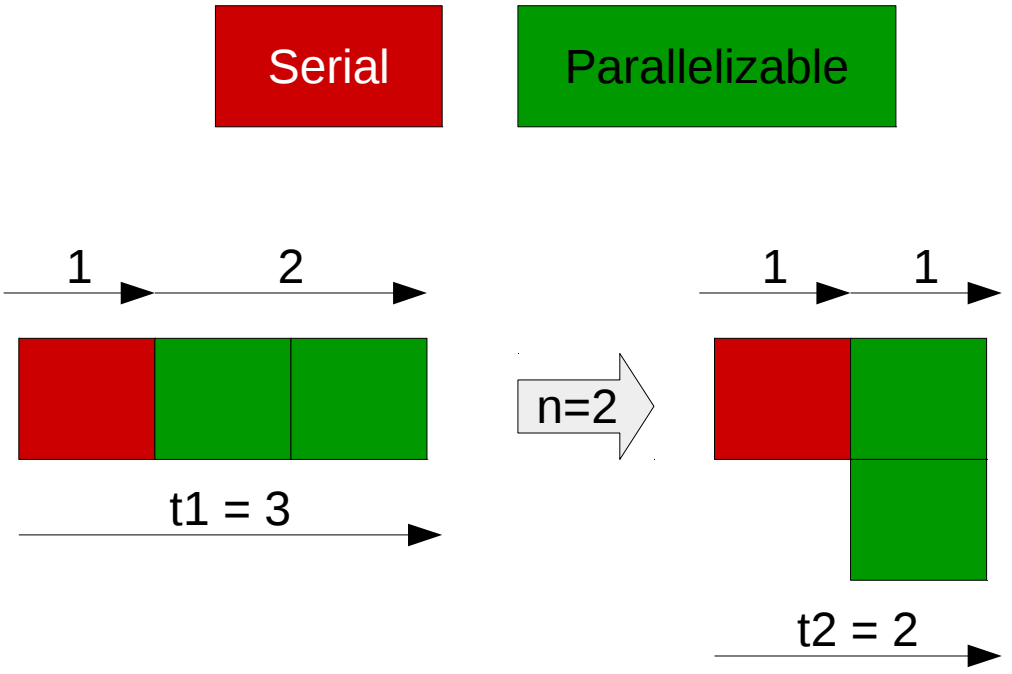
- Depends on the serial portion of the program
- Depends on the number of processors
- Amdah's law

$$S = \frac{t_1}{t_n} = \frac{t_1}{t_1 \cdot \left(s + \frac{(1-s)}{n} \right)}$$

