Introduction to Parallel Computing

S. Van Criekingen
UPJV / MeCS

November 13, 2014

Layout

- Why parallel computing?
- Parallel architectures
- Parallel programming models
- Parallel programming tools
- Parallel efficiency
- Computer performance measurements
- Existing HPC infrastructures & MeCS

Layout

- Why parallel computing?
- Parallel architectures
- Parallel programming models
- Parallel programming tools
- Parallel efficiency
- Computer performance measurements
- Existing HPC infrastructures & MeCS

Serial Computing (*Calcul séquentiel*)

Serial Computing

Performance $\propto$ frequency clock rate *(fréquence d'horloge)*

Limits to Serial Computing

Increasing frequency more and more difficult mainly because of:

- **electrical consumption** (~ freq. ^ 3) and thermal dissipation.
- Data movement within CPU limited:

absolute limit = **speed of light** = 30 cm/nsec

→ limits on chip size, while miniaturization is limited.

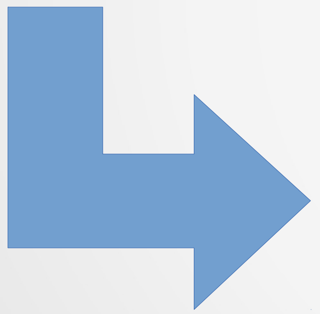
➔ use several CPUs simultaneously to improve performance.

Note: Several moderately fast CPU can be cheaper than a very fast one.

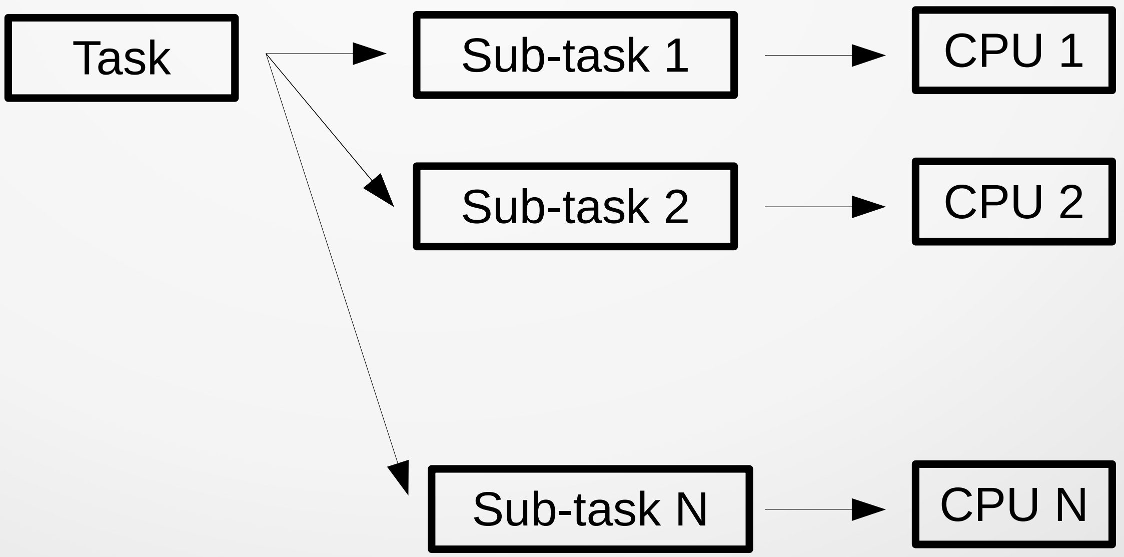
Serial vs. Parallel Computing

Serial Computing

$$\begin{tikzcd}\boxed{\text{Task}} \arrow[r] & \boxed{\text{CPU}}\end{tikzcd}$$



Parallel Computing



Parallel Computing

Software codes must be **parallelized** to
take advantage of parallel performance increase.

« *The Free Lunch Is Over* »
Herb Sutter, 2005

Note on (sometimes confused) terminology

A ***CPU*** is sometimes also named a ***processor.***

These days, a ***CPU*** / ***processor*** has typically several computing units sometimes called "***CPU cores***" or "***processor cores".***

Here ***CPU*** is used to designate a ***single*** computing unit (***one core***).

Layout

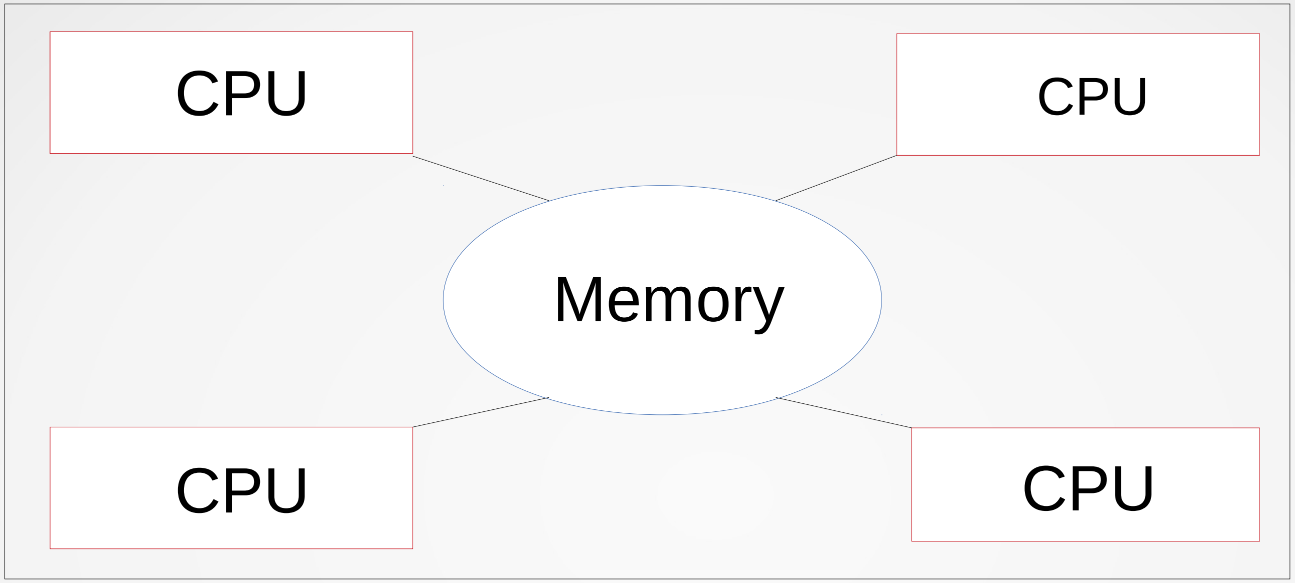
- Why parallel computing?
- Parallel architectures
- Parallel programming models
- Parallel programming tools
- Parallel efficiency
- Computer performance measurements
- Existing HPC infrastructures & MeCS

Architecture of Parallel Computers

Architecture defined by **memory handling**:

- Shared-Memory architecture
- Distributed-Memory architecture
- Mixed (or hybrid) architecture = shared + distrib^{ed}
- Hybrid architecture including "accelerators".

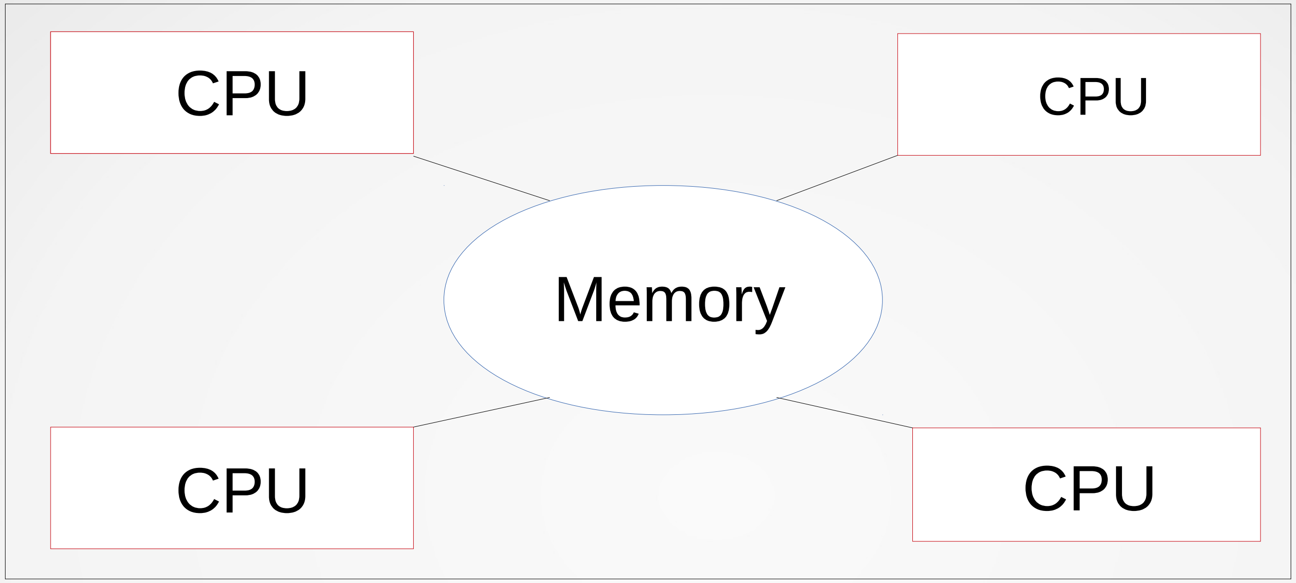
Shared-Memory architectures



Direct access to the whole memory for all the CPUs
(global address space - *espace d'adressage global*)

If one CPU modifies a variable in memory,
all the CPUs know about it.

Shared-Memory architectures

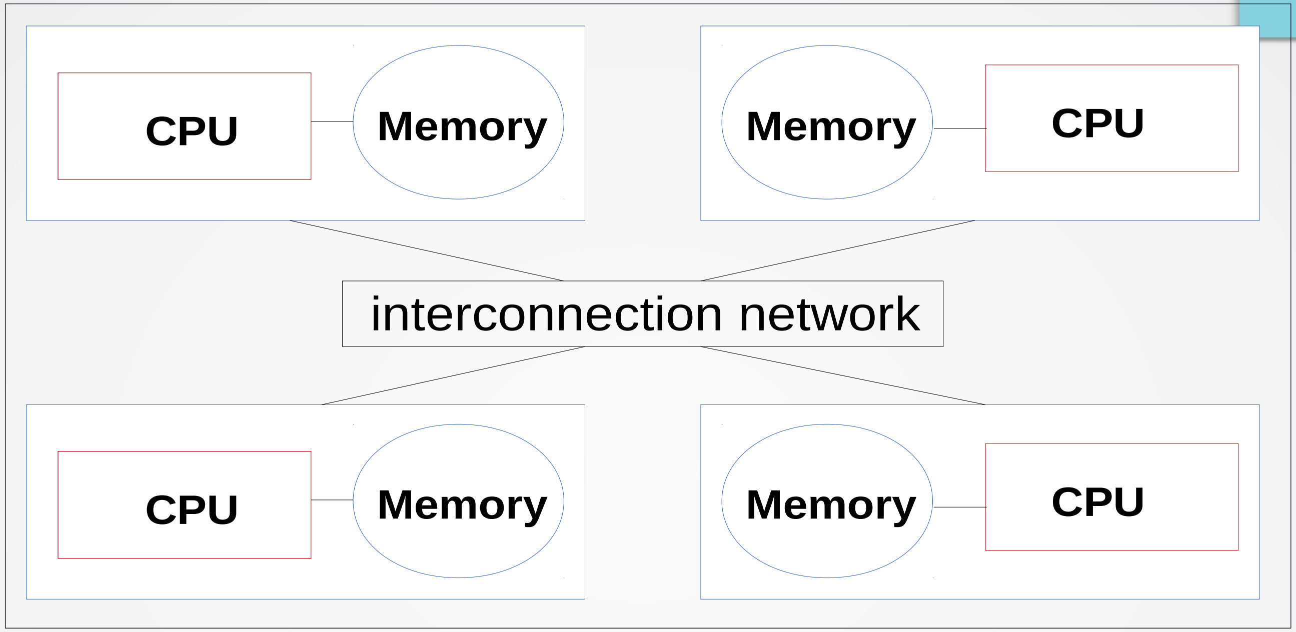


Notes :

• Access speed to different memory zones can differ : Uniform / Non-Uniform Memory Access (UMA / NUMA)

• Shared-memory architectures also known as **Symmetric Multi-Processing (SMP)**.

