The Abdus Salam
International Centre
for Theoretical Physics

Overview on Parallel Programming Paradigms

Ivan Girotto – igirotto@ictp.it

Information & Communication Technology Section (ICTS)

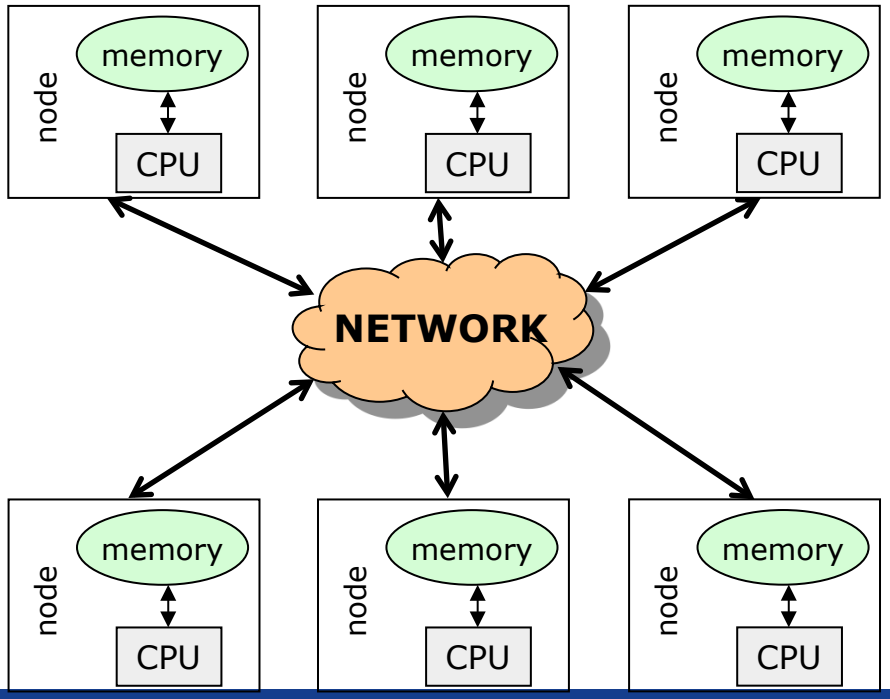
International Centre for Theoretical Physics (ICTP)

What Determines Performance?

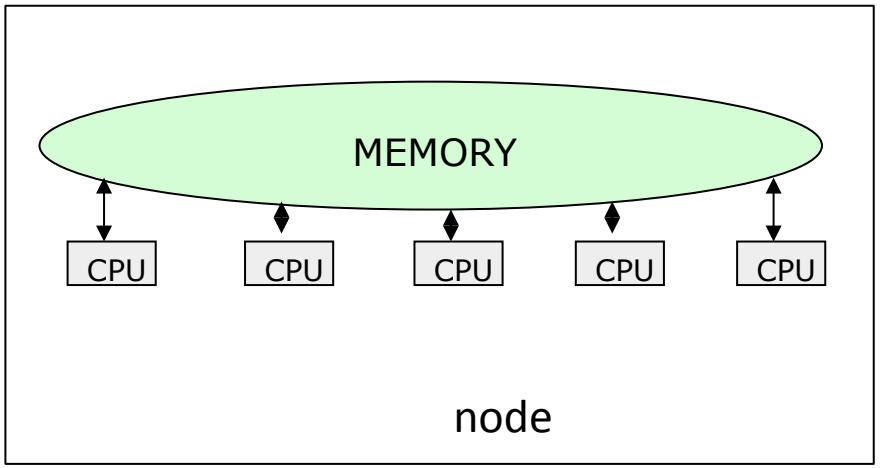
- How fast is my CPU?
- How fast can I move data around?
- How well can I split work into pieces?
  - Very application specific: never assume that a good solution for one problem is as good a solution for another
  - always run benchmarks to understand requirements of your applications and properties of your hardware
  - respect Amdahl's law

Parallel Architectures

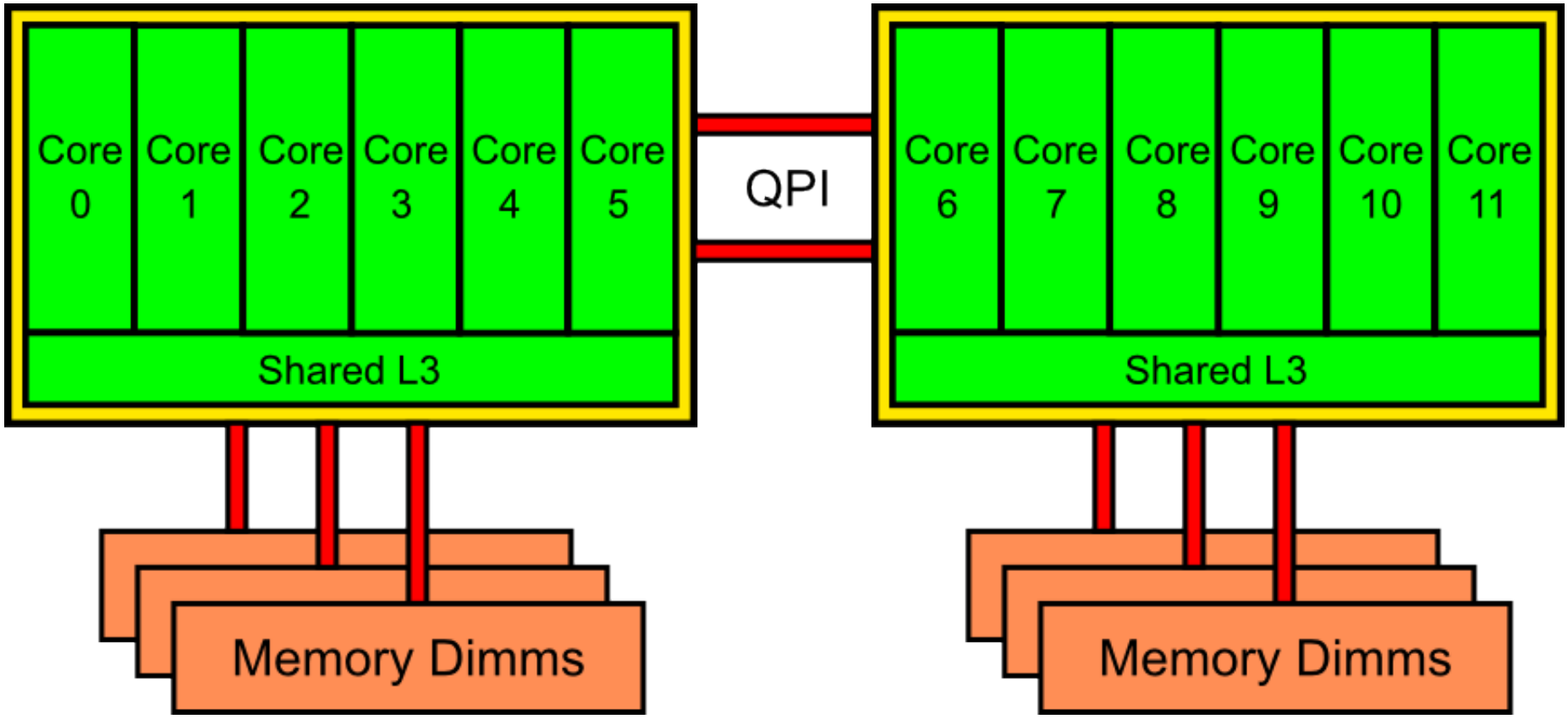
- Distributed Memory



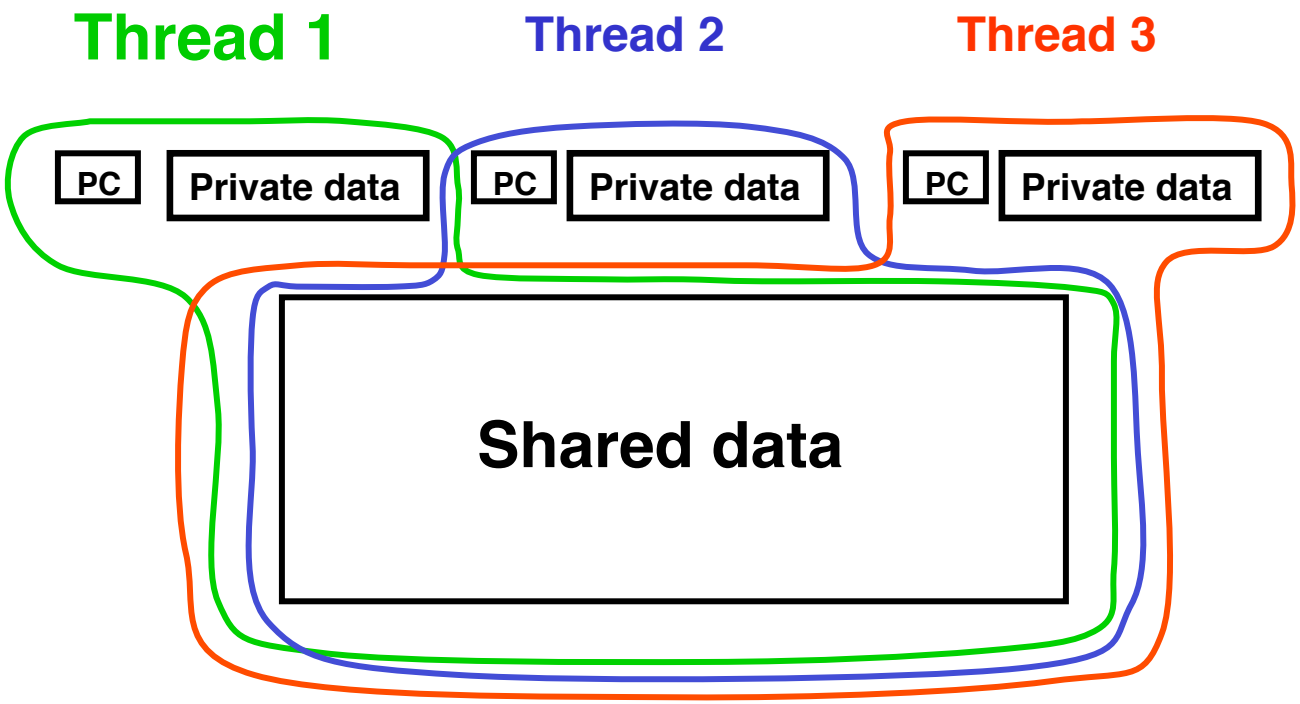
- Shared Memory



Multiple Socket CPUs



Paradigm at Shared Memory /1



Paradigm at Shared Memory /2

- Usually indicated as Multithreading Programming
- Commonly implemented in scientific computing using the OpenMP standard (directive based)
- Thread management overhead
- Limited scalability
- Write access to shared data can easily lead to race conditions and incorrect data

