Modèles de programmation

Ecole informatique

Installation et exploitation d'un cluster de calcul

Guy Moebs

Laboratoire de Mathématiques Jean Leray, CNRS, Université de Nantes

Janvier 2016

Plan de la présentation

Introduction

Parallélisme en mémoire distribuée : MPI

Parallélisme en mémoire partagée : OpenMP

Evolution

TP

Introduction
- But de la parallélisation
- Optimisation de la parallélisation
- Accélération et efficacité
- Interconnexion
- Architecture parallèles
- Modèles de programmation

Parallélisme en mémoire distribuée : MPI

Parallélisme en mémoire partagée : OpenMP

Evolution

TP

Pourquoi paralléliser ?

- La première question à se poser, c'est savoir si la parallélisation de l'application est nécessaire
- Ecrire un programme séquentiel est déjà du travail, souvent difficile ; la parallélisation le rendra encore plus dur
- Il y a eu beaucoup de progrès technologique dans le matériel informatique (processeurs à 3 GHz, bus mémoire, ...) mais il ne faut plus compter sur des avancées au même rythme
- Pourtant il existe toujours des applications scientifiques qui consomment "trop" de ressources en temps de processeur ou en mémoire
- Pour celles-ci, la seule solution, pour des raisons techniques ou économiques, reste la parallélisation

But de la parallélisation

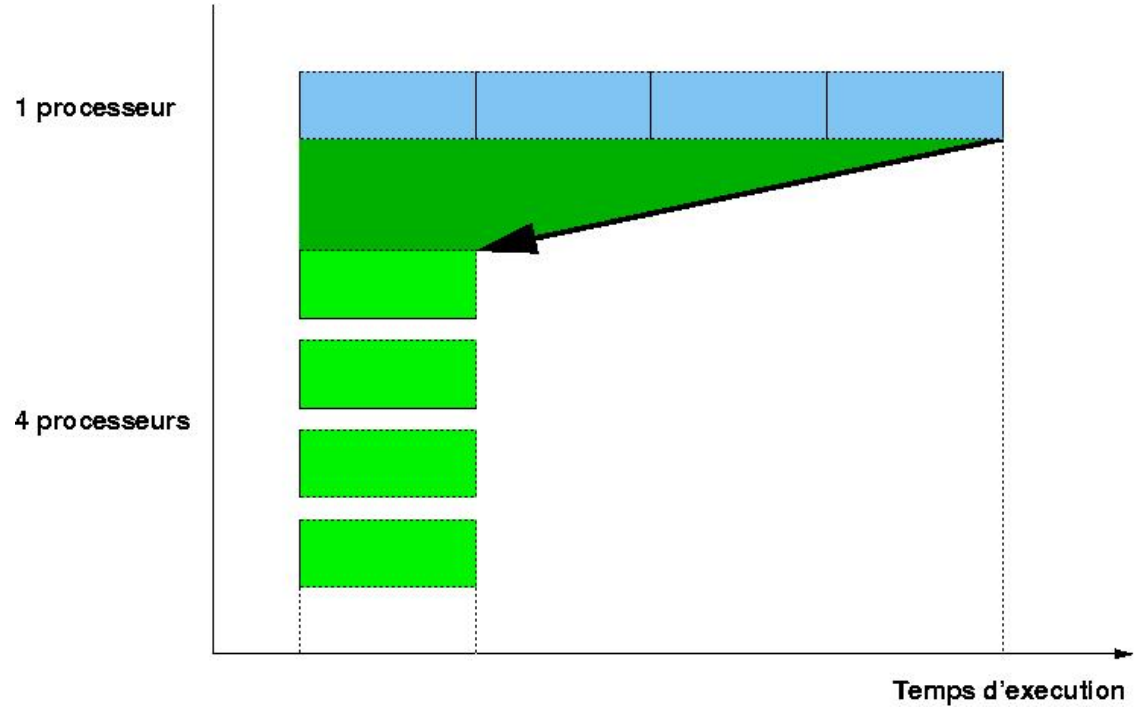
Qu'est-ce que la parallélisation ?