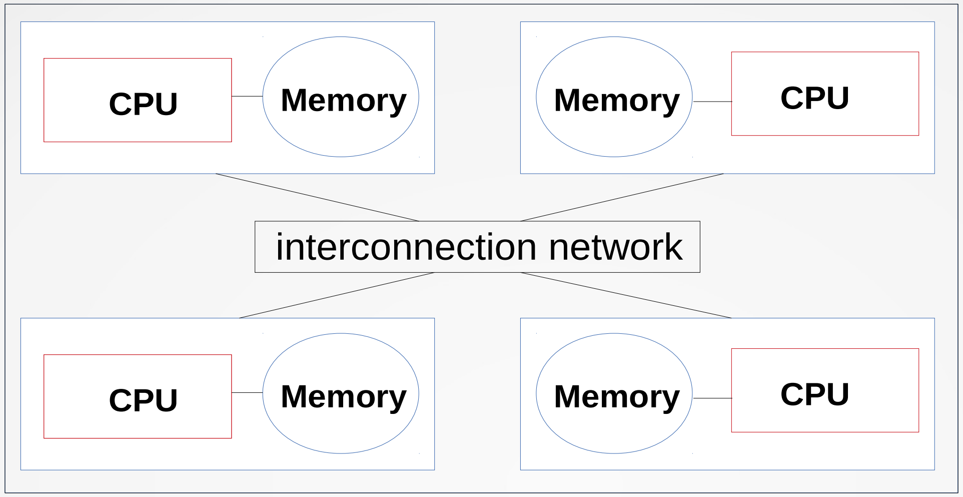
Distributed-Memory architectures



Whole memory access through **interconnection network** : **message passing** necessary (no global adress space).

If one CPU modifies a variable in its memory, the other CPUs do **not** know about it.

Distributed-Memory architectures



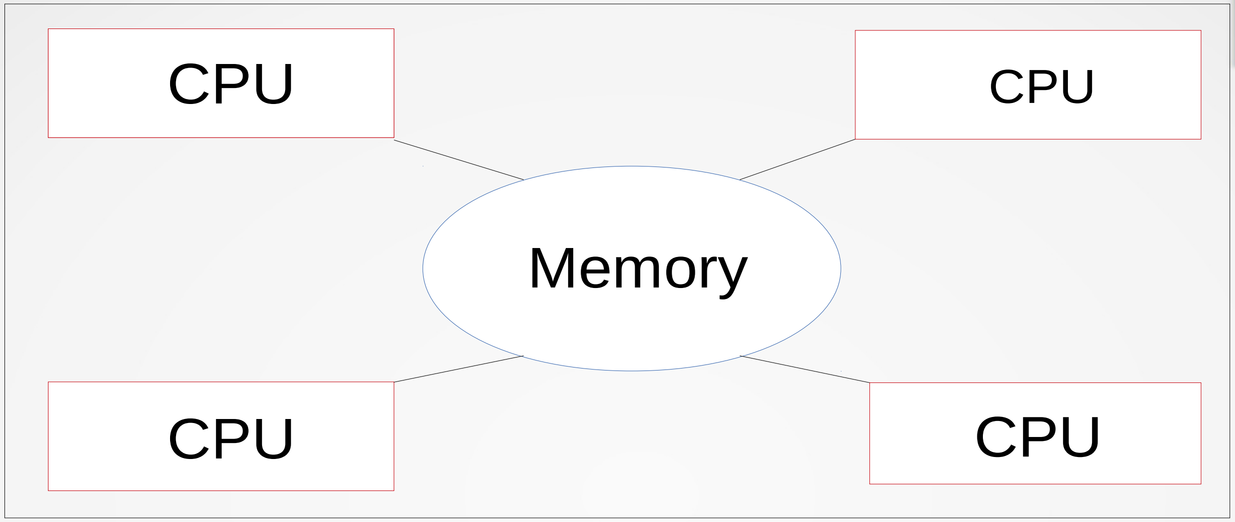
Interconnection network determines **communication speed**

Characterized by:

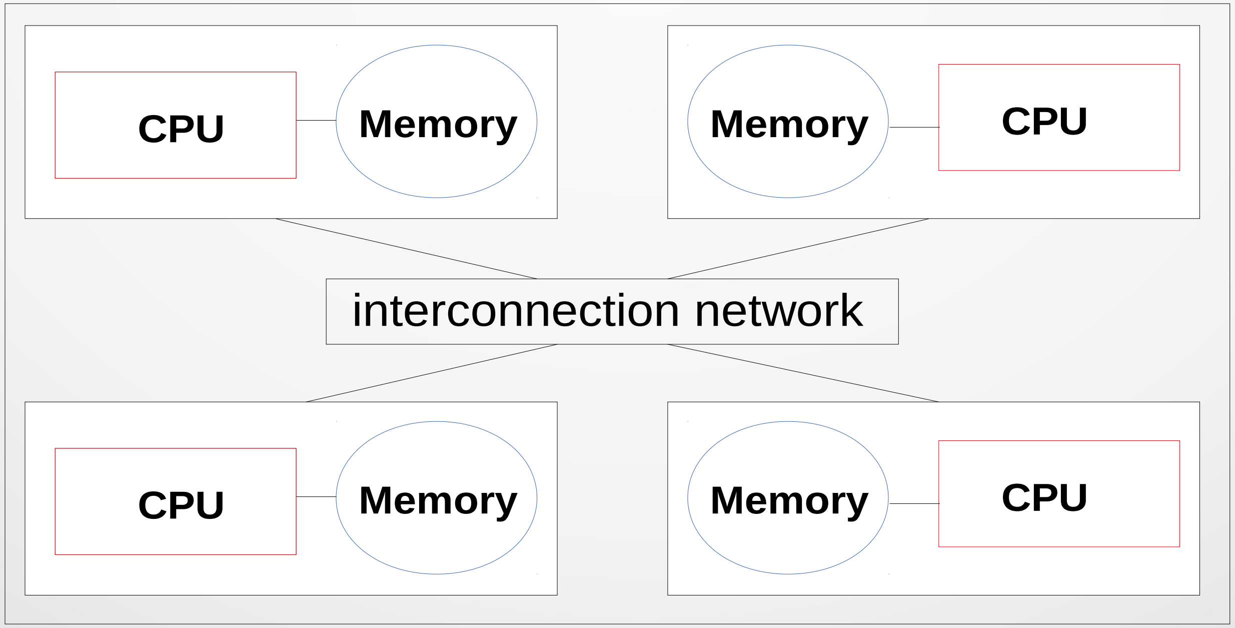
- **Latency** = time to initiate communication (microseconds)
- **Bandwitdh** = amount of data transferable per unit time (gigabytes/sec)
- **Topology** = physical layout (tree, star, ring,...)

Pros and Cons

Shared



Distributed



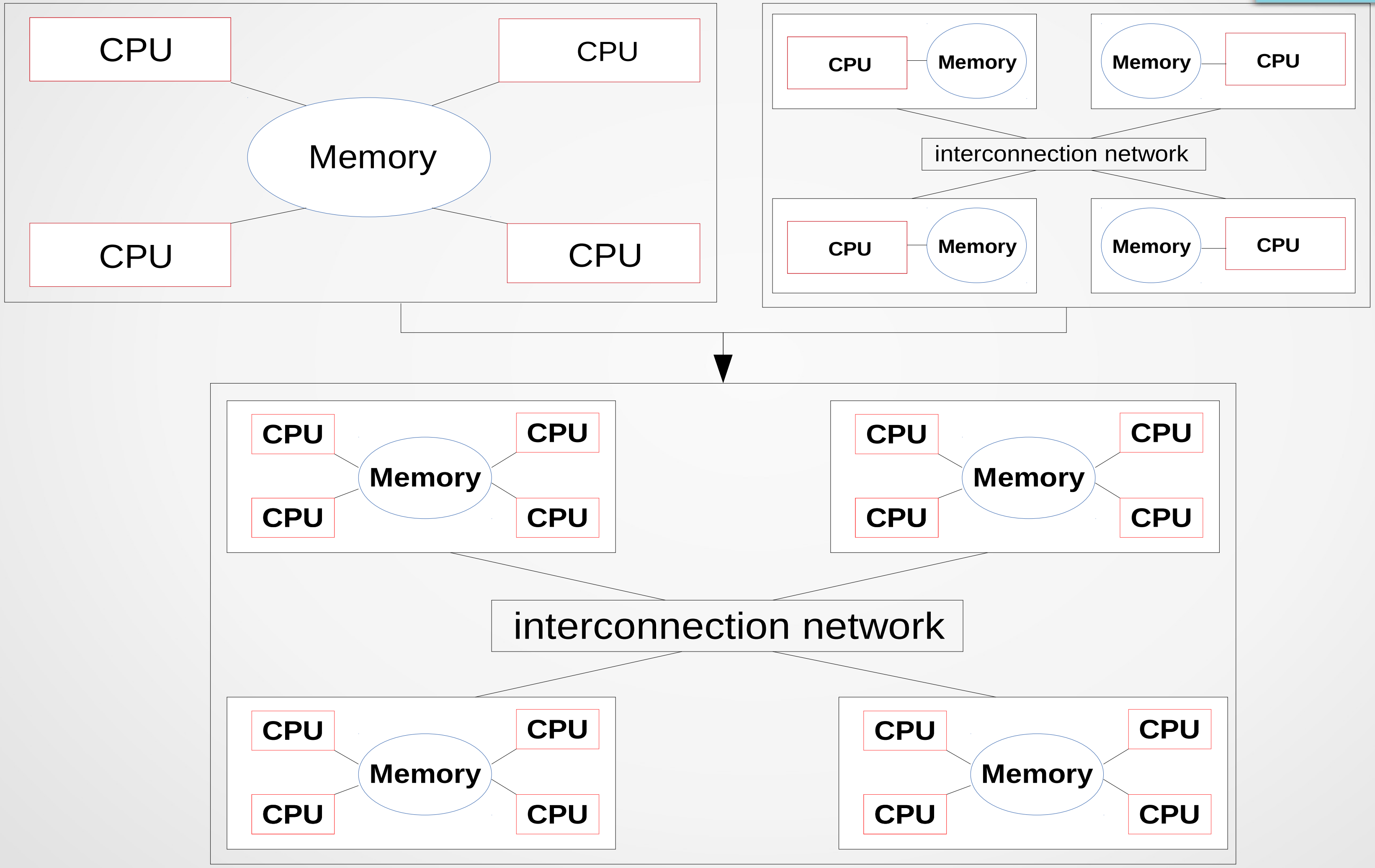
Shared vs. Distributed Memory

- Programming **easier with Shared-Memory** (no message passing between CPUs)

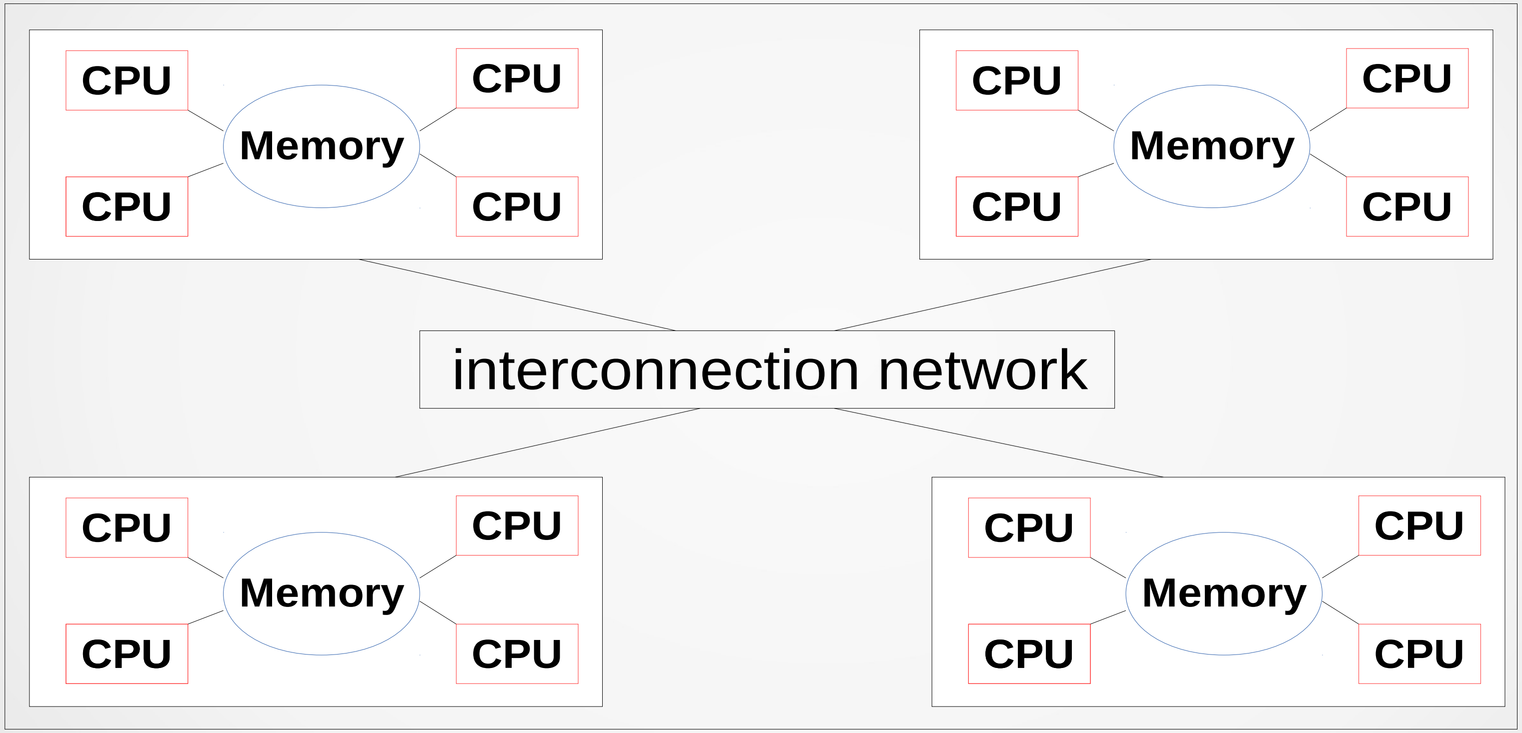
- Shared-Memory calculators are **limited in size**: memory access time and calculator price grow rapidly with number of CPUs.
  Currently up to about 1000 CPUs (288 @ UPJV).