Parallel Programming Paradigms

- MPI (Message Passing Interface)
  - A standard defined for portable message passing
  - It available in the form of library which includes interfaces for expressing the data exchange among processes
  - A framework is provided for spawning the independent processes (i.e., mpirun)
  - Processes communication is via network
  - It works on either shared and distributed mem. architecture
  - ideal for distributing memory among compute nodes

MPI Program Design

- Multiple and separate processes (can be local and remote) concurrently that are coordinated and exchange data through “messages” => a “share nothing” parallelization
- Best for coarse grained parallelization Distribute large data sets; replicate small data
- Minimize communication or overlap communication and computing for efficiency => Amdahl's law

What is MPI?

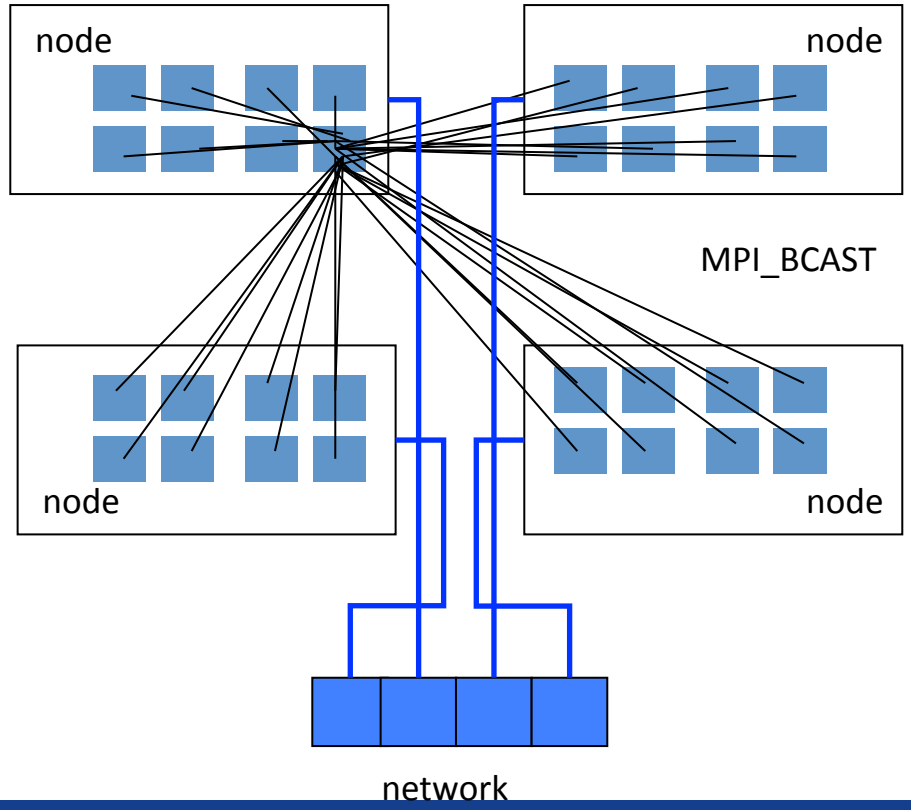
- A standard, i.e. there is a document describing how the API (constants & subroutines) are named and should behave; multiple "levels", MPI-1 (basic), MPI-2 (advanced), MPI-3 (new)
- A library or API to hide the details of low-level communication hardware and how to use it
- Implemented by multiple vendors
- Open source and commercial versions
- Vendor specific versions for certain hardware
- Not binary compatible between implementations

Programming Parallel Paradigms

- Are the tools we use to express the parallelism for on a given architecture
- They differ in how programmers can manage and define key features like:
  - parallel regions
  - concurrency
  - process communication
  - synchronism

MPI inter process communications

MPI on Multi core CPU



1 MPI proces / core
Stress network
Stress OS

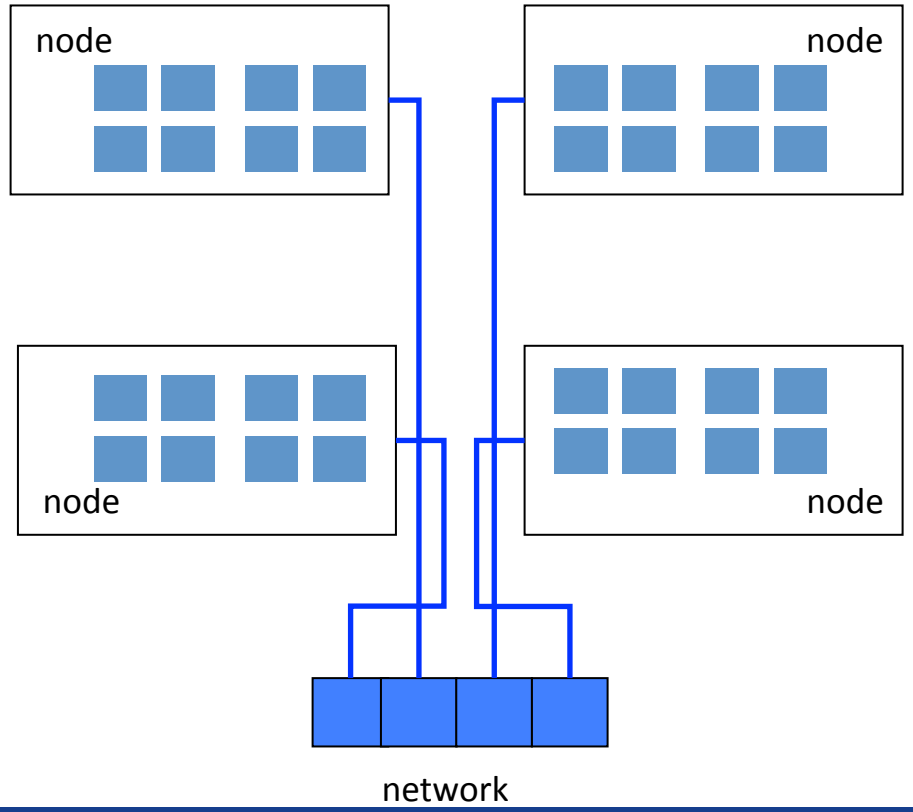
Many MPI codes (QE) based on ALLTOALL
Messages = processes * processes