$$t_n = t_s + \frac{t_p}{n}$$

$$= (t_1 \cdot s) + \frac{(1-s) \cdot t_1}{n}$$

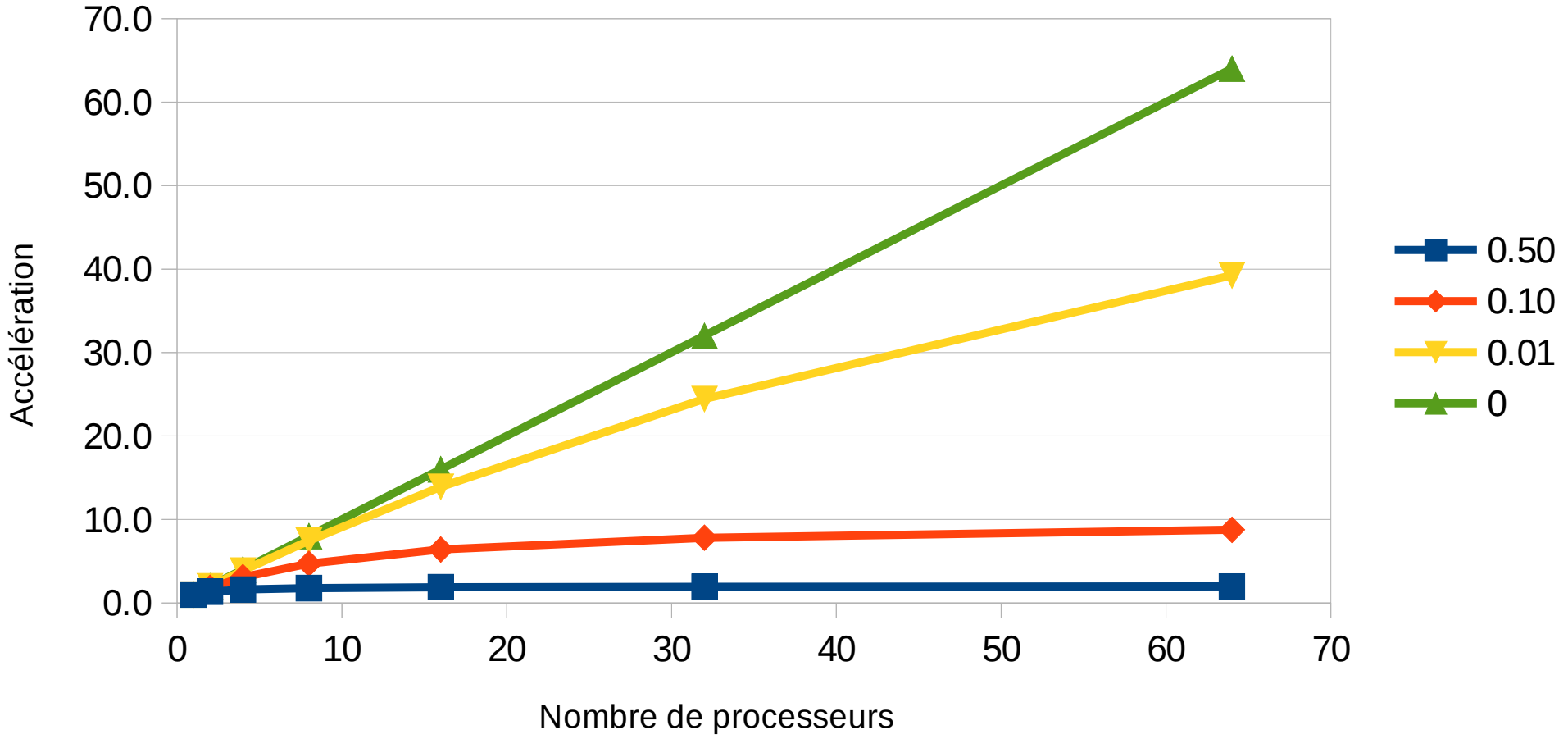
$$= t_1 \cdot \left(s + \frac{(1-s)}{n} \right)$$