- Distributed-Memory calculator enable to increase the number of processors at a relatively lower price.
  Performance depends on interconnexion network.

Mixed (or hybrid) Architectures



Mixed (or hybrid) Architectures



cluster of compute **nodes** (or servers)

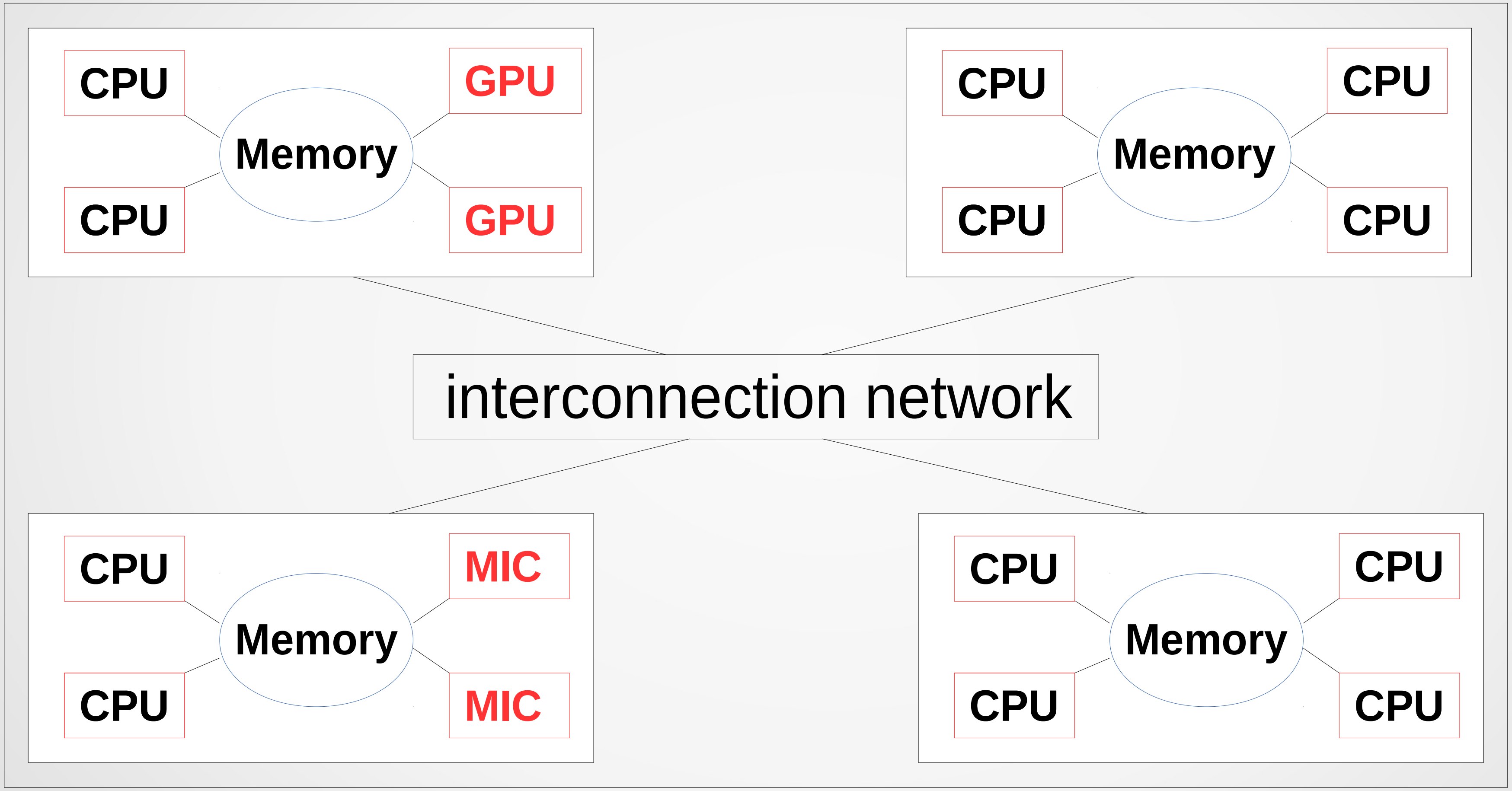
"accelerators" (or "coprocessors")

Nowadays, accelarators available besides CPU:

- **GPU (Graphic Processing Unit)**
  - Originally developped for graphic rendering (video games)
  - Inherently massively parallel, simple operations
  - Now programmable for more complex tasks "General-purpose GPU" (GPGPU)

- **MIC (Many Integrated Cores)**
  - Many traditional CPU cores on a single chip (60^+ in Intel "Xeon Phi")

Hybrid Architectures



Todays' fastest computers have this architecture

Layout

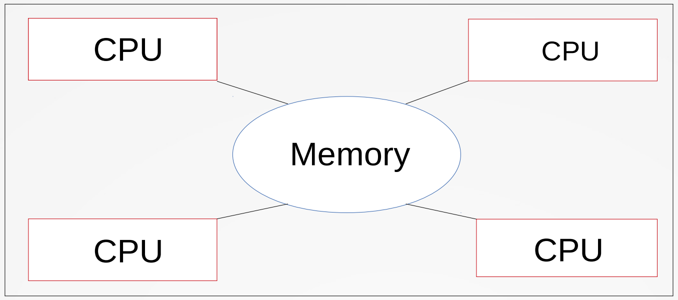
- Why parallel computing?
- Parallel architectures
- Parallel programming models
- Parallel programming tools
- Parallel efficiency
- Computer performance measurements
- Existing HPC infrastructures & MeCS

Programming models

- For shared memory: multi-threading
- For distributed memory: message passing
- On mixed architecture: mixed programming

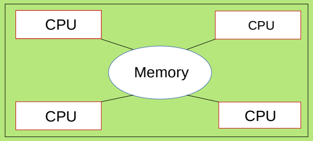
  message passing + multi-threading
- Hybrid programming with accelerators

Multithreading (shared memory)

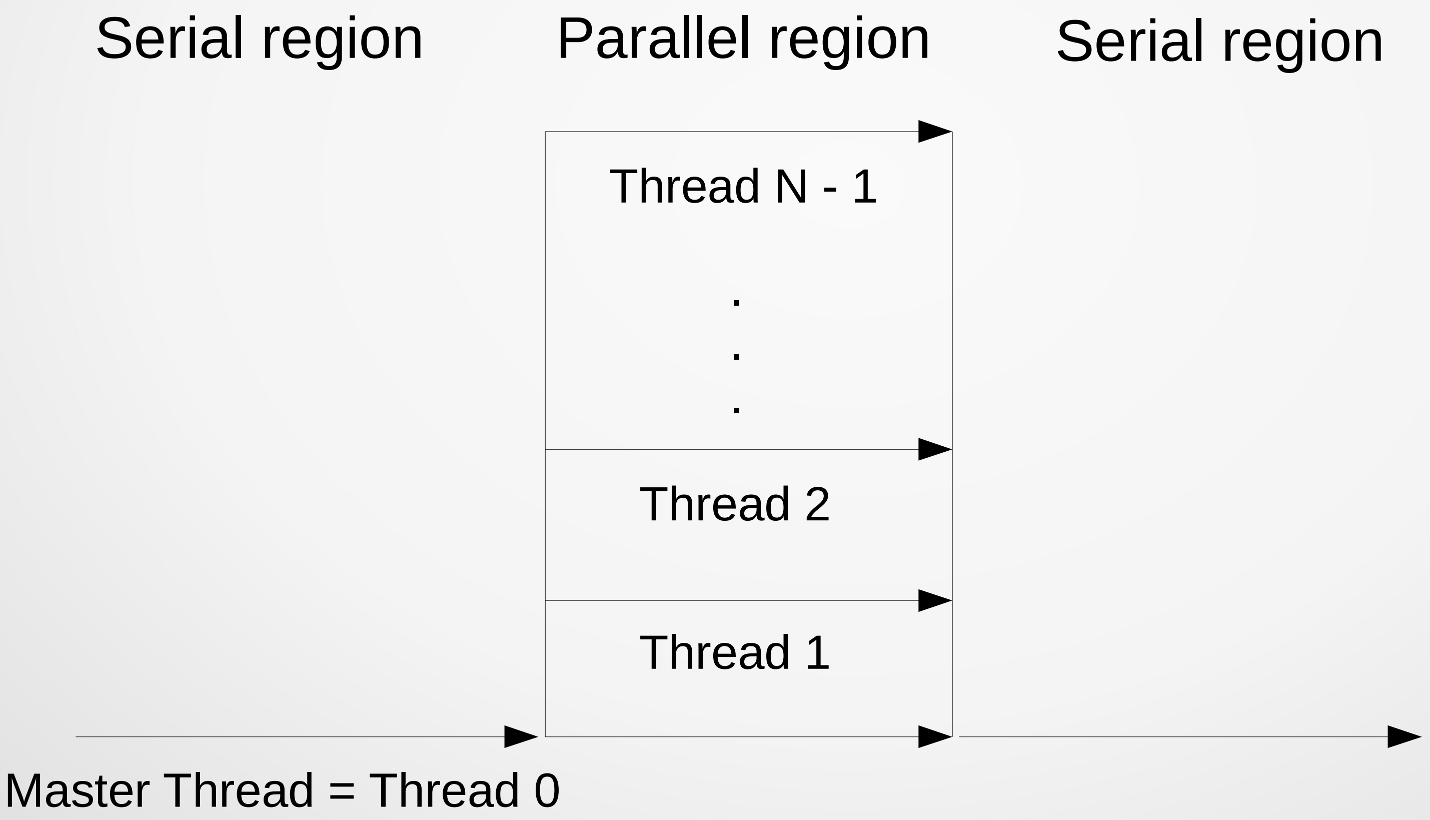


A unique process activates several **threads (*processus légers*)** acting concurrently within the **shared memory**.

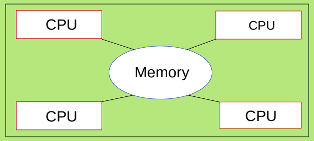
"Fork-join" model



FORK: master thread creates a team of parallel threads
JOIN: threads synchronize at the end of parallel region; then only the master thread continues.



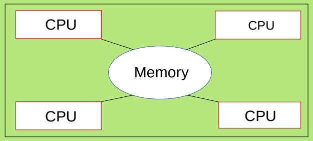
Multithreading API's



Application Program Interfaces (API's) enabling multithreading:

- **OpenMP (open Multi-Processing)**
- Pthreads (POSIX threads)
- TBB (Thread Building Blocks)
- ...

OpenMP "Hello world" (in C)



C code

```
#include <omp.h>
#include <stdio.h>
main () {
 int tid ;
 omp_set_num_threads(3);
 #pragma omp parallel private(tid)
 {
   tid = omp_get_thread_num();
   printf("Hello World from thread = %d \n", tid);
 }
 printf("Out of parallel region \n");
}
```

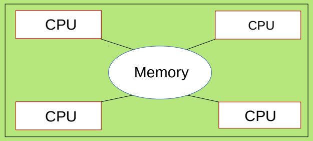
←*set number of threads to 3*

Parallel region within the brackets following
#pragma omp parallel ...

```
> gcc -fopenmp helloWorld.c
> ./a.out
Hello World from thread = 1
Hello World from thread = 2
Hello World from thread = 0
Out of parallel region
```

Order not deterministic

OpenMP "Hello world" (in Fortran)



F90 code

```
program main
  use omp_lib
  implicit none
  integer :: tid
  call omp_set_num_threads(3)
  !$omp parallel private (tid)
    tid = omp_get_thread_num()
    write(*,'(a,i2)') 'Hello World from thread ', tid
  !$omp end parallel
 write(*,'(a)') 'Out of parallel region'
end
```

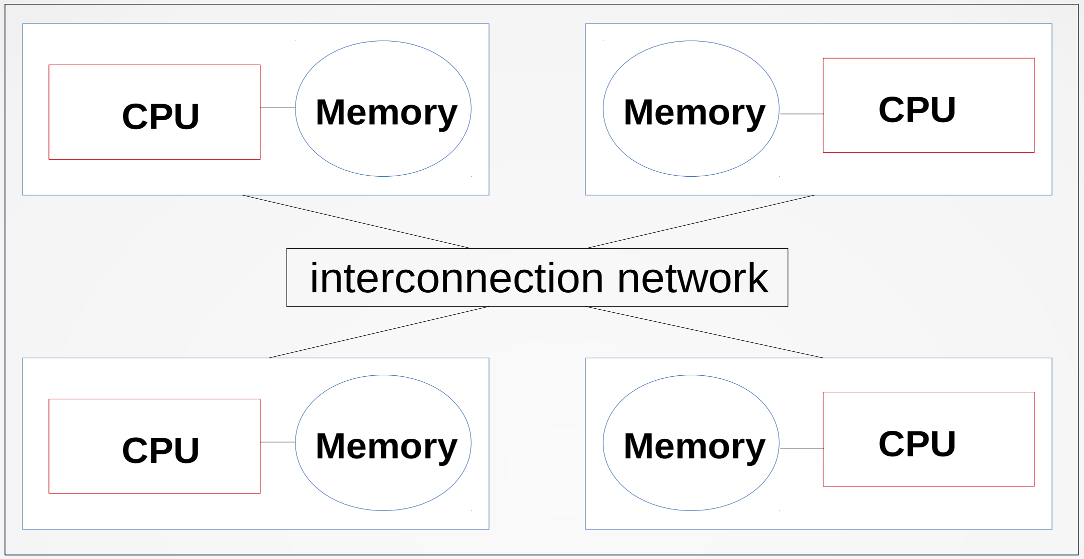
←*set number of threads to 3*

Parallel region between
!$omp parallel …
and
!$omp end parallel

```
> gfortran -fopenmp helloWorld.f90
> ./a.out
Hello World from thread 1
Hello World from thread 2
Hello World from thread 0
Out of parallel region
```

Order not deterministic

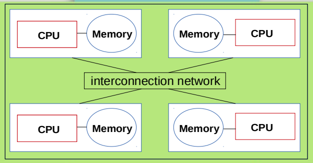
Message Passing (distributed memory)



Several processes act each on their own data and memory (own part of **distributed memory**).