We need to exploit the hierarchy

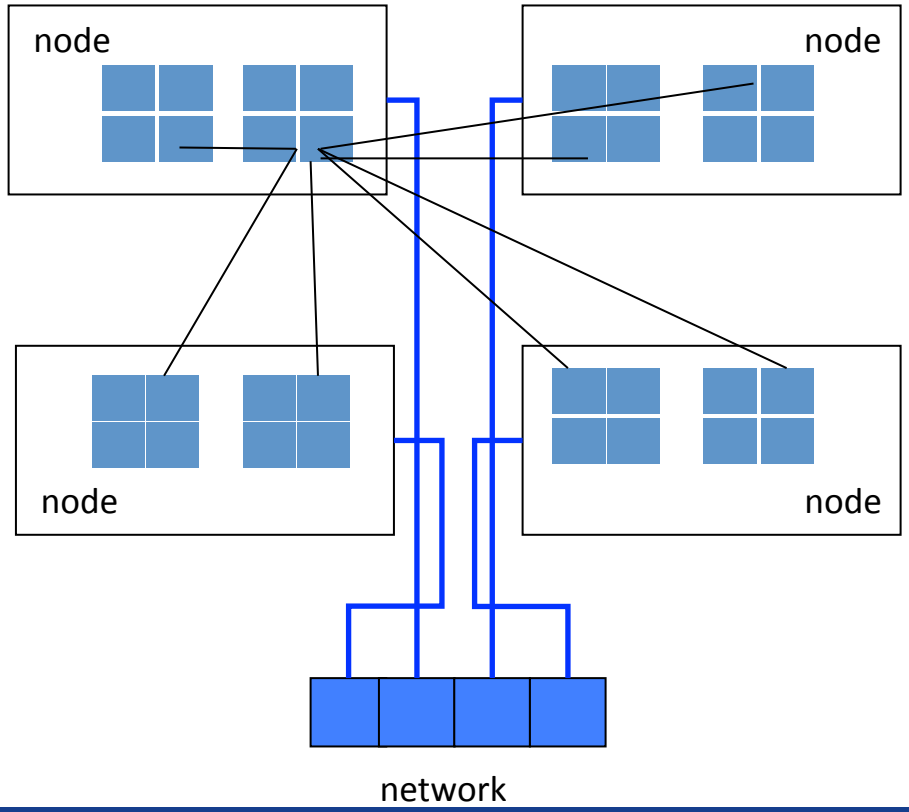
Re-design applications

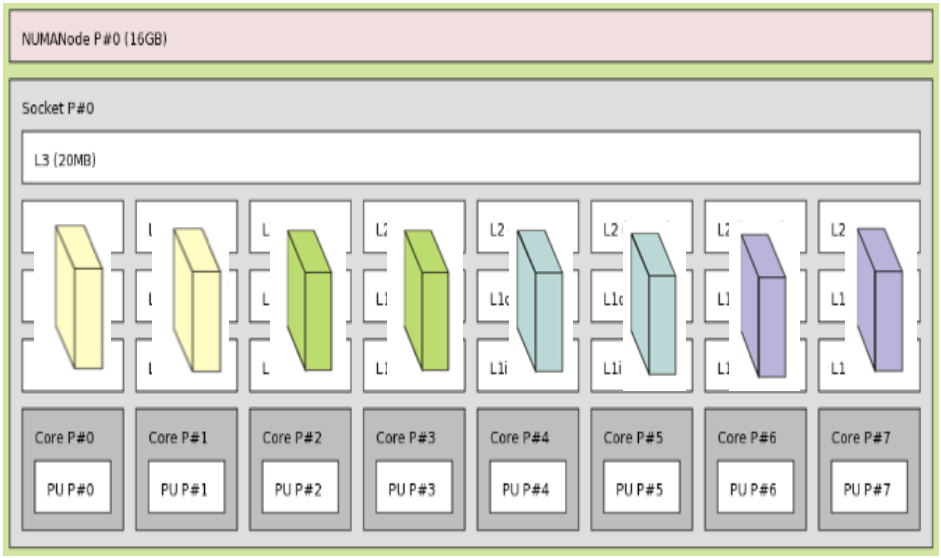
Mix message passing And multi-threading

The Hybrid Mode

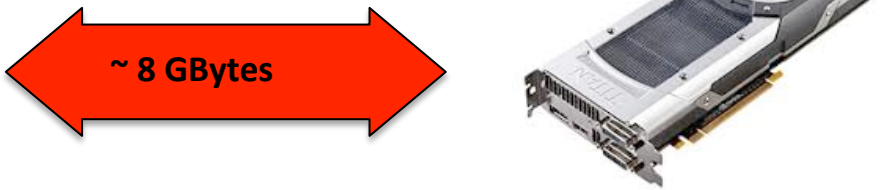


The Hybrid Mode

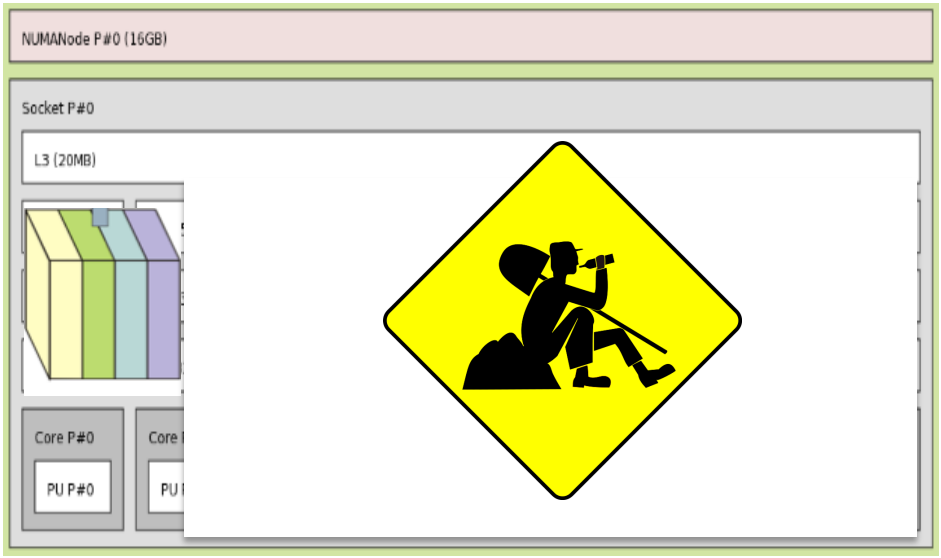




The Intel Xeon E5-2665
Sandy Bridge-EP 2.4GHz

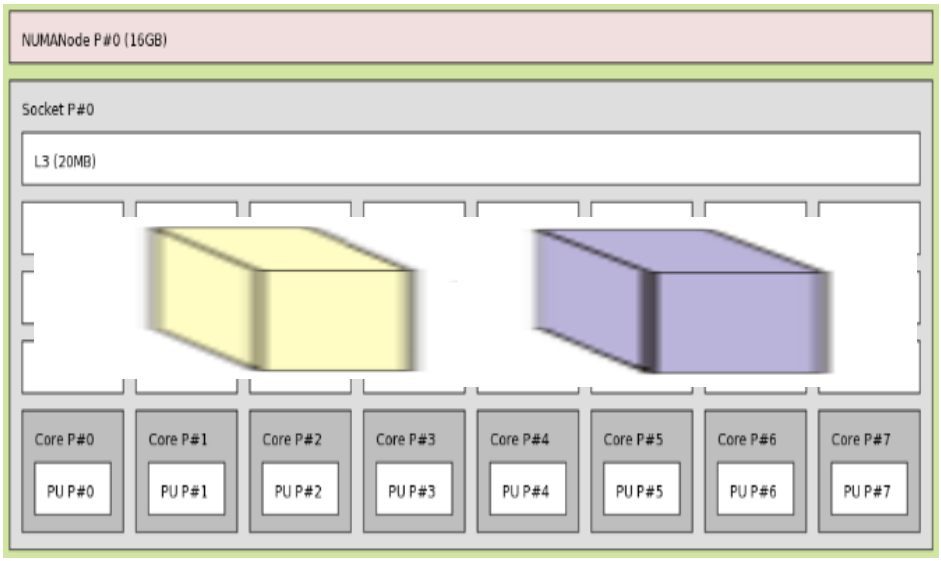



```
mpirun -np 8 pw-gpu.x -inp input file
```

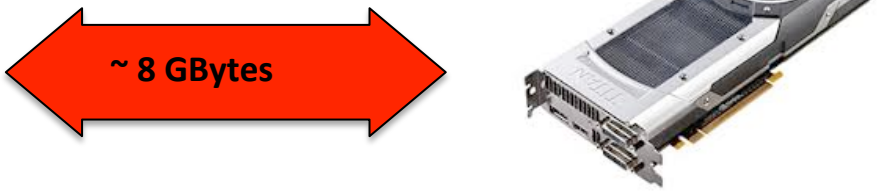


The Intel Xeon E5-2665
Sandy Bridge-EP 2.4GHz

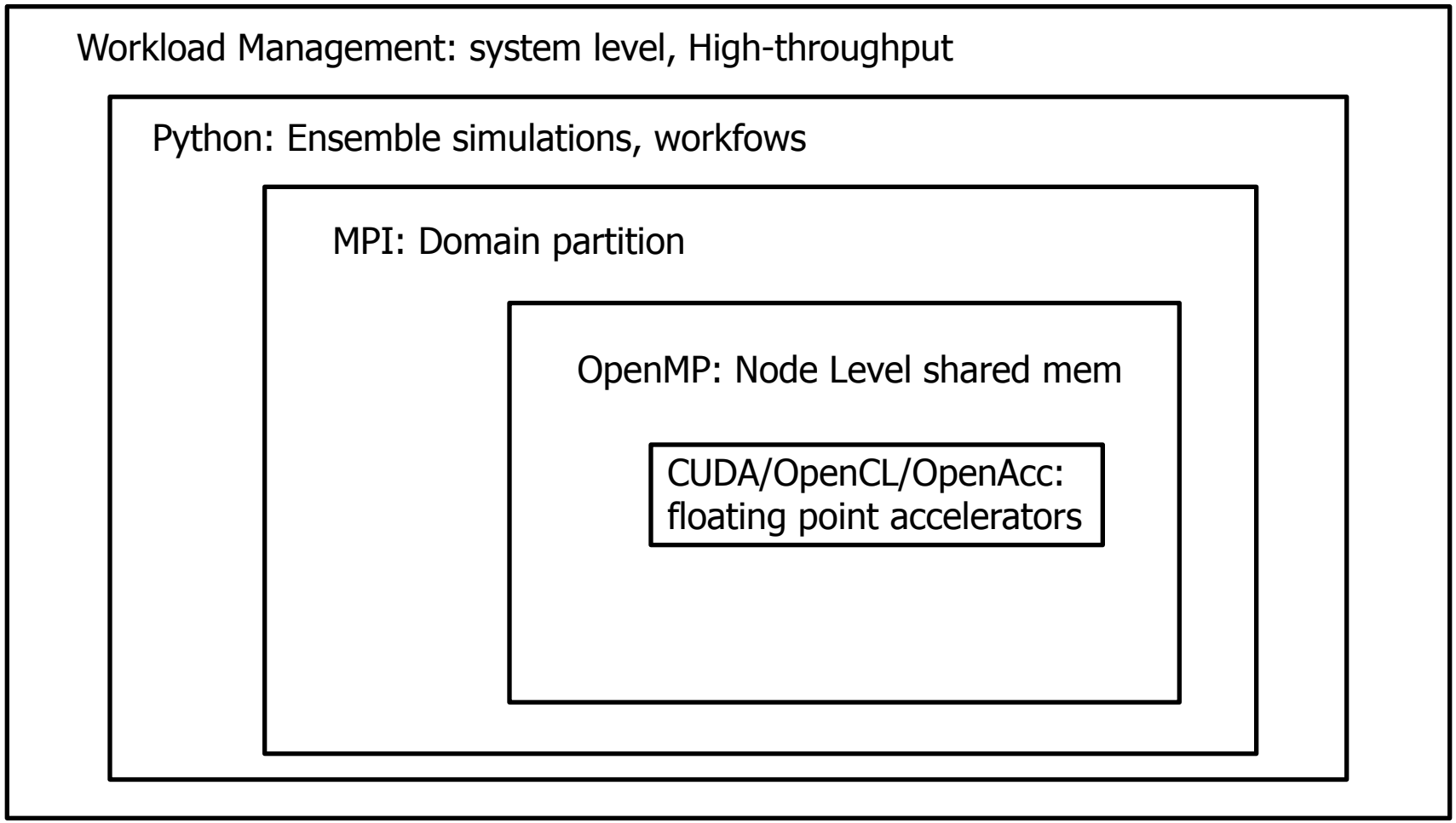
```
mpirun -np 1 pw-gpu.x -inp input file
```