POLYTECHNIQUE MONTRÉAL

ERICSSON

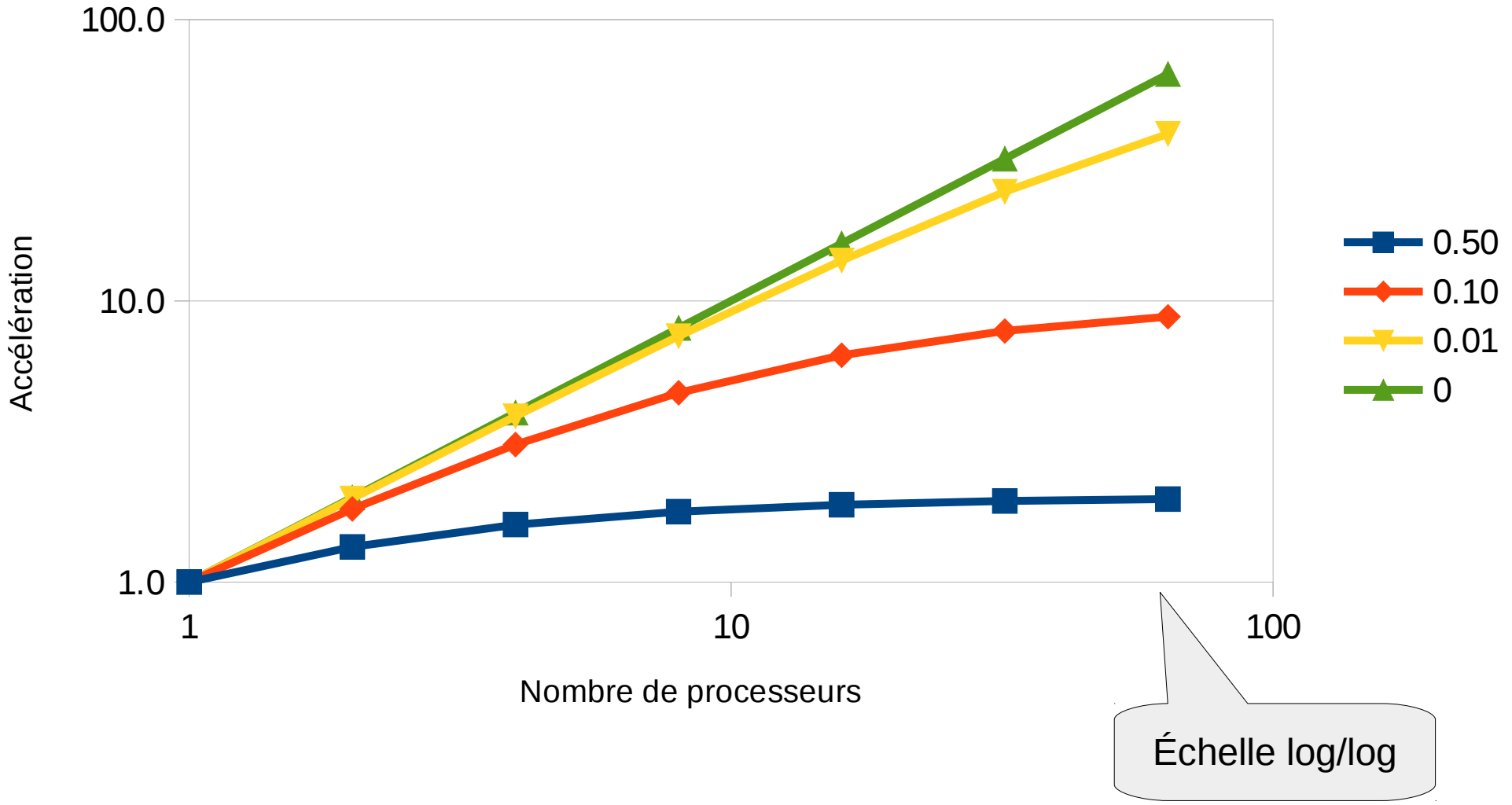
Accélération selon le nombre de processeur et la proportion sérielle