Inter-process messages necessary for data exchange and synchronization.

Message Passing API's

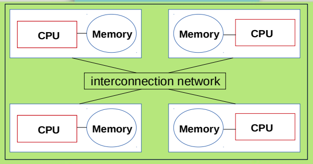


Application Program Interfaces (API's) enabling message passing:

- **MPI (Message Passing Interface)**
- Linda
- PVM (Parallel Virtual Machine)

Note: Can also be used on shared-memory systems.

MPI "Hello world" (in C)



C code

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[ ]) {
   int  ntasks, rank;
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&ntasks);
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
   printf ("Hello from rank %d out of %d tasks \n", rank, ntasks);
   MPI_Finalize();
}
```

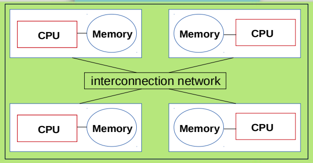
MPI statements between MPI_init and MPI_Finalize

```
> mpicc helloWorld.c
> mpirun -np 3 ./a.out
Hello from rank 2 out of 3 tasks
Hello from rank 0 out of 3 tasks
Hello from rank 1 out of 3 tasks
```

←*set number of tasks to 3*

Order not deterministic

MPI Send/Receive (in C)



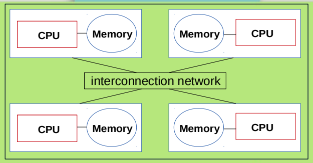
MPI_Send (&send_msg, count, datatype, **destination**, tag, comm)

↓

C code

MPI_Recv (&recv_msg, count, datatype, **source**, tag, comm, &status)

MPI "Hello world" (in Fortran)



F90 code

```
program main
  use mpi
  implicit none
  Integer :: ntasks, rank, ierr
  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, ntasks, ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
  write(*,'(a,i2,a,I2,a)') 'Hello from rank ', rank, ' out of ',ntasks,' tasks.'
  call MPI_FINALIZE(ierr)
end
```

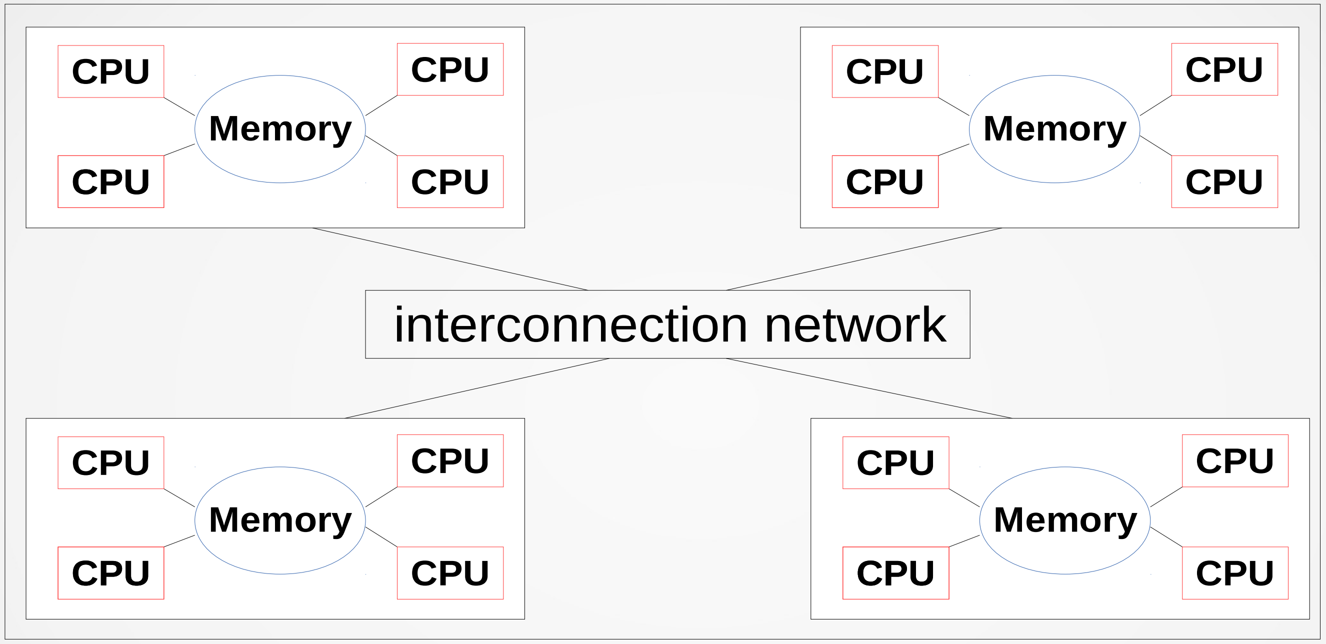
MPI statements between MPI_init and MPI_Finalize

```
> mpif90 helloWorld.f90
> mpirun -np 3 ./a.out
Hello from rank 2 out of 3 tasks.
Hello from rank 0 out of 3 tasks.
Hello from rank 1 out of 3 tasks.
```

←*set number of tasks to 3*

Order not deterministic

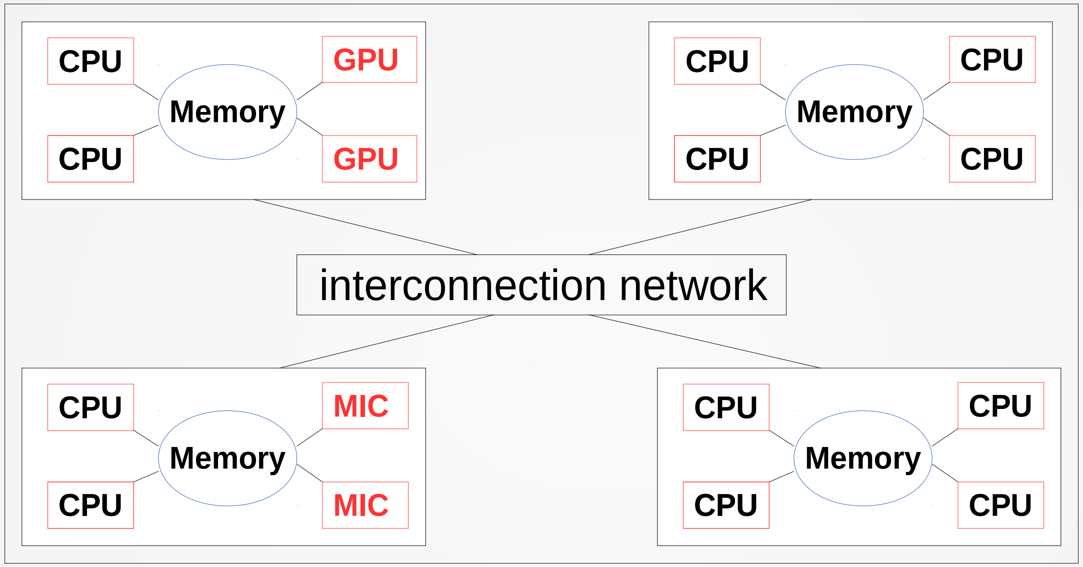
Mixed Programming



Message passing betwen nodes + Multithreading within nodes

Typically MPI + OpenMP

Hybrid Programming with accelerators



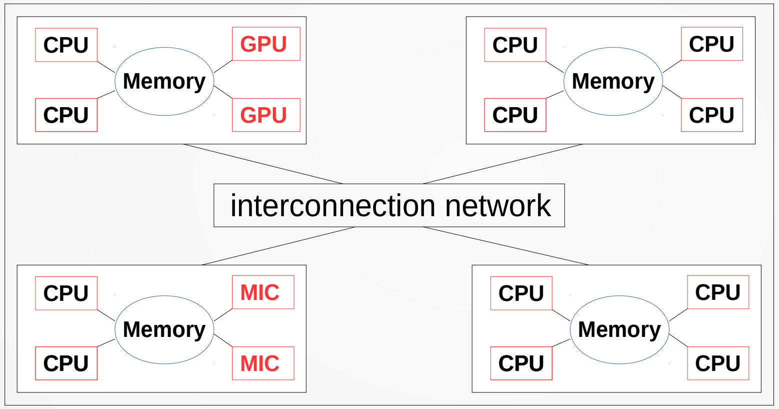
Idea: offload computationally-intensive tasks from CPU to GPU/MIC accelerators.

Accelerator programming

- **CUDA** (Compute Unified Device Architecture) for GPU (proprietary to Nvidia)
- **OpenCL** (Open Computing Language) for CPU, MIC and GPU (cross-platform)
- **OpenACC** for CPU, MIC and GPU (cross-platform) - can be seen as an "extension of openMP to accelerators"

Note on "HPC" terminology

The terminology “**High Performance Computing**” (**HPC**) usually implies the use of **fast interconnection** between processors.



If the sub-problems are (quasi-)independent, a fast interconnexion between processors is not necessary. For such "**embarrassingly parallel**" problems, **grid computing** (= loosely coupled CPUs) is more appropriate.

Layout

- Why parallel computing?
- Parallel architectures
- Parallel programming models
- Parallel programming tools
- Parallel efficiency
- Computer performance measurements
- Existing HPC infrastructures & MeCS

Parallel programming tools

To parallelize an existing code is long and complex, but is the only way to take advantage of massive parallelism.

It can yield substantial improvements to your simulation capabilities.

To parallelize an existing code or to build a new one, there exist open-source parallel libraries implementing “fundamental” tasks.
Ex.: linear/non-linear algebra solvers, finite element packages...

Parallel programming tools

One should be aware of existing parallel libraries in one's field.

These libraries can “help you concentrate on your science”.

They are based on message passing and/or multi-threading. Thus it is important to know about the “basic” standards OpenMP and MPI.

Parallel tools in linear algebra

Ex: PETSc, Trilinos, MUMPS, Pastix, Paralution,
FreeFEM++, FEniCS, ...

More complete list on:

http://www.netlib.org/utk/people/JackDongarra/la-sw.html

More on PETSc

- *Portable, Extensible Toolkit for Scientific Computation*
- Open-source set of tools for parallel solution of PDEs
- Specialized in large sparse iterative solvers
  (+ interfaces with direct solvers)
- Emphasis on scalability
- Interface for C/C++, Fortran, Python
- Based on MPI distributed parallelism, but recent developments for shared-memory (pthreads) and GPU
- More on **http://www.mcs.anl.gov/petsc/**

PETSc typical code (1/2)

C code

```
PetscInitialize( &argc, &argv, 0, 0 );
...
VecCreateMPI ( MPI_Comm comm, int m, int M, Vec *x);
VecSetValues ( Vec x, int n, int *indices, PetscScalar *values, ...);
VecAssemblyBegin (Vec x);
VecAssemblyEnd (Vec x);
…
MatCreateAIJ(MPI_Comm comm, int m, int n, int M, int N,
                    int d_nz, int d_nnz[ ], int o_nz, int o_nnz[ ],Mat *A);
...
```

PETSc typical code (2/2)

C code