Ensemble de techniques logicielles et matérielles permettant l'exécution simultanée de séquences d'instructions indépendantes, sur des processeurs différents

Bénéfice de la parallélisation ?

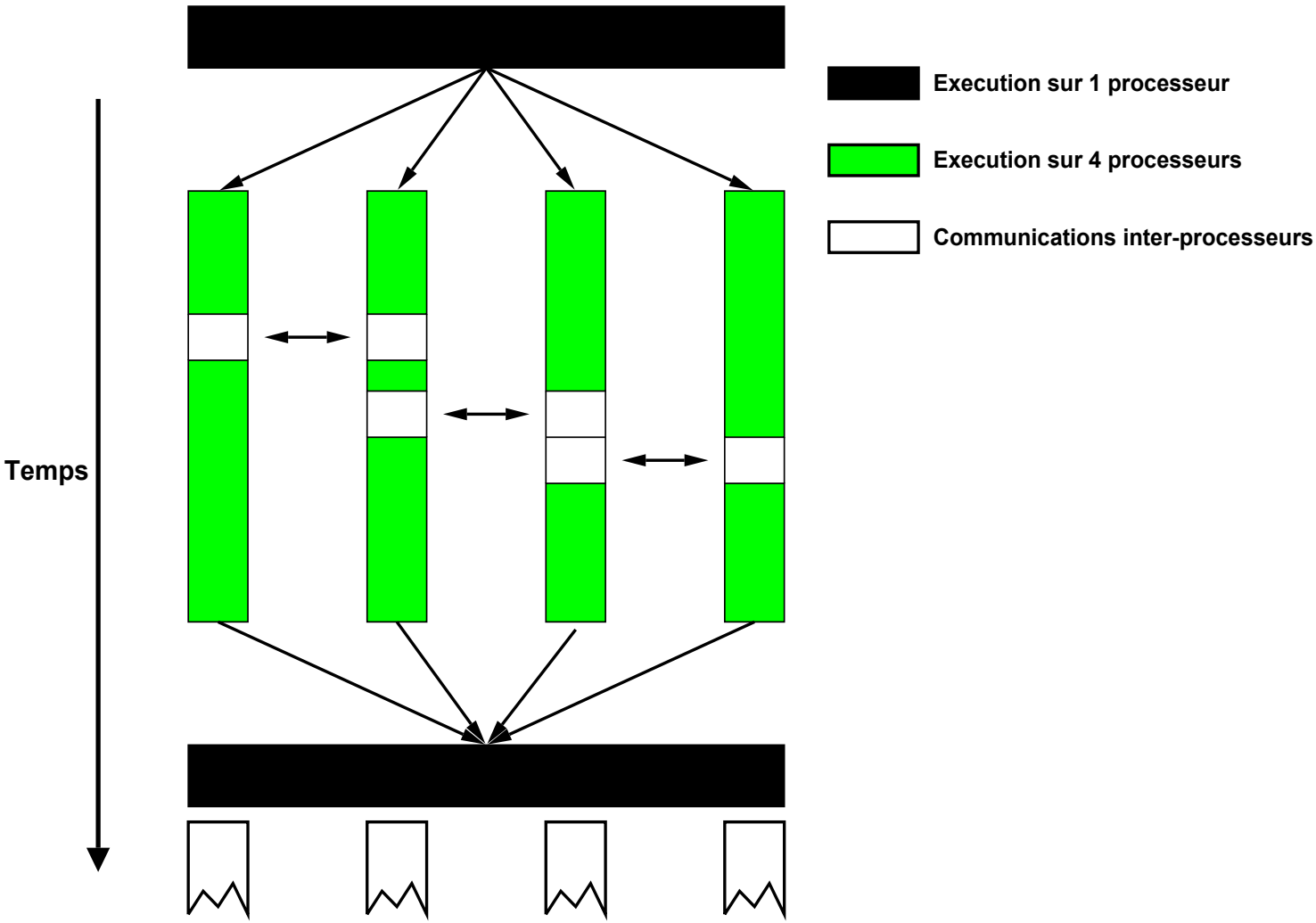
- Exécution plus rapide du programme (gain en temps de restitution) en distribuant le travail
- Résolution de problèmes plus gros (plus de ressource matérielle accessible, notamment la mémoire)