POLYTECHNIQUE MONTRÉAL

ERICSSON

Accélération selon le nombre de processeur et la proportion sérielle



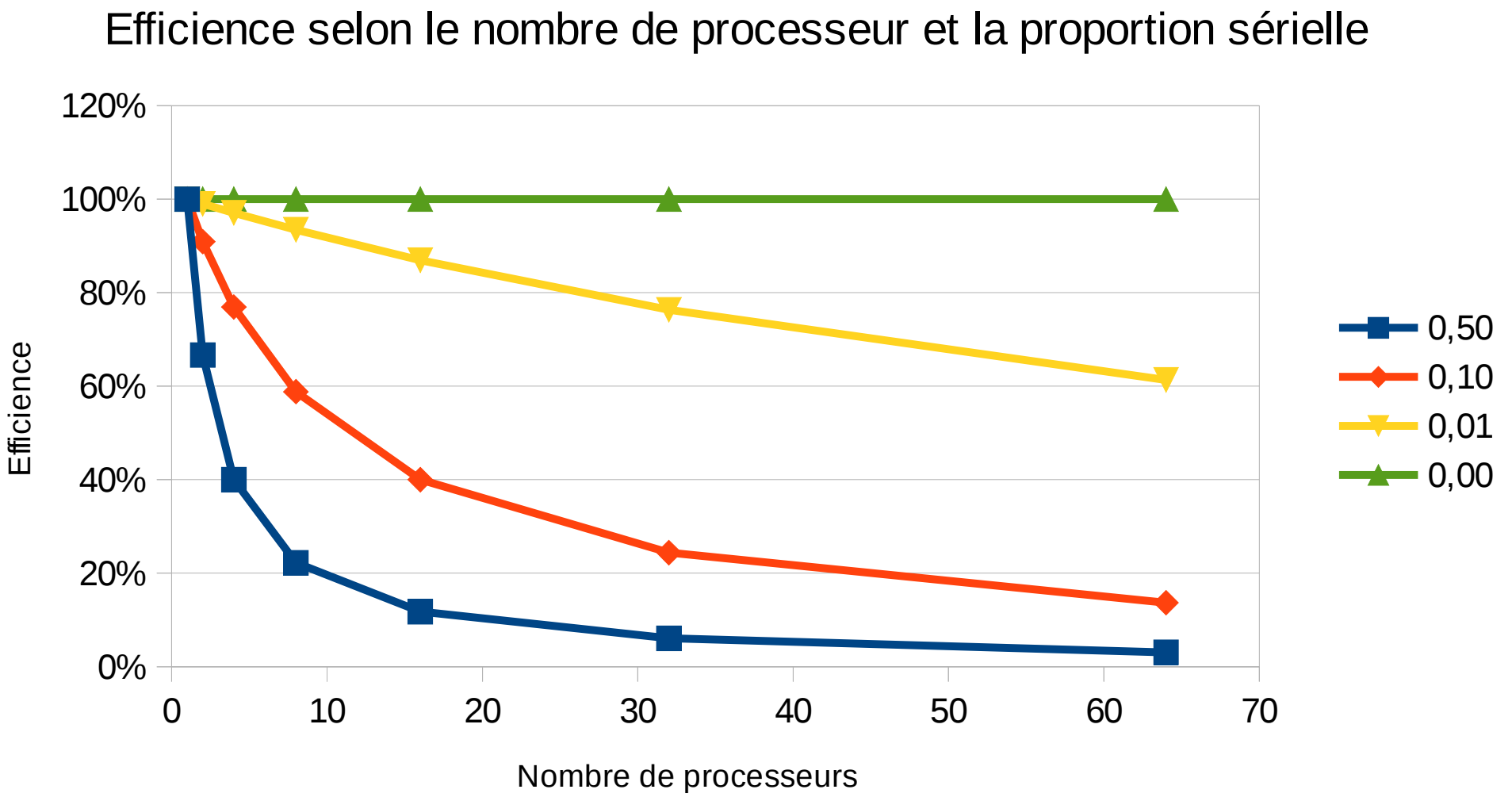
POLYTECHNIQUE MONTRÉAL

ERICSSON

Efficiency

- The speedup increase is not proportional to number of
- Tradeoff between acceleration and resources utilization

$$Efficiency = \frac{Speedup}{N}$$



POLYTECHNIQUE MONTRÉAL

ERICSSON

Exercice

- A program process video frames for an autonomous vehicle. Each frame takes 250ms (4 fps) which is not fast enough to react to emergency braking. Profiling shows that 80% of the image processing can be made parallel.
- Compute the maximum speedup, the efficiency, the time to process a frame and the achieved frequency with a 64 cores processor.

$$T = 250 \cdot 0.2 + \frac{250 \cdot 0.8}{64} = 50\,ms + 3\,ms = 53\,ms$$

$$f = \frac{1}{0.053} = 19\,fps$$

$$S = \frac{t_1}{t_1 \cdot \left(s + \frac{(1-s)}{n} \right)} = \frac{250}{250 \cdot 0.2 + \frac{250 \cdot 0.8}{64}} = 4.7$$

$$E = \frac{4.7}{64} = 7\,percent$$

Import multicore project

- git clone git://git.dorsal.polymtl.ca/~fgiraldeau/multicore.git
- Shadow build in the build directory
- cd multicore/build
- Debug: qmake -qt=qt5 -r “CONFIG += debug” ../
- Release: qmake -qt=qt5 -r “CONFIG += release” ../
- Open Eclipse CDT Neon
- Import project, select multicore directory
- Support for C++11: the settings is not properly saved/restored, you need to set it yourself after import:
- Project -> Properties -> C/C++ General -> Providers -> CDT GCC Built-in Compiler Settings -> Command to get compiler specs -> append -std=c++11
- clean, build, rebuild index and if the stars are aligned, it should work™

Connect to octosquare

- Octosquare is a 64 cores machines for scalability experiments
- Access is done through test, but add this config to .ssh/config file to connect directly:

```
Host octosquare
    User ericsson
    ProxyCommand ssh ericsson@test.dorsal.polymtl.ca -W %h:%p
```

- ssh-copy-id ericsson@test.dorsal.polymtl.ca
- ssh-copy-id octosquare
- Ask for the password
- Tadam! You can directly login with “ssh octosquare”, and scp works too
- Please create a sub-directory inside home to hold your files

Activity

- Profile the program dragonizer using Valgrind
  - cmd: use “draw” to draw the dragon fractal
  - power: the number of fractal iterations as a power of two, use for example 25 (2^25)
  - lib: use “serial”, that’s what we want to parallelize
  - The output is saved in dragon.ppm by default
  - valgrind --tool=callgrind ./dragonizer --cmd draw --lib serial --power 25
- Open the profile using kcachegrind
- What functions would you target for parallelization?