```
…
KSPCreate(MPI_Comm comm, KSP *ksp);
KSPSetOperators(KSP ksp, Mat A, …);
KSPSetType(KSP ksp, KSPType kspType);
…
KSPgetPC(KSP ksp,PC *pc);
PCSetType(PC pc, PCType pcType);
...
KSPSolve(KSP ksp,Vec b,Vec x);
...
PetscFinalize();
```

where **kspType** = KSPCG, KSPGMRES, ...

+ KSPPREONLY

where **pcType** = PCJACOBI, PCSOR, PCILU, PCASM,...

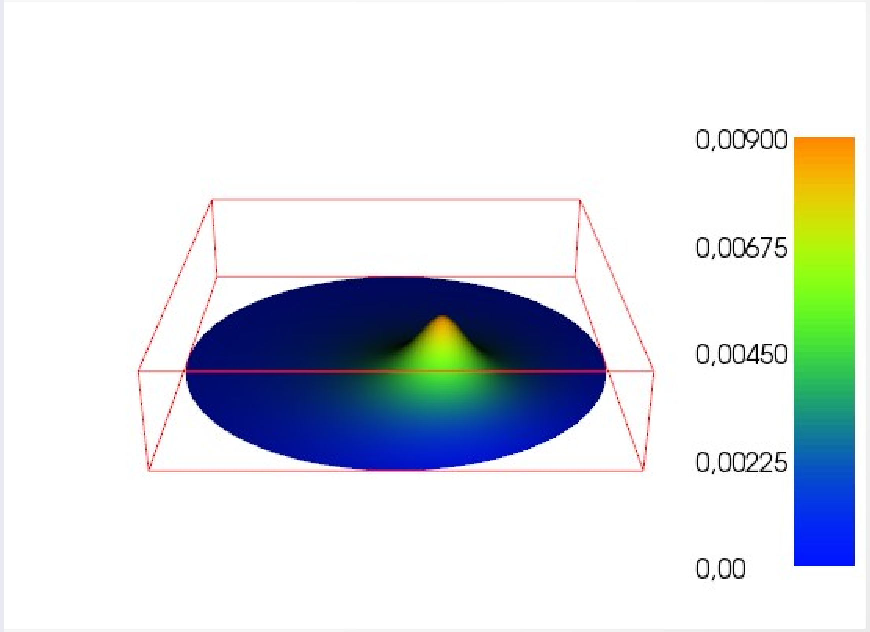
More on FEniCS

- "Collection of free software for automated, efficient solution of differential equations"
- Primarily designed for Finite Element applications
- Includes meshing and post-processing
- Interface for C++ and Python
- Various linear algebra backends: PETSc, Trilinos, ...
- More on **http://fenicsproject.org/**

FeniCS exemple

```
from dolfin import *
import numpy

x0 = 0.2
y0 = -0.5
sigma = 0.06
meshCellSize = 1./50.
mesh = CircleMesh(Point(0, 0), 1.0, meshCellSize)
V = FunctionSpace(mesh, 'Lagrange', 1)

# Define boundary conditions
def boundary(x, on_boundary):
    return on_boundary
bc = DirichletBC(V, Constant(0.0), boundary)

# Define variational problem
u = TrialFunction(V)
v = TestFunction(V)
a = inner(nabla_grad(u), nabla_grad(v))*dx
f = Expression('exp(-0.5*(pow((x[0] - x0)/sigma, 2)) '
               '    -0.5*(pow((x[1] - y0)/sigma, 2)))',
               x0=x0, y0=y0, sigma=sigma)
L = f*v*dx

# Compute solution
u = Function(V)
problem = LinearVariationalProblem(a, L, u, bc)
solver  = LinearVariationalSolver(problem)
solver.parameters['linear_solver'] = 'cg'
solver.parameters['preconditioner'] = 'ilu'
solver.solve()