But de la parallélisation

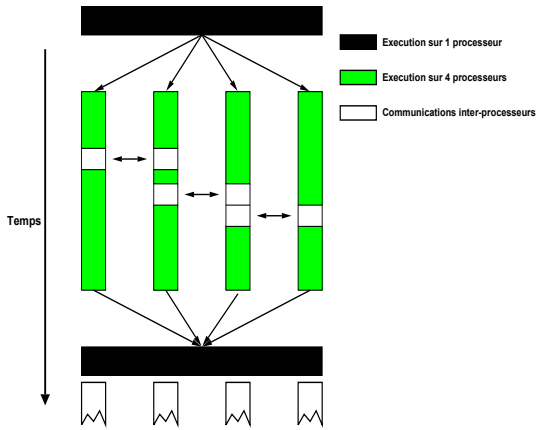


- Diminution du temps d'exécution – le but
- (Faible) augmentation du temps CPU total - effet de bord

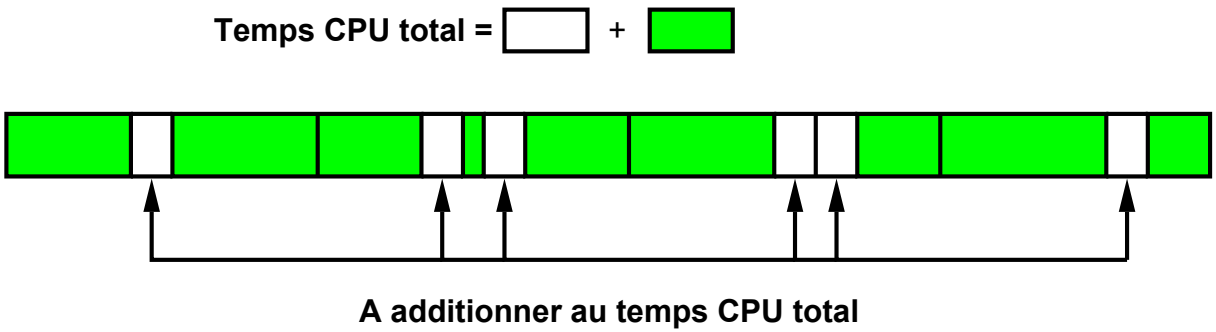
Optimisation de la parallélisation



Optimisation de la parallélisation



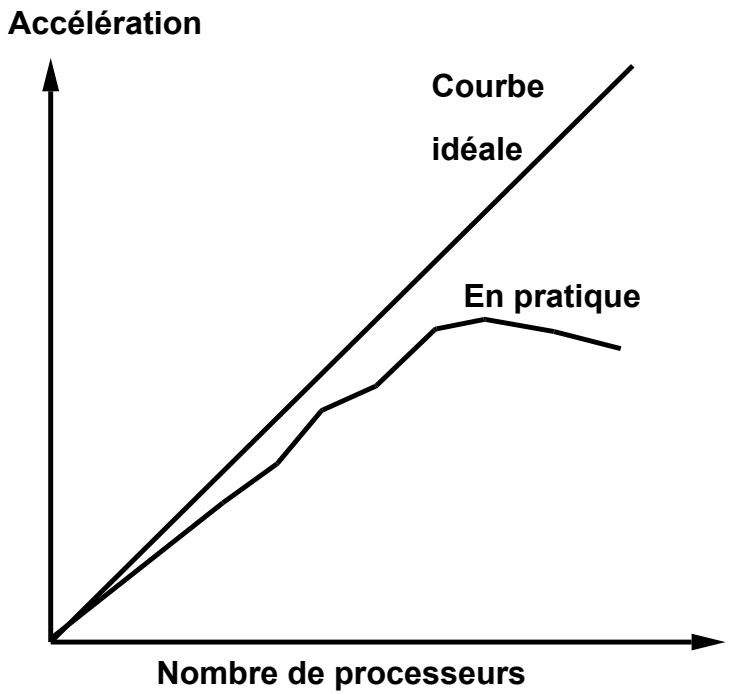
Une parallélisation efficace
minimise les communications