The Intel Xeon E5-2665
Sandy Bridge-EP 2.4GHz




```
export OMP_NUM_THREADS=4
export OPENBLAS_NUM_THREADS=$OMP_NUM_THREADS
mpirun -np 2 pw-gpu.x -inp input file
```


Workload Management: system level, High-throughput
Python: Ensemble simulations, workfows
MPI: Domain partition
OpenMP: Node Level shared mem
CUDA/OpenCL/OpenAcc:
floating point accelerators

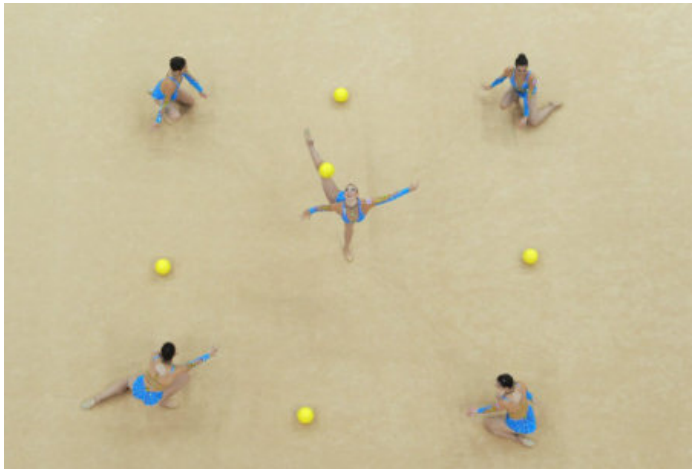
Type of Parallelism

- **Functional (or task) parallelism**: different people are performing different task at the same time

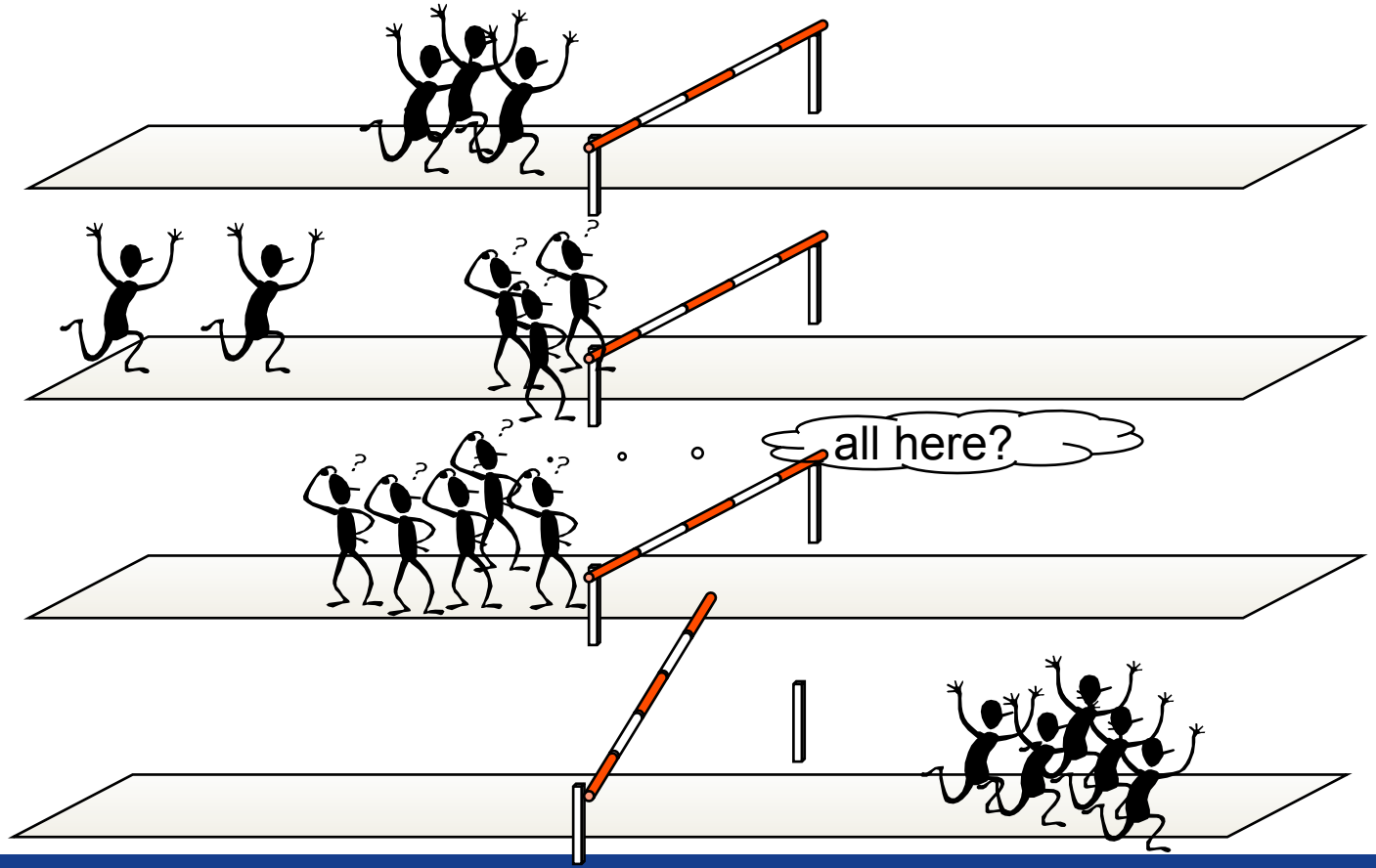
- **Data Parallelism**: different people are performing the same task, but on different equivalent and independent objects

Process Interactions

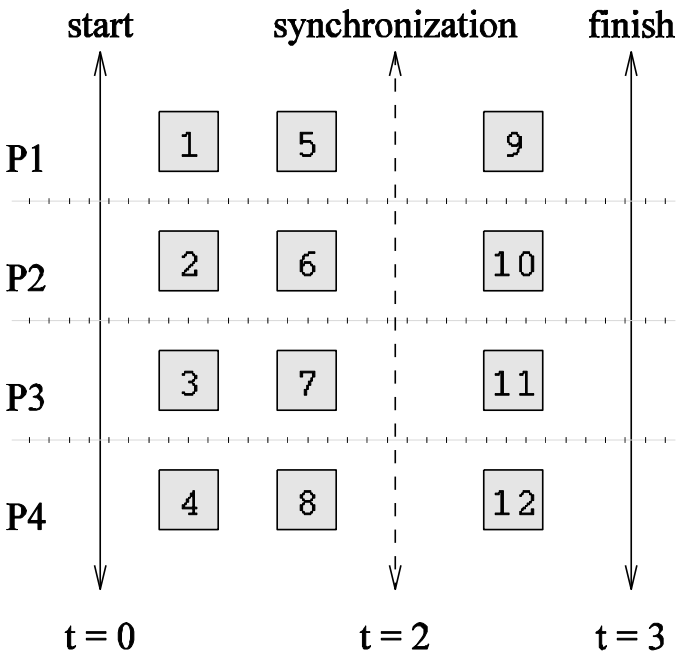
- The effective speed-up obtained by the parallelization depend by the amount of overhead we introduce making the algorithm parallel
- There are mainly two key sources of overhead:
  1. Time spent in inter-process interactions (**communication**)
  2. Time some process may spent being idle (**synchronization**)