# Plot solution and mesh
plot(u, title='u')
plot(mesh, title='mesh')
interactive()
```

$$\Delta u = f$$



Layout

- Why parallel computing?
- Parallel architectures
- Parallel programming models
- Parallel programming tools
- Parallel efficiency
- Computer performance measurements
- Existing HPC infrastructures & MeCS

Parallel efficiency

Parallelization is efficient
as long as
the computing time necessary to reach the solution
diminishes when the number of processors grows.

Parallel efficiency

Quality of parallelism is measured through **Speed-Up** and **Efficiency**.

Let :

- T_1 = time of serial execution
- T_N = time of parallel execution (on N processors)

Then :

Speed-up $S_N = T_1 / T_N$ **Efficiency $E_N = S_N / N$**

If perfect parallelism: $S_N = N$ and $E_N = 1$

Scalability (*Extensibilité*)

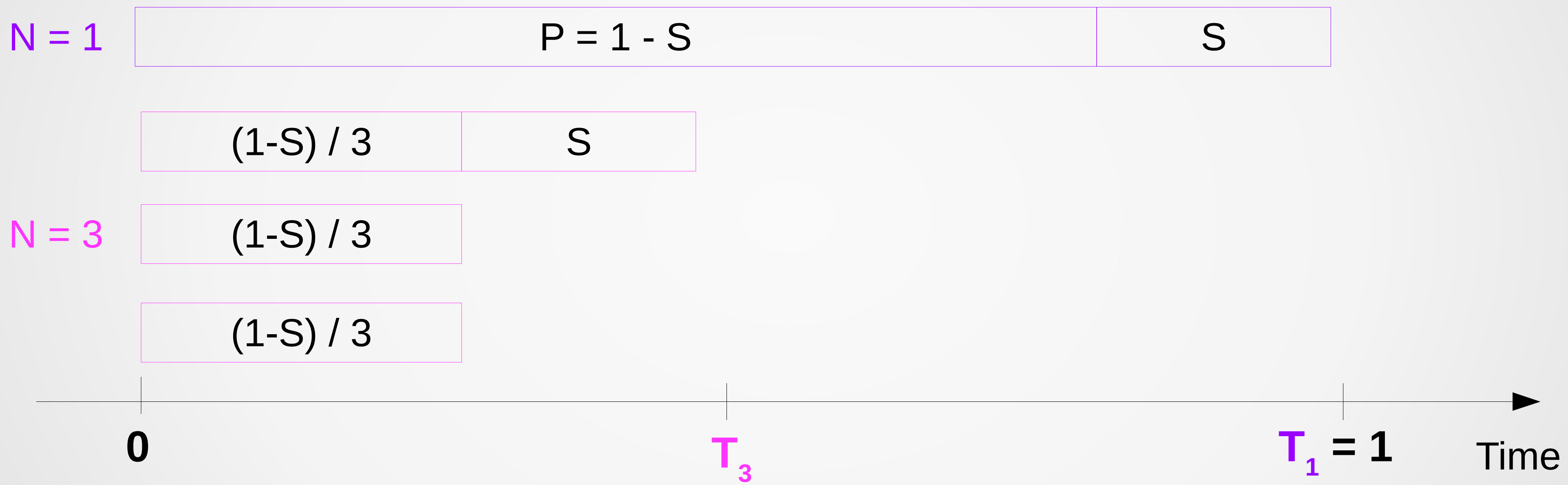
A parallel program "**scales**" (« *passe à l'échelle* ») if it demonstrates a proportionate increase in speedup when adding more resources.

Scalability limited by:

- **Communication overhead**
- **Load balancing**: the slowest task determines overall performance → work should be distributed to minimize task idle time.
- Fraction of work that can not be parallelized: **Amdahl's law**

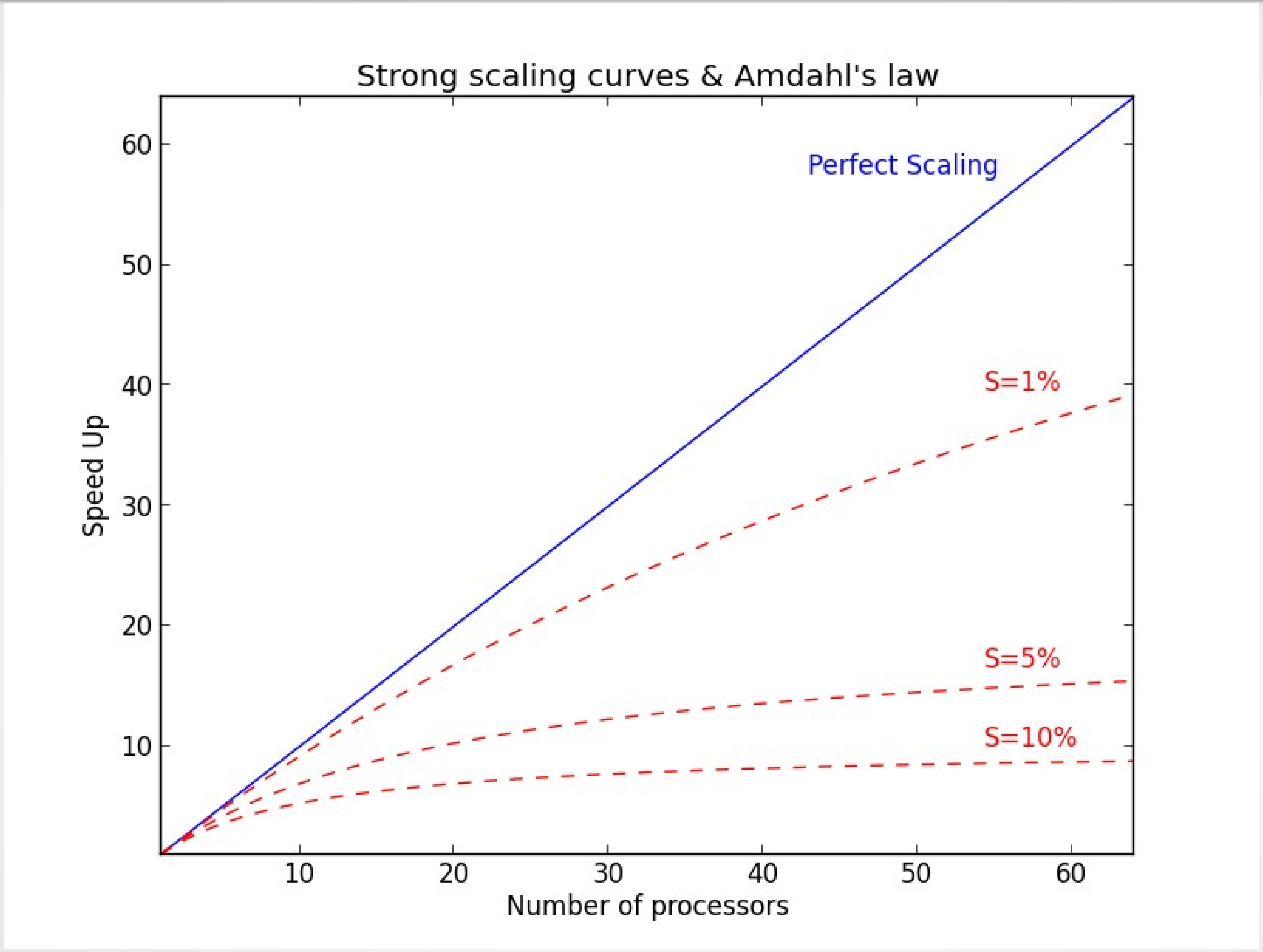
Amdahl's Law

- S = serial fraction of the code
- P = 1 – S = parallel fraction of the code



$$\textbf{Speed-up} = \frac{T_1}{T_N} = \frac{1}{(1-S)/N + S} \xrightarrow[N \to \infty]{} \frac{1}{S} \neq N$$

Amdahl's Law



After a certain point, using more processors yields no acceleration …
unless the problem size is enlarged.

Note: Strong vs. Weak scaling (1/2)

Amdahl's law assumes a **fixed problem size**
- this corresponds to **strong scaling**.

One can also compute the speed-up assuming
a **fixed problem size** **per processor** (thus
enlarging the problem size when adding processors).
- this corresponds to **weak scaling.**

Note: Strong vs. Weak scaling (2/2)

A weak scaling curve is a plot of $\frac{T_1^P}{T_N^{N*P}}$

where T_1^P = time to solve problem P on 1 processor

T_N^{N*P} = time to solve problem N*P (i.e. problem P **enlarged** by a factor N) on N processors.

→ an ideal weak scaling curve is flat.

Layout

- Why parallel computing?
- Parallel architectures
- Parallel programming models
- Parallel programming tools
- Parallel efficiency
- Computer performance measurements
- Existing HPC infrastructures & MeCS

Performance measurement

Unit: **FLOPS =** FLoating point Operations **Per Second**

(opérations en virgule flottante)

$\downarrow$

= Additions and multiplications on real numbers

Performance measurement

Peak performance rate (*Puissance de crête*, in FLOPS):

$$\mathbf{R_{PEAK} = N_{PROC} * frequency * N_{FLOP}/cycle}$$

where :

$\mathbf{N_{PROC}}$ = number of processors

frequency = frequency clock rate of the processors

$\mathbf{N_{FLOP}/cycle}$ = number of floating point operations per clock cycle

Performance measurment

R_{PEAK} is theoretical.

In practice: **maximum performance rate = R_{MAX}**

$$\mathbf{R_{MAX} < R_{PEAK}}$$

Performance measurment

$\mathbf{R_{MAX}}$ depends on

- machine load
- I/O operations and quality of file system
- problem at hand (number of memory accesses required, …)
- …

→ $\mathbf{R_{MAX}}$ measured for given **benchmarks**

Performance measurment

The **Linpack benchmark** consists in solving a dense system of linear equations Ax = b.

This benchmark is currently used to rank the fastest computers in the world
→ **Top 500 ranking** (updated every 6 months)

Layout

- Why parallel computing?
- Parallel architectures
- Parallel programming models
- Parallel programming tools
- Parallel efficiency
- Computer performance measurements
- Existing HPC infrastructures & MeCS

Tianhe-2 (China)

http://phys.org

#1 top 500 (June 2014)

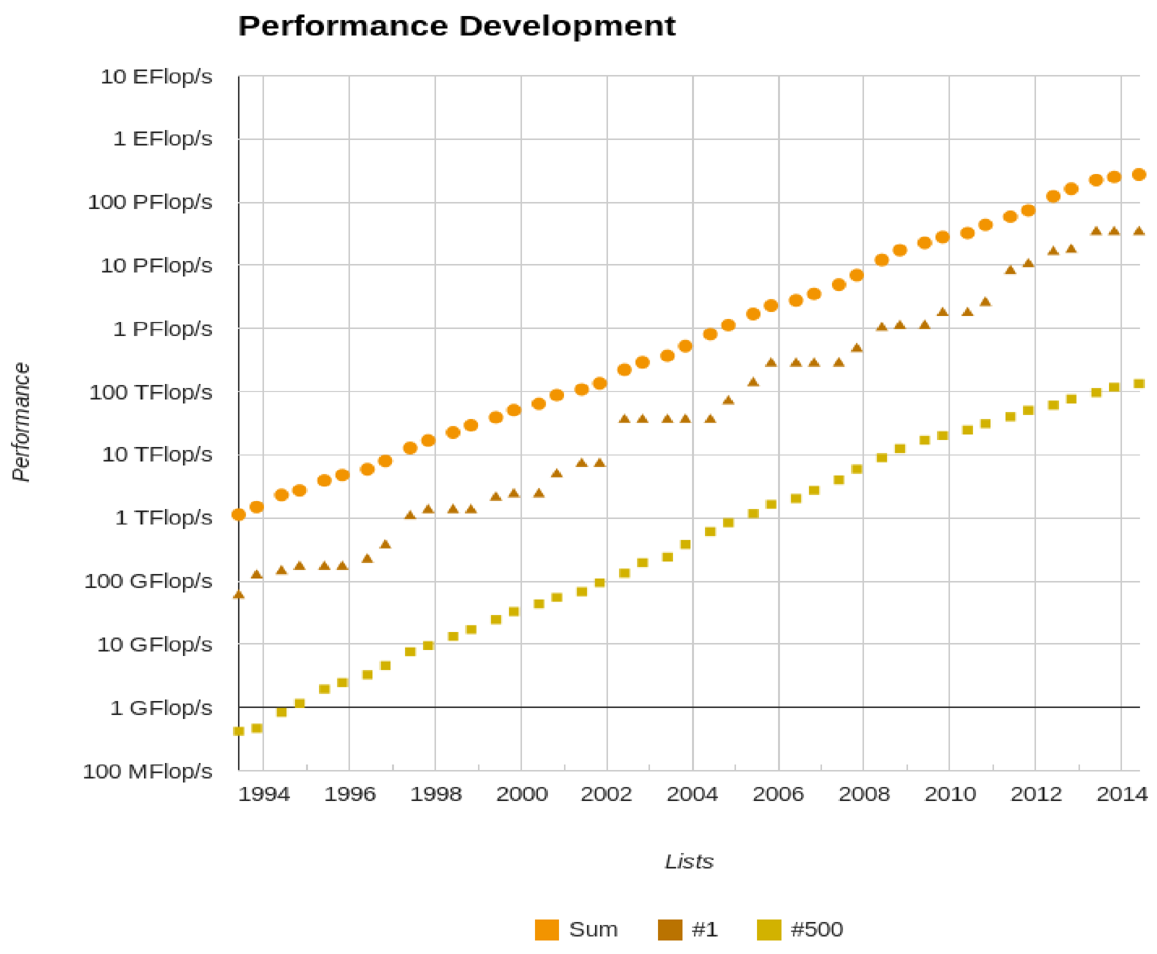
Top 500 (June 2014)

Rank	Country	Name	# cores	R_{MAX} (PFlops)	R_{PEAK} (PFlops)	Power (MW)
1	🇨🇳	Tianhe-2	3 120 000 (CPU + MIC)	33,8	54,9	17,8
2	🇺🇸	Titan	560 640 (CPU + GPU)	17,6	27,1	8,2
3	🇺🇸	Sequoia	1 572 864 (CPU)	17,2	20,1	7,9