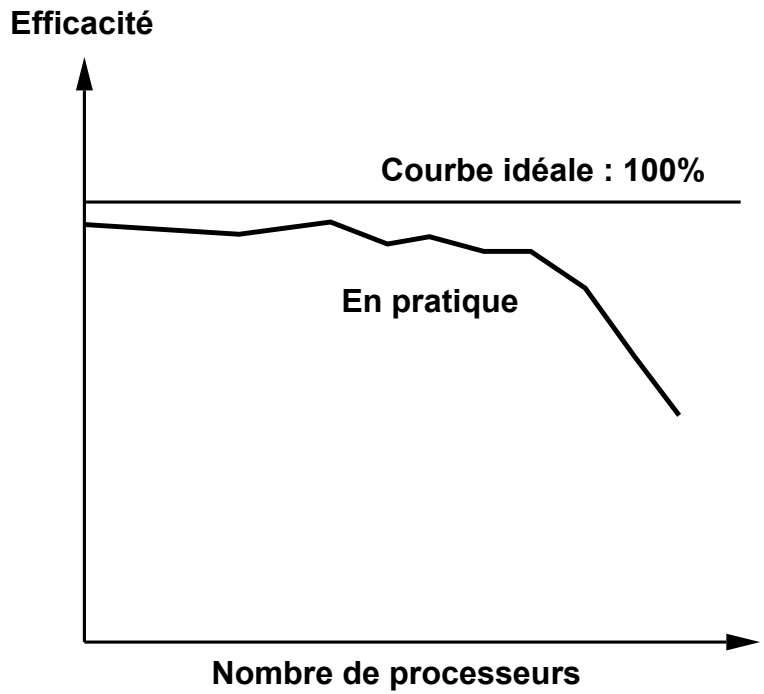


Accélération et efficacité

- Les deux sont une mesure de la qualité de la parallélisation
- Soit `T(p)` le temps d'exécution sur `p` processeurs
- *L'Accélération* `A(p)` et *l'Efficacité* `E(p)` sont définies comme étant :
  A(p) = T(1) / T(p) (p=1, 2, 3, ...)
  E(p) = A(p) / p
- Pour une accélération parallèle parfaite on obtient :
  T(p) = T(1) / p
  A(p) = T(1) / T(p) = T(1) / (T(1) / p) = p
  E(p) = A(p) / p = p / p = 100%