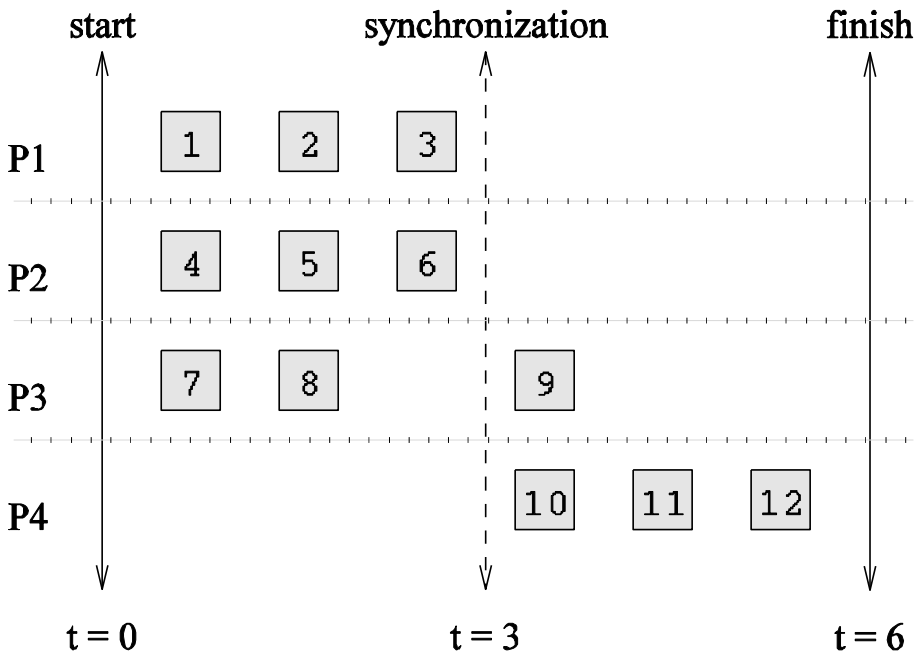
Effect of load-unbalancing



Mapping and Synchronization



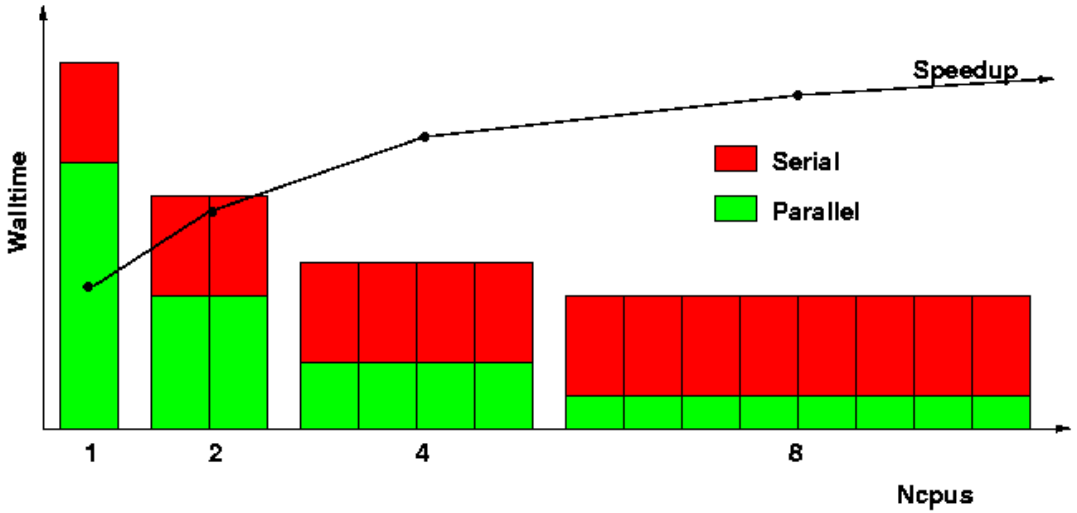
(a)



(b)

Amdahl's law

In a massively parallel context, an upper limit for the scalability of parallel applications is determined by the fraction of the overall execution time spent in non-scalable operations (Amdahl's law).



maximum speedup tends to
$1 / (1 - P)$
P= parallel fraction

1000000 core

P = 0.999999

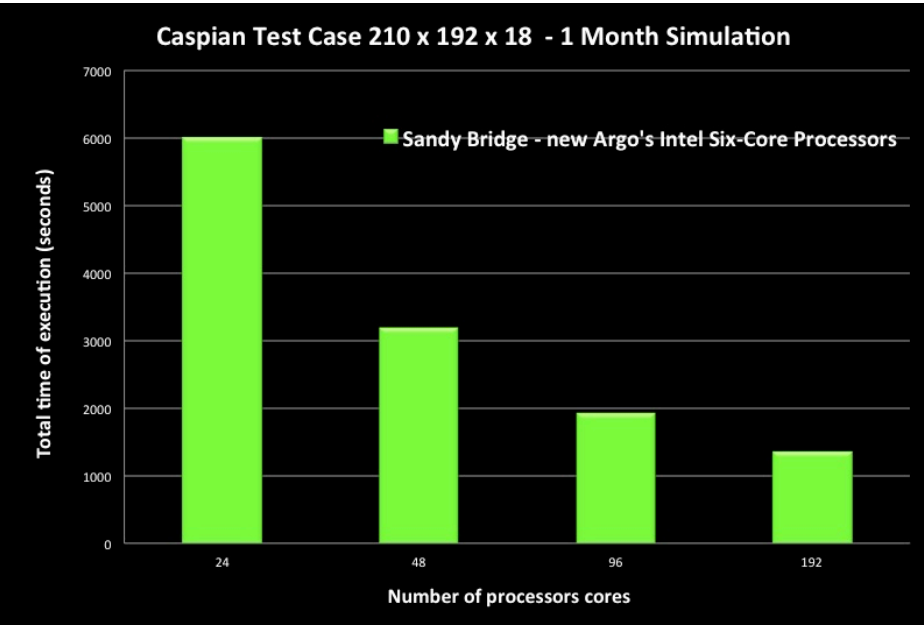
serial fraction= 0.000001

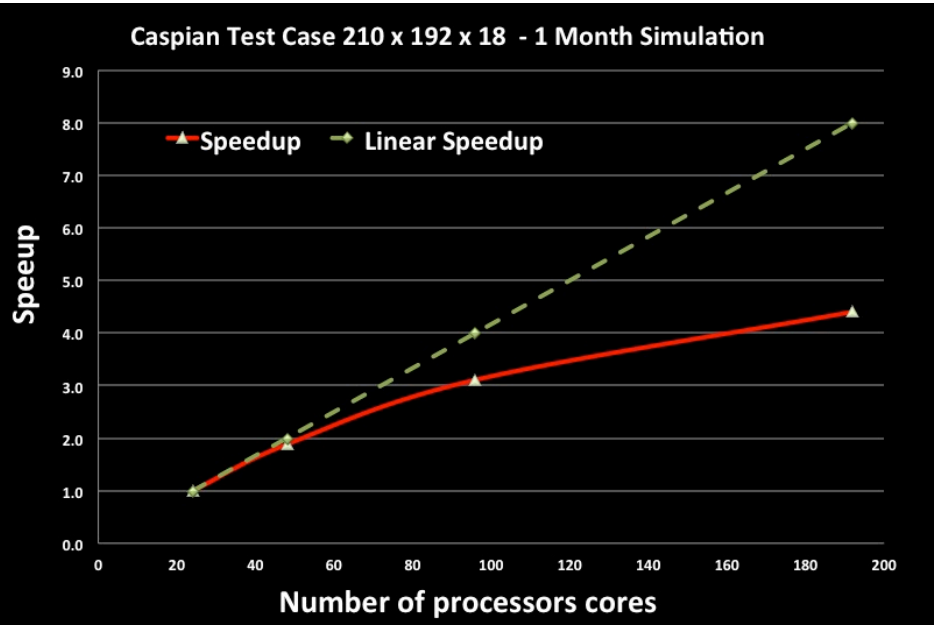
How do we evaluate the improvement?

- We want estimate the amount of the introduced overhead => $T_o = n_{pes} T_P - T_S$
- But to quantify the improvement we use the term **Speedup**:

$$S_P = \frac{T_S}{T_P}$$

Speedup


Caspian Test Case 210 x 192 x 18 - 1 Month Simulation
Sandy Bridge - new Argo's Intel Six-Core Processors
Total time of execution (seconds)
7000
6000
5000
4000
3000
2000
1000
0
24
48
96
192
Number of processors cores

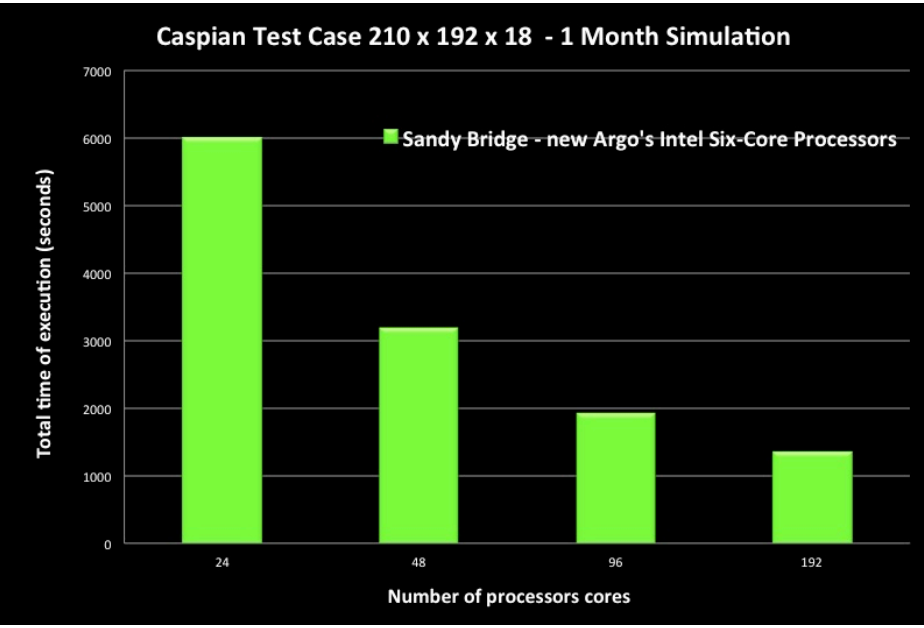

Caspian Test Case 210 x 192 x 18 - 1 Month Simulation
Speedup
Linear Speedup
Speeup
9.0
8.0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0.0
0
20
40
60
80
100
120
140
160
180
200
Number of processors cores