NB: PFlops = Petaflops = 10^{15} Flops

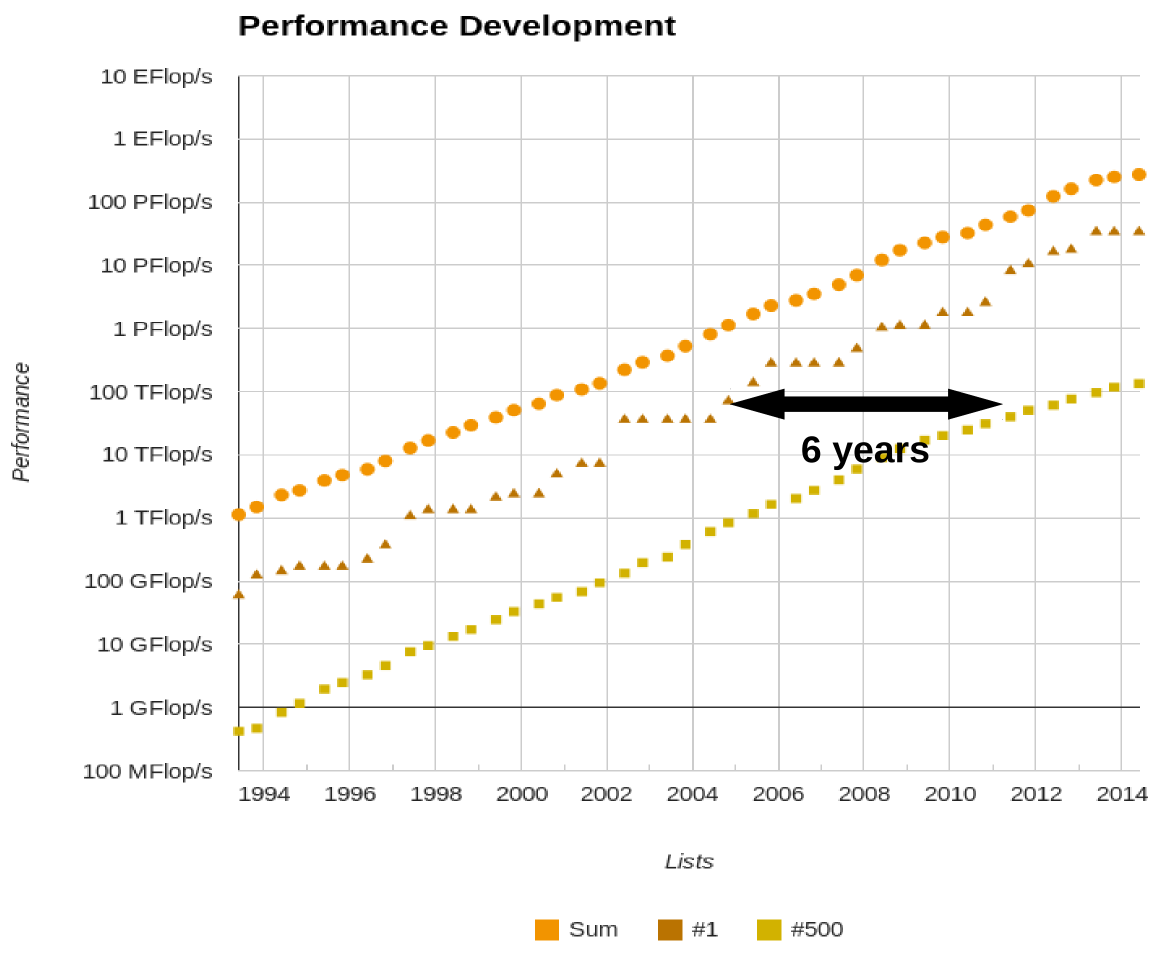
Top 500: evolution

from top500.org



Top 500: evolution

from top500.org



Top 500 (June 2014) - France

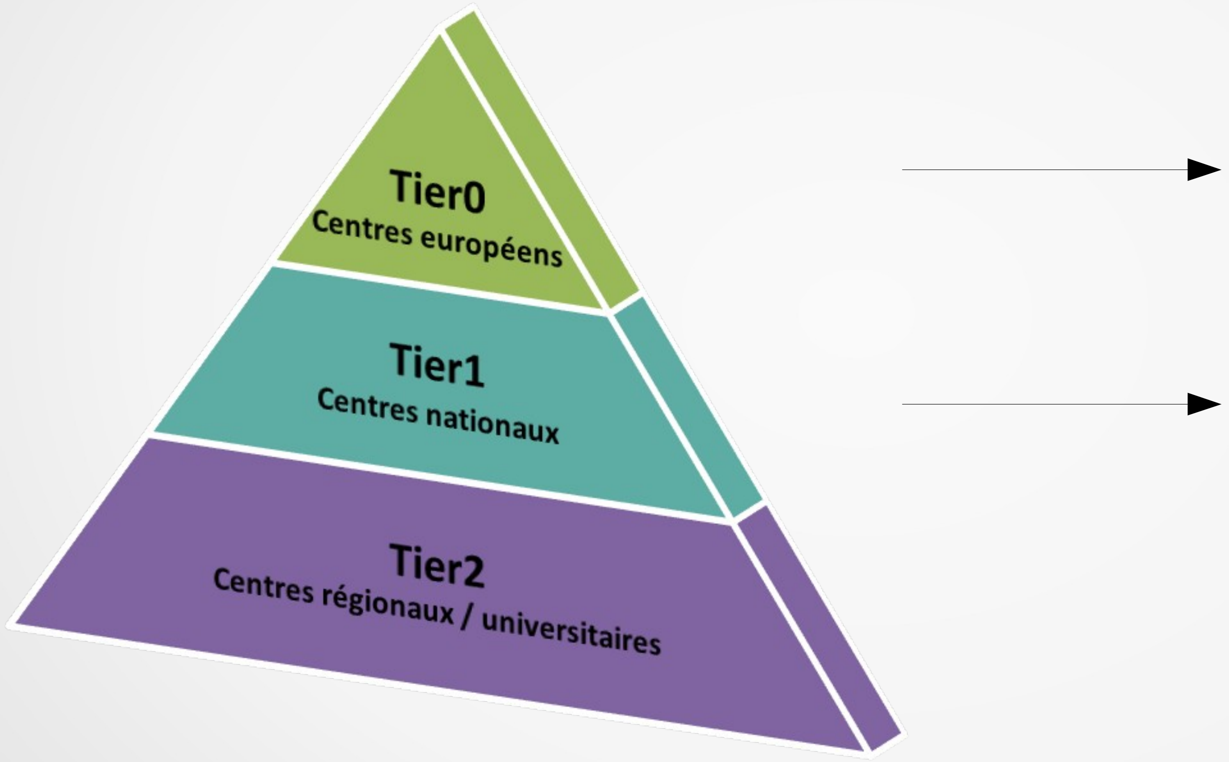
Rank	Institution	Name	# cores	R_{MAX} (PFlops)	R_{PEAK} (PFlops)	Power (MW)
16	Total	Pangea	110 400 (CPU)	2,1	2,3	2,1
26	CEA/ TGCC	Curie	77 184 (CPU)	1,4	1,7	2,3

NB: PFlops = Petaflops = 10^{15} Flops

Curie (CEA / TGCC)

http://www.genci.fr

HPC in France (and Europe)



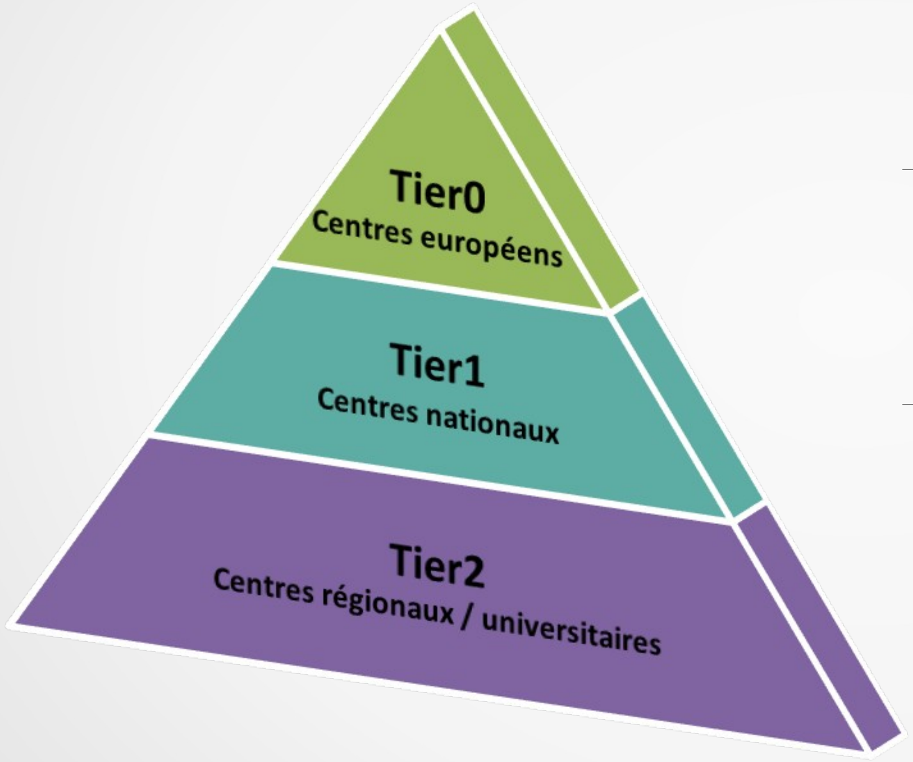
http://www.genci.fr

HPC national research centers in France

***GENCI** – Grand Equipement National de Calcul Intensif:*

- **CEA / TGCC** - Bruyères-le-Chatel *(Très Grand Centre de Calcul)*
- **IDRIS** - Orsay *(Institut du Développement et des Ressources en Informatique Scientifique*)
- **CINES** - Montpellier (*Centre Informatique National de l'Enseignement Supérieur*)

HPC in France (and Europe)



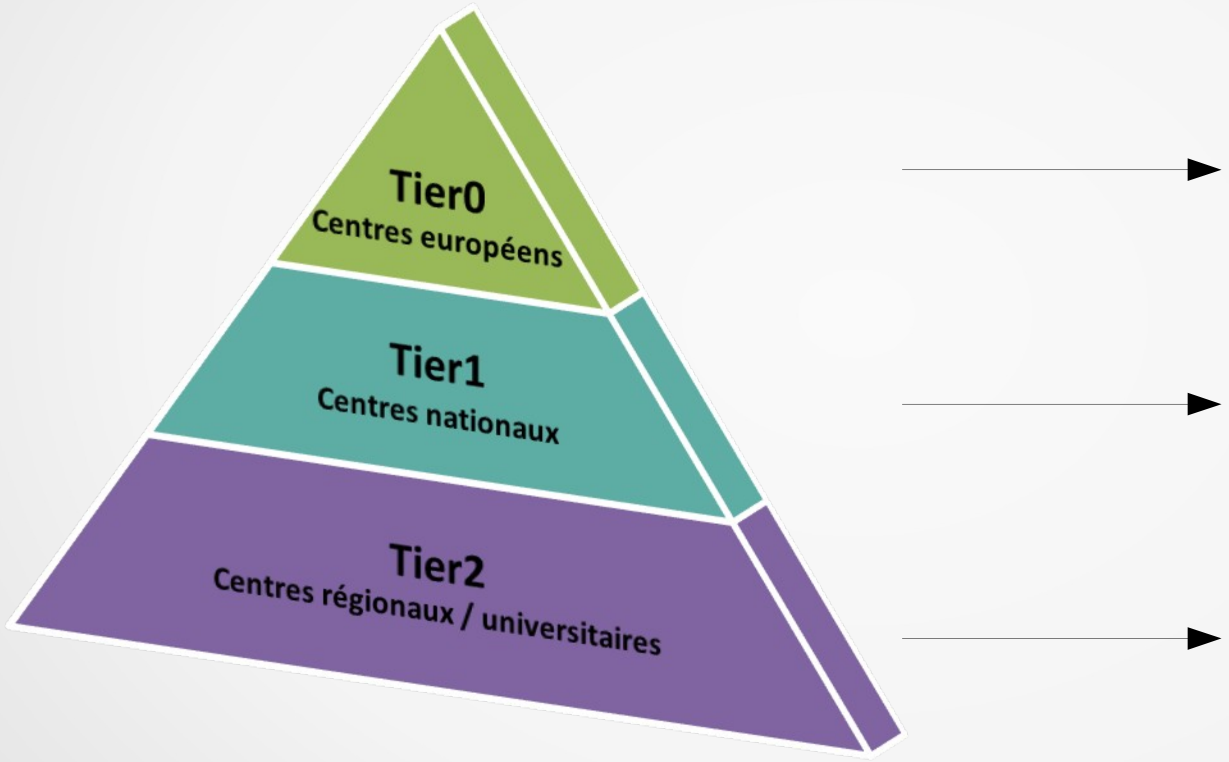
http://www.genci.fr

PRACE

GENCI

Regular calls for computing time allocation

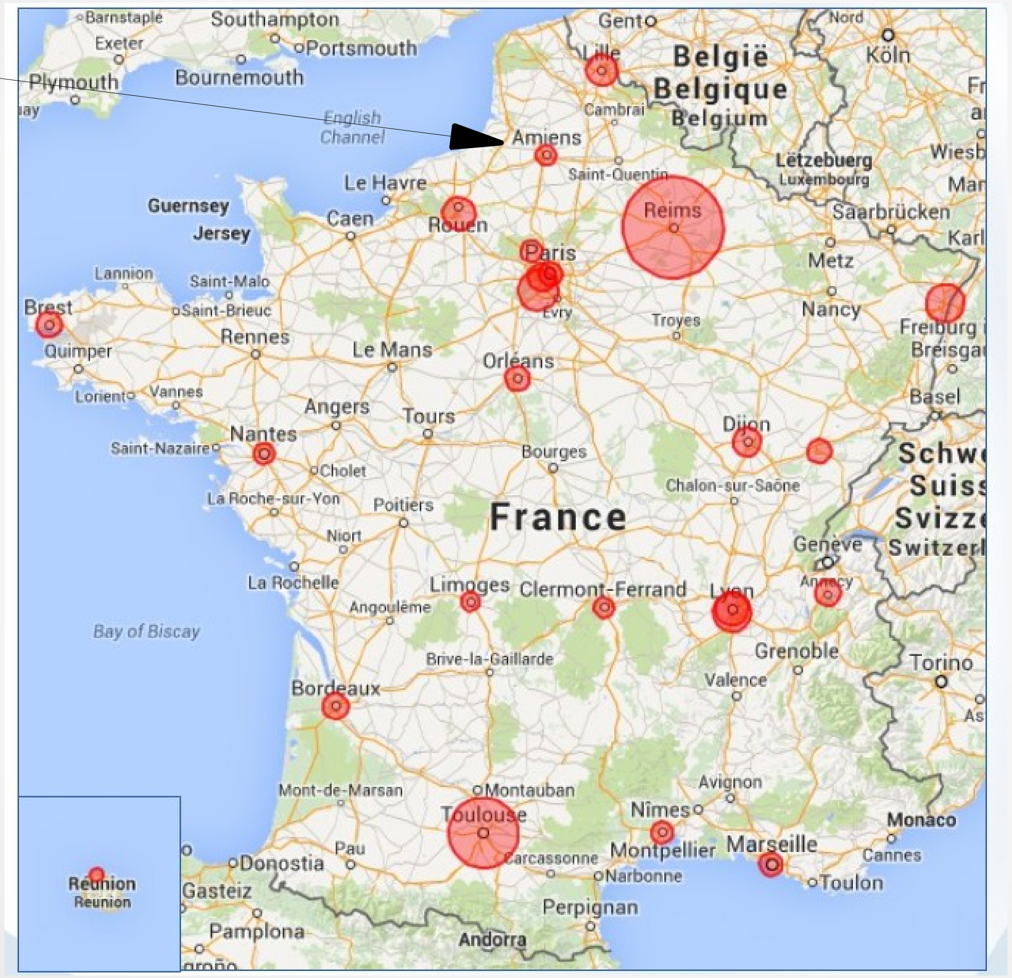
HPC in France (and Europe)



« Méso-centres »

http://www.genci.fr

« Méso-centres » : peak powers



Courtesy of Loïc Gouarin

The MeCS platform

- *<u>M</u>odélisation <u>e</u>t <u>C</u>alcul <u>S</u>cientifique*
- Dedicated to numerical simulations using HPC
- Mutualized equipment within UPJV
- Open to researchers from all fields

Terralia Building

http://www.amiens-amenagement.fr

The MeCS parallel computer

SGI uv100 computer:

- 288 CPU cores
- R_{peak}: 3 Tflops (Teraflops = 10^{12} flops)
- Shared memory 1.2 TB ("SMP")

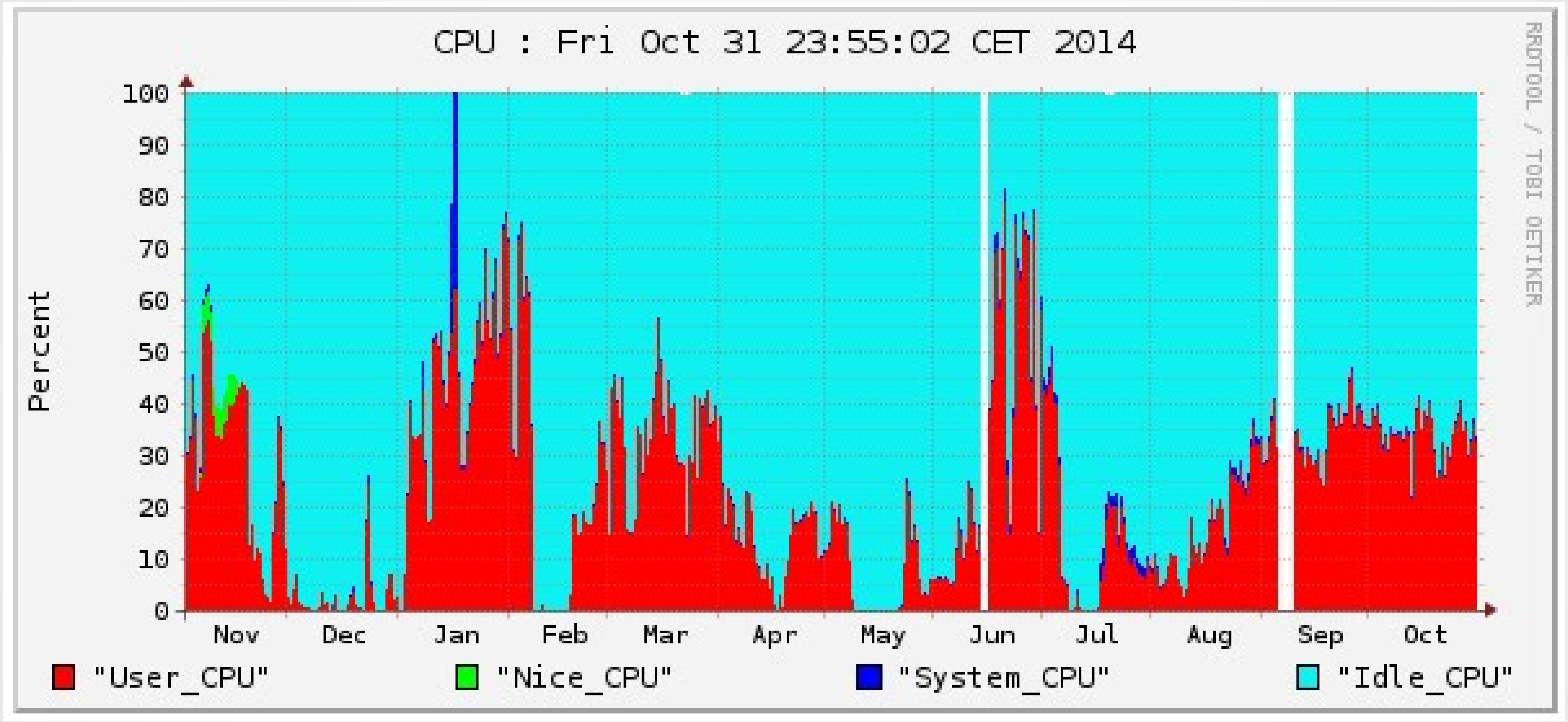
Public funding : *Contrat de Plan Etat-Région* (CPER 2010-2011)

The MeCS parallel computer

Storage server 98TB

Public funding : *Contrat de Plan Etat-Région* (CPER 2013)

MeCS CPU use (Nov. 2013 to Oct. 14)



MeCS CPU use, from **November 1st, 2013**, to **October 31st, 2014**

MeCS top users (Nov. 2013 to Oct. 14)

	Username	# jobs	CPU Time[s]	Labo
1	**rstancu**	1049	**5,7E+8**	**LAMFA**
2	**bbouvier**	276	**1,9E+8**	**LG2A**
3	**chuminli**	1553	**9,4E+7**	**MIS**
4	**wleclerc**	222	**5,7E+7**	**LTI**
5	**nferguen**	90	**5,1E+7**	**LTI**
6	**ymammeri**	159	**3,2E+7**	**LAMFA**
7	**vreal**	297	**2,9E+7**	**LAMFA**
8	**vmartin**	618	**2,0E+7**	**LAMFA**
9	**MAXSAT**	456	**1,6E+7**	**MIS**
10	**mquiroga**	956	**6,1E+6**	**LRCS**

1 year = 3,2E+07s

*** 272 = 8,58E+09s**

Total used :
1.1E09s

MeCS top users (30/10/2013 -14)

	Username	# jobs	CPU Time[s]	Labo
11	**ccezard**	36	**4,6E+6**	**LG2A**
12	**gfranz**	55	**1,0E+6**	**LTI**
13	**kxue**	1220	**1,4E+6**	**LRCS**
14	**pgarnier**	25	**6,8E+5**	**LAMFA**
15	**abezard**	65	**5,3E+5**	**MIS**
16	**pblondel**	29	**1,4E+5**	**MIS**
17	**lemahec**	49	**2,4E+4**	**MIS**
18	**cmachado**	15	**2,1E+4**	**LTI**
19	**clecat**	1351	**7,4E+3**	**MIS**
20	**mdiallo**	6	**2,4E+3**	**LAMFA**

1 year = 3,2E+07s

*** 272 = 8,58E+09s**

Total used : 1.1E09s

MeCS user count

- Active accounts : 30
- "Activable" accounts : 20
- Growth : 10 accounts opened since mid-april 2014

MeCS training activities

- Ecole doctorale (2014,16 ?) : MPI/OpenMP (20h.)
- Ecole doctorale 2015 ? : Parallel programming Tools (PETSc, FEniCS)
- Seminars :
  - Introduction to parallel computing (13/11/2014)
  - OpenMP (04/12/2014)
  - MPI (TBA)

MeCS Contact

Technical questions: ***mecs@u-picardie.fr***

- Laurent Renault (LAMFA & MeCS)
- Fabien Berini (stagiaire MeCS – Master 2 ISRI)
- Serge Van Criekingen

MeCS Website

www.mecs.u-picardie.fr

www.mecs.u-picardie.fr

S'identifier

Plateforme MeCS
Plateforme Modélisation et Calcul Scientifique

Derniers changements Gestionnaire de médias Plan du site

Vous êtes ici: accueil

Accueil:
- contact
- formation
- logiciels
- ouverture_de_compte
- projets
- ressources
- seminaire

Partie Privée:

Livret d'Exploitation:

Boite à outils:

Créer une nouvelle page:

Vous n'avez pas les droits pour ajouter une page

accueil

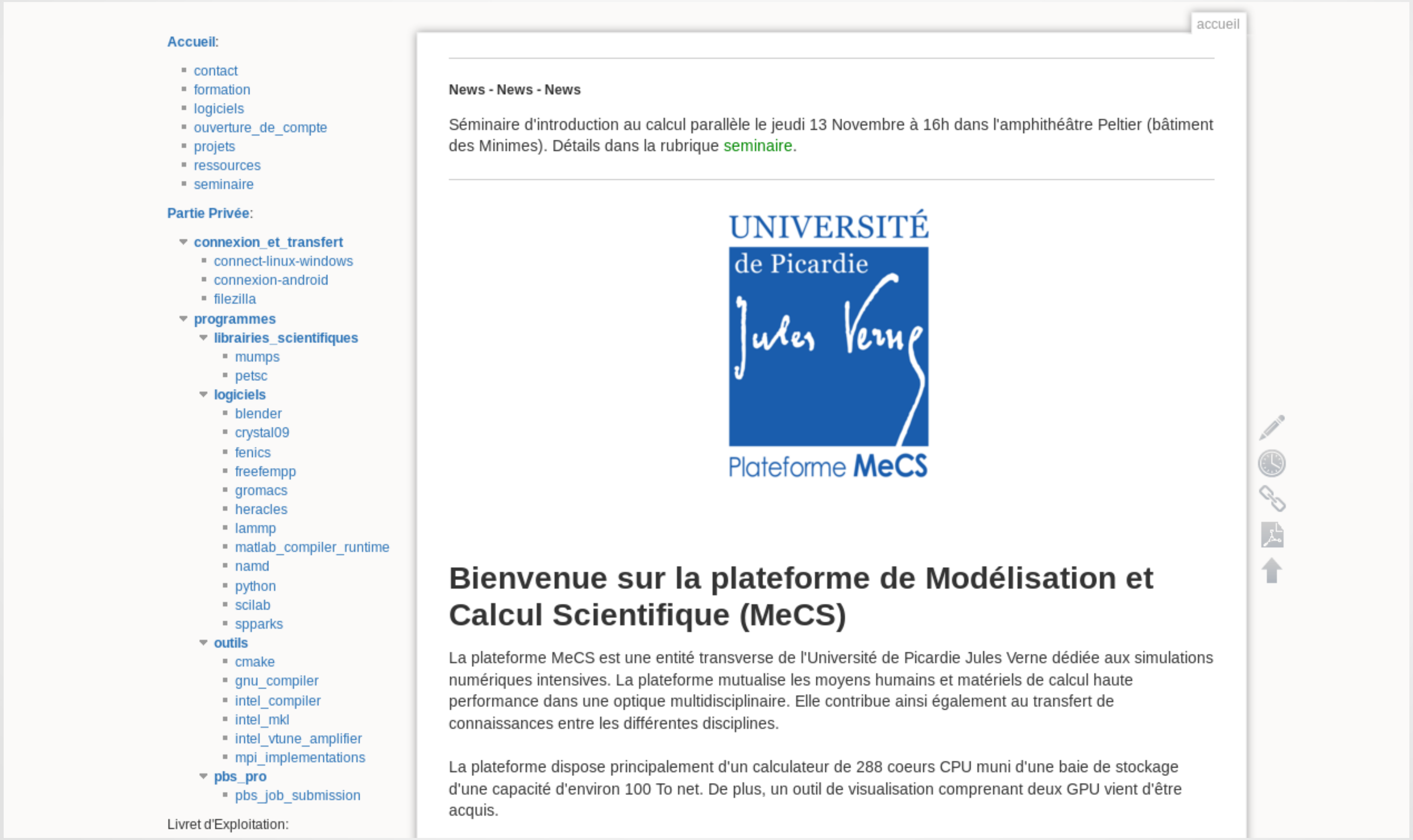
News - News - News