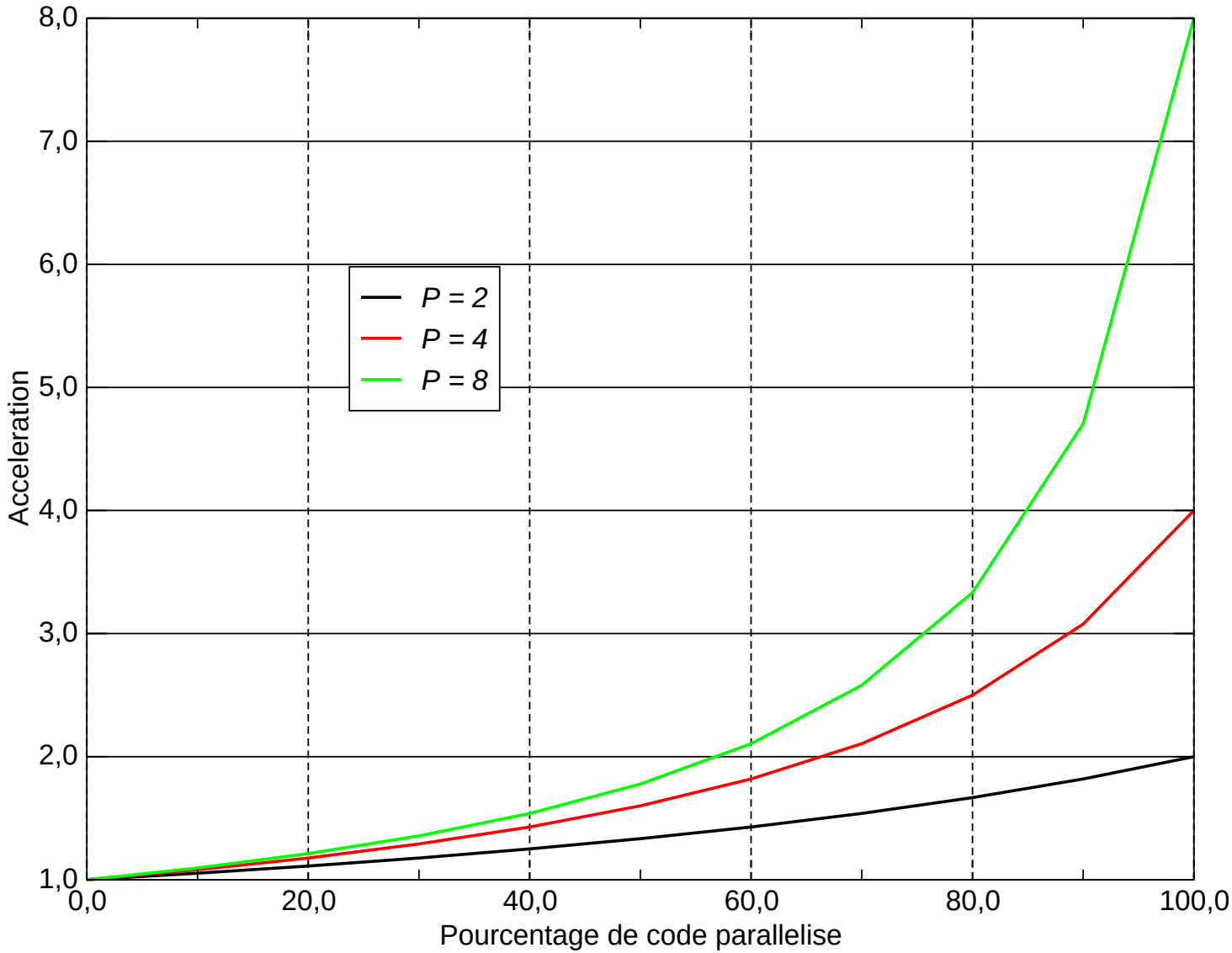
Accélération et efficacité





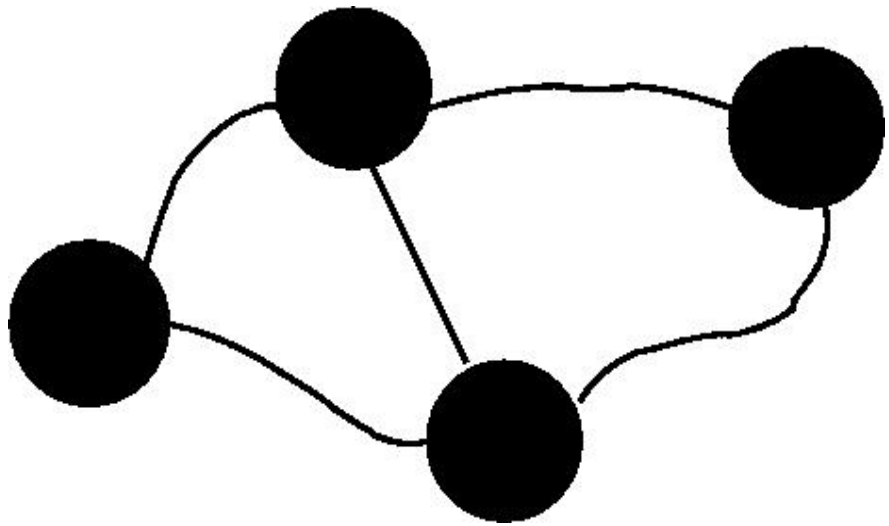
- Les programmes scalables demeurent efficaces pour un grand nombre de processeurs
  (scalables = passage à l'échelle)

Loi d'Amdahl


8,0
7,0
6,0
5,0
4,0
3,0
2,0
1,0
Acceleration
P = 2
P = 4
P = 8
0,0
20,0
40,0
60,0
80,0
100,0
Pourcentage de code parallelise