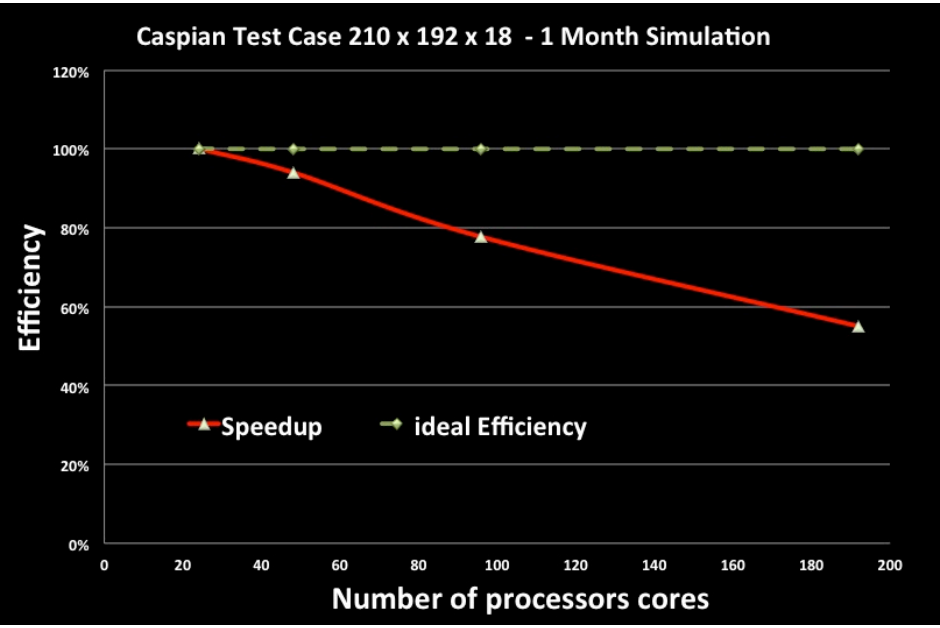
Efficiency

- Only embarrassing parallel algorithm can obtain an ideal Speedup
- The **Efficiency** is a measure of the fraction of time for which a processing element is usefully employed:

$$E_P = \frac{S_P}{p}$$

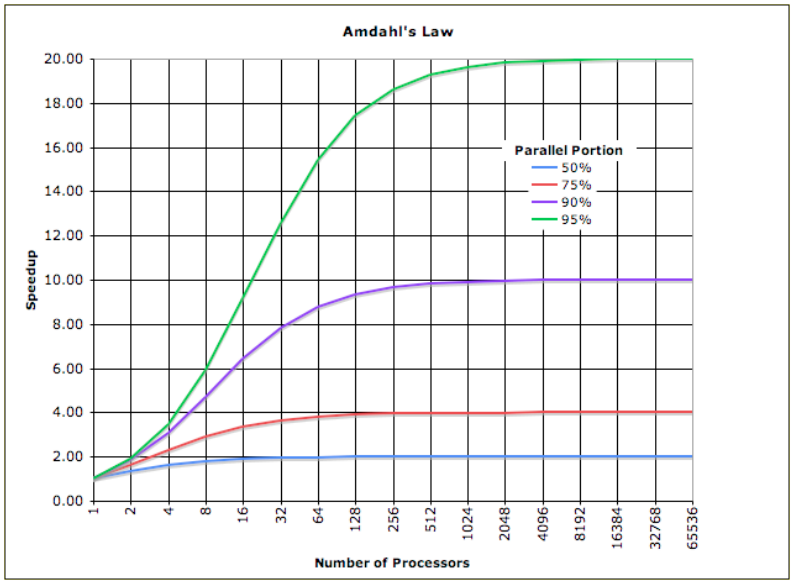
Efficiency

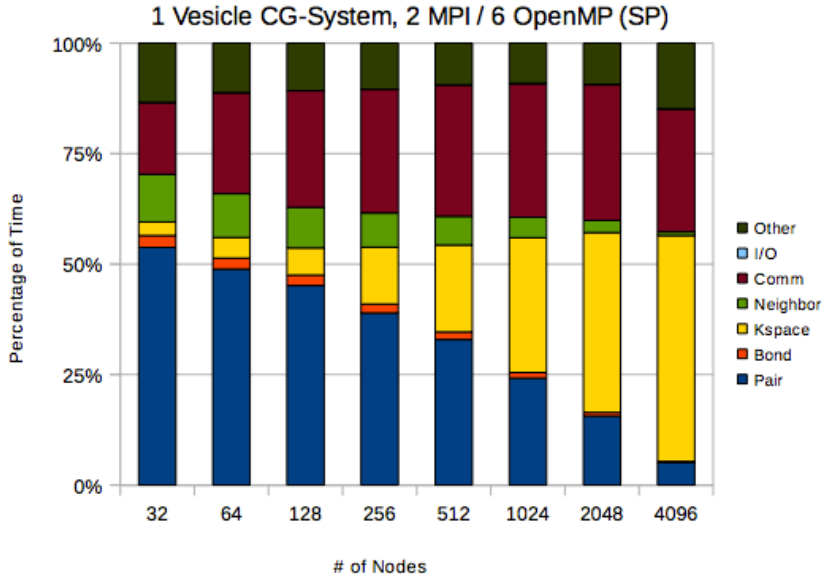

Caspian Test Case 210 x 192 x 18 - 1 Month Simulation
Total time of execution (seconds)
7000
6000
5000
4000
3000
2000
1000
0
Sandy Bridge - new Argo's Intel Six-Core Processors
24
48
96
192
Number of processors cores


Caspian Test Case 210 x 192 x 18 - 1 Month Simulation
Efficiency
120%
100%
80%
60%
40%
20%
0%
Speedup
ideal Efficiency
0
20
40
60
80
100
120
140
160
180
200
Number of processors cores

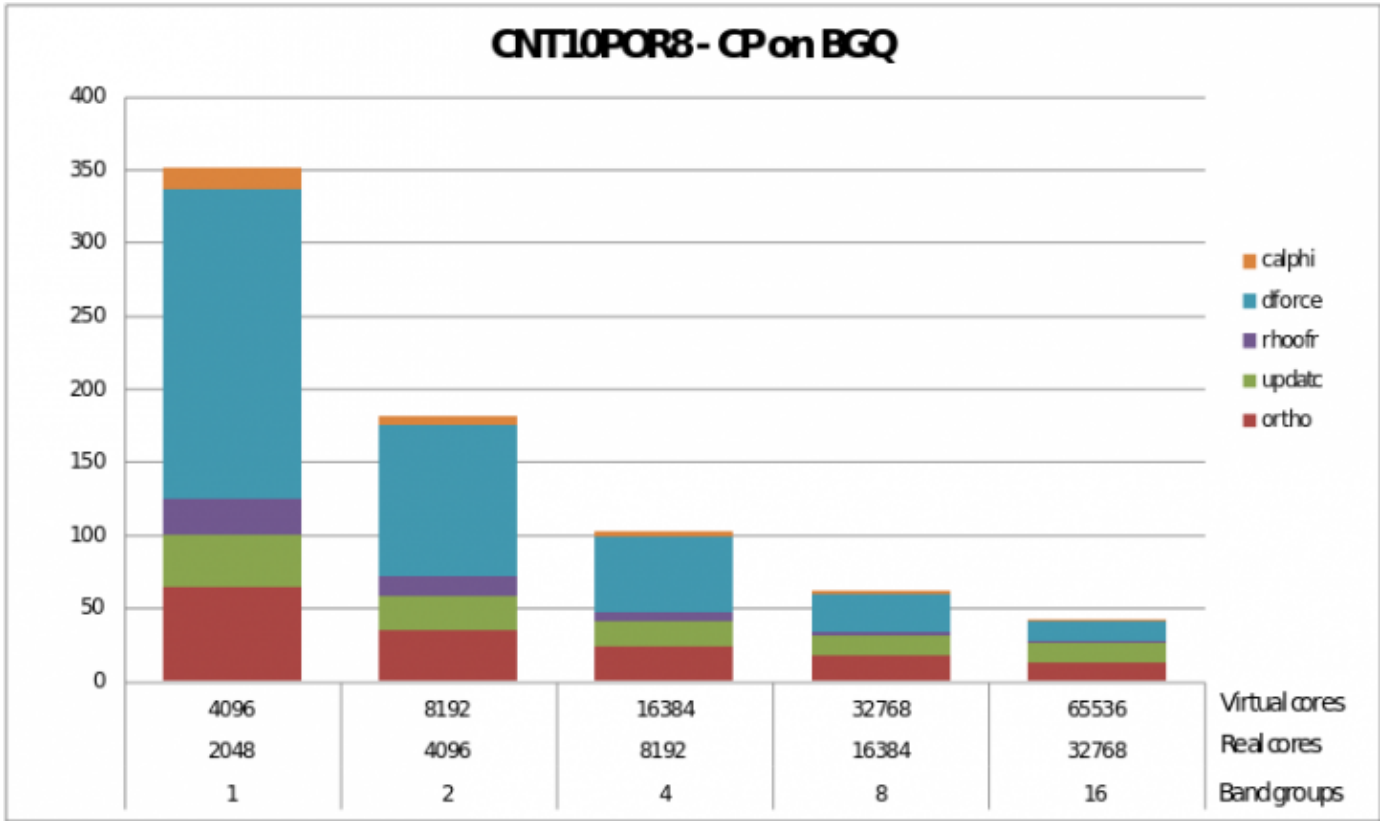
Amdal's Law And Real Life

- The speedup of a parallel program is limited by the sequential fraction of the program
- This assumes perfect scaling and no overhead