Séminaire d'introduction au calcul parallèle le jeudi 13 Novembre à 16h dans l'amphithéâtre Peltier (bâtiment des Minimes). Détails dans la rubrique seminaire.

Bienvenue sur la plateforme de Modélisation et Calcul Scientifique (MeCS)

La plateforme MeCS est une entité transverse de l'Université de Picardie Jules Verne dédiée aux simulations numériques intensives. La plateforme mutualise les moyens humains et matériels de calcul haute performance dans une optique multidisciplinaire. Elle contribue ainsi également au transfert de connaissances entre les différentes disciplines.

www.mecs.u-picardie.fr



Accueil:

- contact
- formation
- logiciels
- ouverture_de_compte
- projets
- ressources
- seminaire

Partie Privée:

- **connexion_et_transfert**
  - connect-linux-windows
  - connexion-android
  - filezilla
- **programmes**
  - **librairies_scientifiques**
    - mumps
    - petsc
  - **logiciels**
    - blender
    - crystal09
    - fenics
    - freefempp
    - gromacs
    - heracles
    - lammp
    - matlab_compiler_runtime
    - namd
    - python
    - scilab
    - spparks
  - **outils**
    - cmake
    - gnu_compiler
    - intel_compiler
    - intel_mkl
    - intel_vtune_amplifier
    - mpi_implementations
  - **pbs_pro**
    - pbs_job_submission

Livret d'Exploitation:

accueil

News - News - News

Séminaire d'introduction au calcul parallèle le jeudi 13 Novembre à 16h dans l'amphithéâtre Peltier (bâtiment des Minimes). Détails dans la rubrique seminaire.

UNIVERSITÉ de Picardie Jules Verne

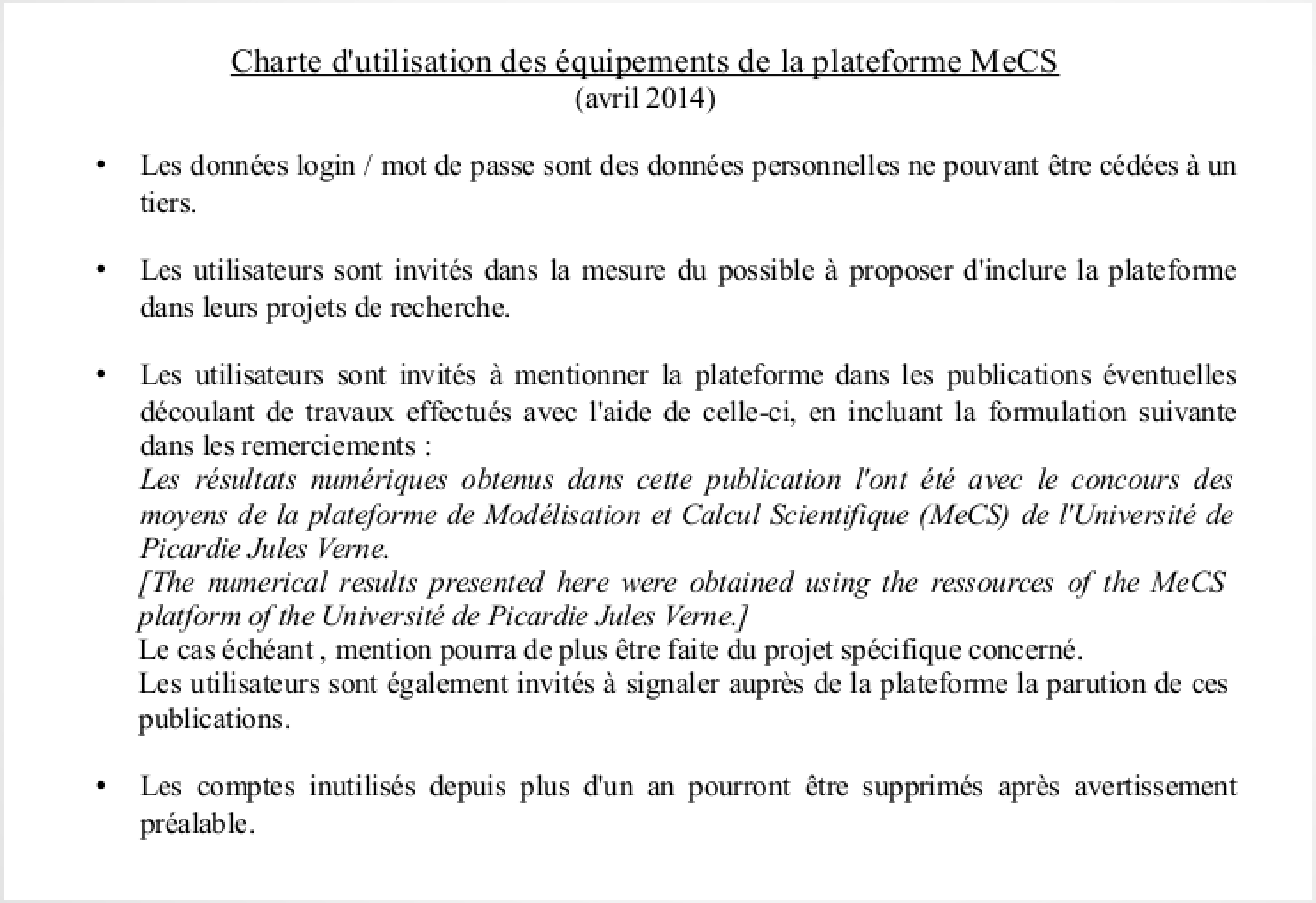
Plateforme MeCS

Bienvenue sur la plateforme de Modélisation et Calcul Scientifique (MeCS)

La plateforme MeCS est une entité transverse de l'Université de Picardie Jules Verne dédiée aux simulations numériques intensives. La plateforme mutualise les moyens humains et matériels de calcul haute performance dans une optique multidisciplinaire. Elle contribue ainsi également au transfert de connaissances entre les différentes disciplines.

La plateforme dispose principalement d'un calculateur de 288 coeurs CPU muni d'une baie de stockage d'une capacité d'environ 100 To net. De plus, un outil de visualisation comprenant deux GPU vient d'être acquis.

Charte



Charte d'utilisation des équipements de la plateforme MeCS
(avril 2014)

- Les données login / mot de passe sont des données personnelles ne pouvant être cédées à un tiers.
- Les utilisateurs sont invités dans la mesure du possible à proposer d'inclure la plateforme dans leurs projets de recherche.
- Les utilisateurs sont invités à mentionner la plateforme dans les publications éventuelles découlant de travaux effectués avec l'aide de celle-ci, en incluant la formulation suivante dans les remerciements :
  Les résultats numériques obtenus dans cette publication l'ont été avec le concours des moyens de la plateforme de Modélisation et Calcul Scientifique (MeCS) de l'Université de Picardie Jules Verne.
  [The numerical results presented here were obtained using the ressources of the MeCS platform of the Université de Picardie Jules Verne.]
  Le cas échéant , mention pourra de plus être faite du projet spécifique concerné.
  Les utilisateurs sont également invités à signaler auprès de la plateforme la parution de ces publications.
- Les comptes inutilisés depuis plus d'un an pourront être supprimés après avertissement préalable.

UNIVERSITÉ de Picardie Jules Verne

Plateforme MeCS

www.mecs.u-picardie.fr

Current MeCS PBS rules

```
Queue name   ncpus (max/min)  walltime (max/default)   mem (max/default)  max_jobs  priority
--------------------------------------------------------------------------------------------
paral1          65 / 128          2h / 30 min.         256 Gb / 16Gb        3        100
paral2          17 / 64          24h / 1h.             128 Gb /  8Gb        6         90
paral3           2 / 16          48h / 1h.              32 Gb /  4Gb        6         80
paral4           2 / 16         240h / 1h.              32 Gb /  4Gb       3+2        70
serial           1 / 1          240h / 1h.              10 Gb /  1Gb      12+8        60
```