Scaling - QE-CP on Fermi BGQ @ CINECA



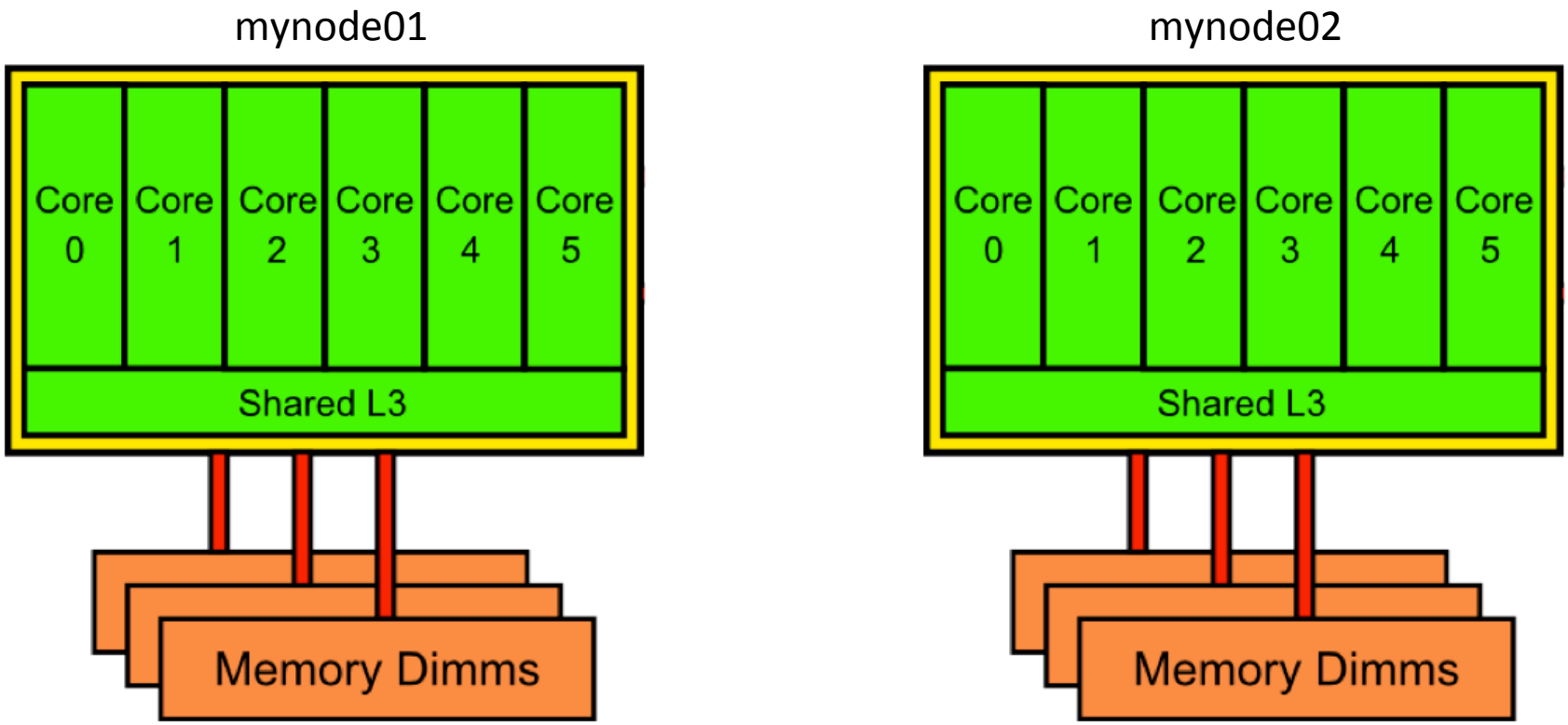
Easy Parallel Computing

- Farming, embarrassingly parallel
  - Executing multiple instances on the same program with different inputs/initial cond.
  - Reading large binary files by splitting the workload among processes
  - Searching elements on large data-sets
  - Other parallel execution of embarrassingly parallel problem (no communication among tasks)
- Ensemble simulations (weather forecast)
- Parameter space (find the best wing shape)

Single Program on Multiple Data

- performing the same program (set of instructions) among different data
- Same model adopted by the MPI library
- A parallel tool is needed to handle the different processes working in parallel
- The MPI library provides the *mpirun* application to execute parallel instances of the same program

```
$ mpirun -np 12 my_program.x
```



Ivan Girotto

```
[igirotto@mynode01 ~]$ mpirun -np 12 /bin/hostname
mynode01
mynode02
mynode01
mynode02
mynode01
mynode02
mynode01
mynode02
mynode01
mynode02
mynode01
mynode02
```

PATH name common to all processes !!

Parallel Operations in Practice

- Parallel reading and computing in parallel is always allowed
- Parallel writing is extremely dangerous!
- To control the parallel flow each process should be unique and identifiable (ID)
- The OpenMPI implementation of the MPI library provides a series of environment variables defined for each MPI process

OMPI_COMM_WORLD_SIZE - the number of processes in this process' MPI Comm_World

OMPI_COMM_WORLD_RANK - the MPI rank of this process

OMPI_COMM_WORLD_LOCAL_RANK - the relative rank of this process on this node within its job. For example, if four processes in a job share a node, they will each be given a local rank ranging from 0 to 3.

OMPI_UNIVERSE_SIZE - the number of process slots allocated to this job. Note that this may be different than the number of processes in the job.

OMPI_COMM_WORLD_LOCAL_SIZE - the number of ranks from this job that are running on this node.

OMPI_COMM_WORLD_NODE_RANK - the relative rank of this process on this node looking across ALL jobs.

http://www.open-mpi.org

In Python

```
import os
myid = os.environ['OMPI_COMM_WORLD_RANK']
[...]
```

In BASH

```
#!/bin/bash
myid=${OMPI_COMM_WORLD_RANK}
[...]
```

```
[igirotto@mynode01 ~]$ mpirun ./myprogram.[py/sh...]
```

Possible Applications

- Executing multiple instances on the same program with different inputs/initial cond.
- Reading large binary files by splitting the workload among processes
- Searching elements on large data-sets
- Other parallel execution of embarrassingly parallel problem (no communication among tasks)

Conclusions

- Task Farming is a simple model to parallelize simple problems that can be divided in independent task
- The *mpirun* application aids to easily perform multiple processes, includes environment setting
- Load balancing remains a main problem, but moving from serial to parallel processing can substantially speed-up time of simulation

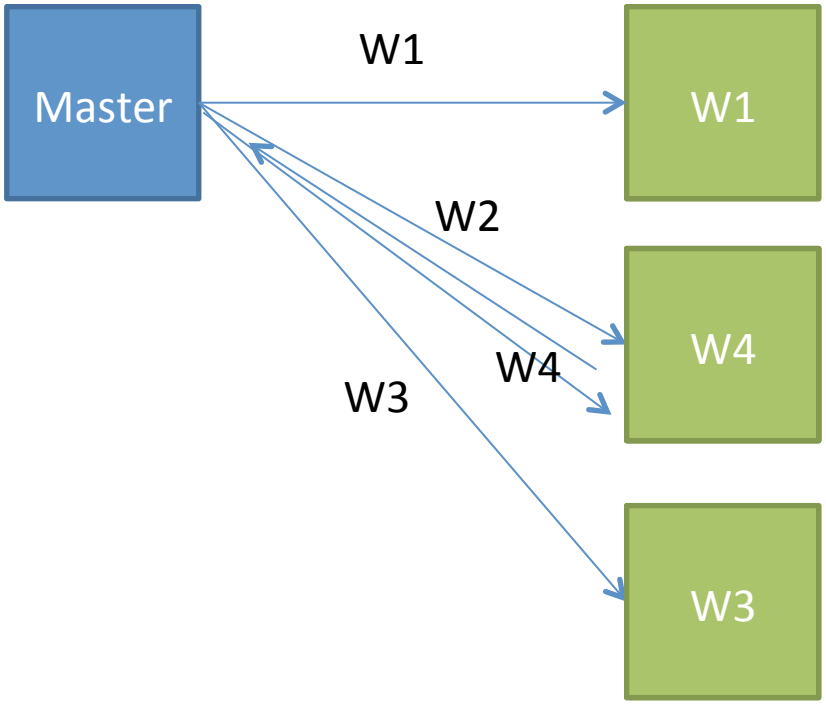
Task Farming

- Many independent programs (tasks) running at once
  - each task can be serial or parallel
  - "independent" means they don't communicate directly
  - Processes possibly driven by the mpirun framework

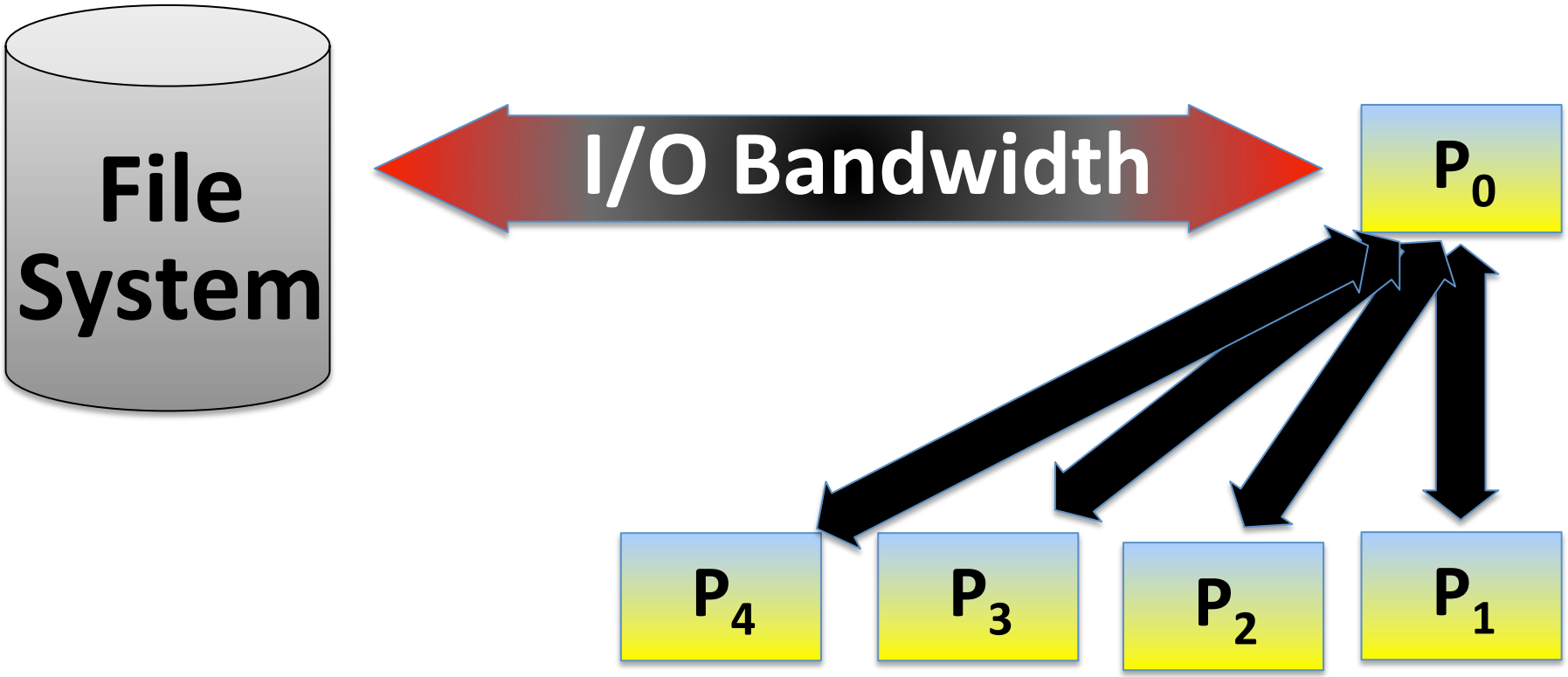
```
[igirotto@localhost]$ more my_shell_wrapper.sh
#!/bin/bash
#example for the OpenMPI implementation
./prog.x --input input_${OMPI_COMM_WORLD_RANK}.dat

[igirotto@localhost]$ mpirun -np 400 ./my_shell_wrapper.sh
```

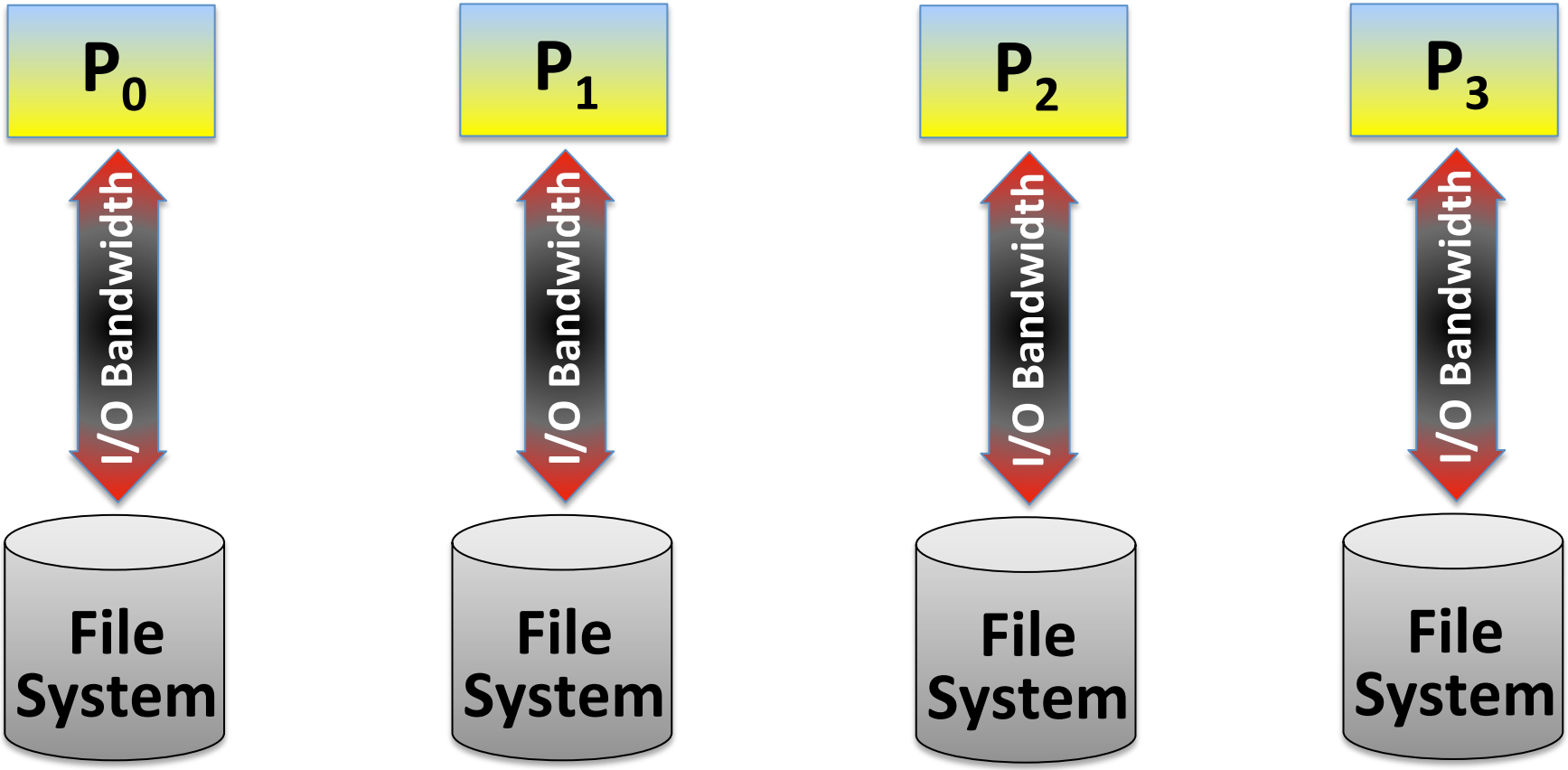
Master/Slave



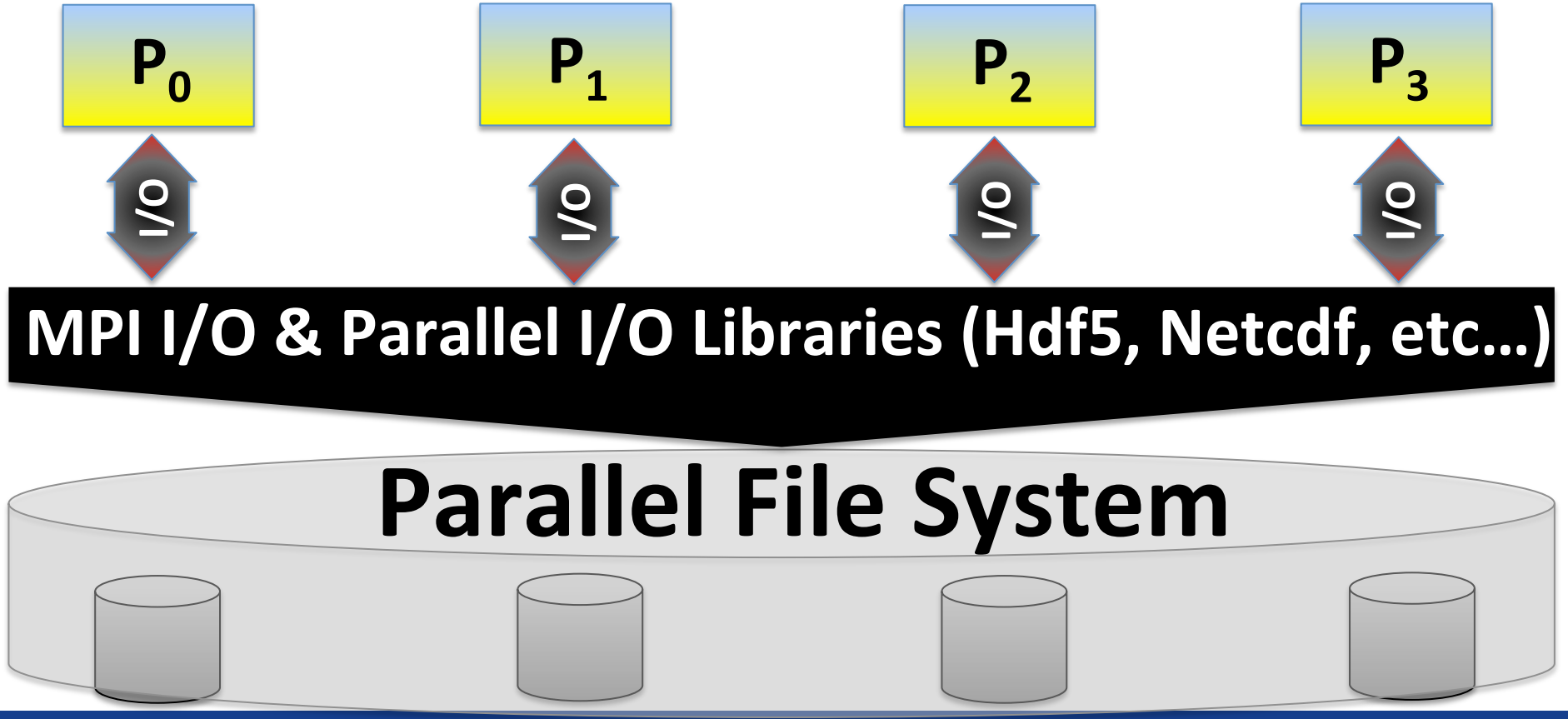
Parallel I/O



Parallel I/O


P0
P1
P2
P3
I/O Bandwidth
I/O Bandwidth
I/O Bandwidth
I/O Bandwidth
File System
File System
File System
File System

Parallel I/O


P0
P1
P2
P3
I/O
I/O
I/O
I/O
MPI I/O & Parallel I/O Libraries (Hdf5, Netcdf, etc…)
Parallel File System

What If You Want to Learning How to Program All This?!

- Introductory School on Parallel Programming and Parallel Architecture for High Performance Computing | (smr 2877)
- 3 October 2016 - 14 October 2016

What If You Want to Master All This?!

Introductory School on Pa... | Welcome to Master in Hig...

www.mhpc.it

MHPC

Portal_ICTP | Helpdesk | RT | ICTP | PhoneBook | Google_Calendar | SIS | Google | GoogleMap | Tuttocitta' | ICTP - Time and ... | ICTP - FABS Help ... | CPU-Bench

MHPC
Master in High Performance Computing

SISSA | ICTP The Abdus Salam International Centre for Theoretical Physics

ABOUT | OBJECTIVES | COURSES | APPLY | FAQ | PEOPLE | SPONSORS

MHPC

The Master in High-Performance Computing (MHPC) is a high-level degree program that aims to train students to solve complex problems with HPC techniques.

WHY

Set in a stimulating research environment, the MHPC is an innovative, hands-on training and education program to prepare students for exciting careers in the fast-growing field of HPC.

THE TARGET

The master is intended for people with strong interest in advanced programming for scientific computing, software optimization and management of computing platforms.

READ ALL

Thanks for your attention!!