Interconnexion

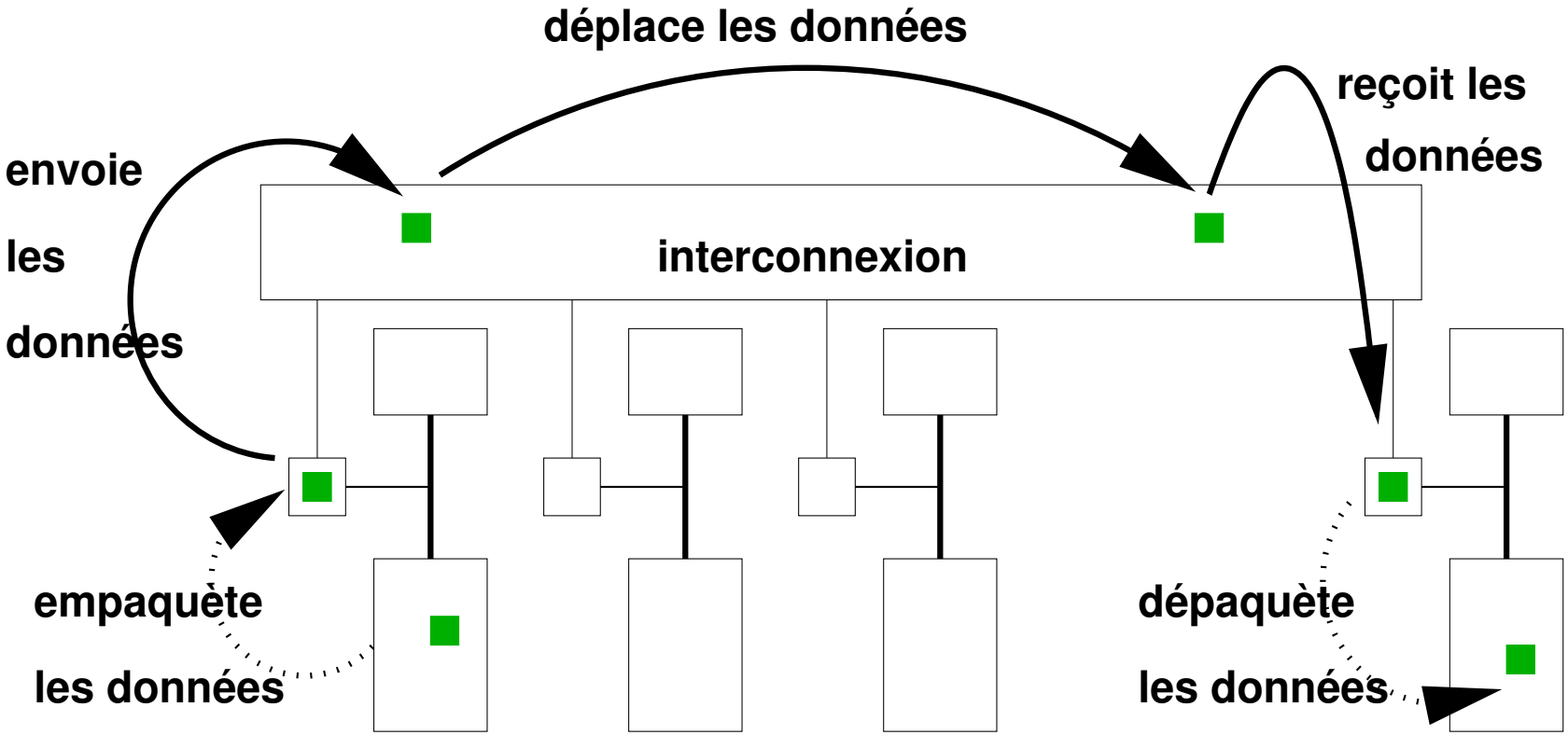
- Pour un programmeur, un point essentiel réside dans le type de mémoire de l'architecture cible : mémoire partagée ou mémoire distribuée



- L'interconnexion a un grand impact sur :
  - le choix de l'implémentation de la parallélisation
  - la performance parallèle
  - la facilité d'utilisation
- L'application elle-même, va influer sur la relative importance de ces facteurs

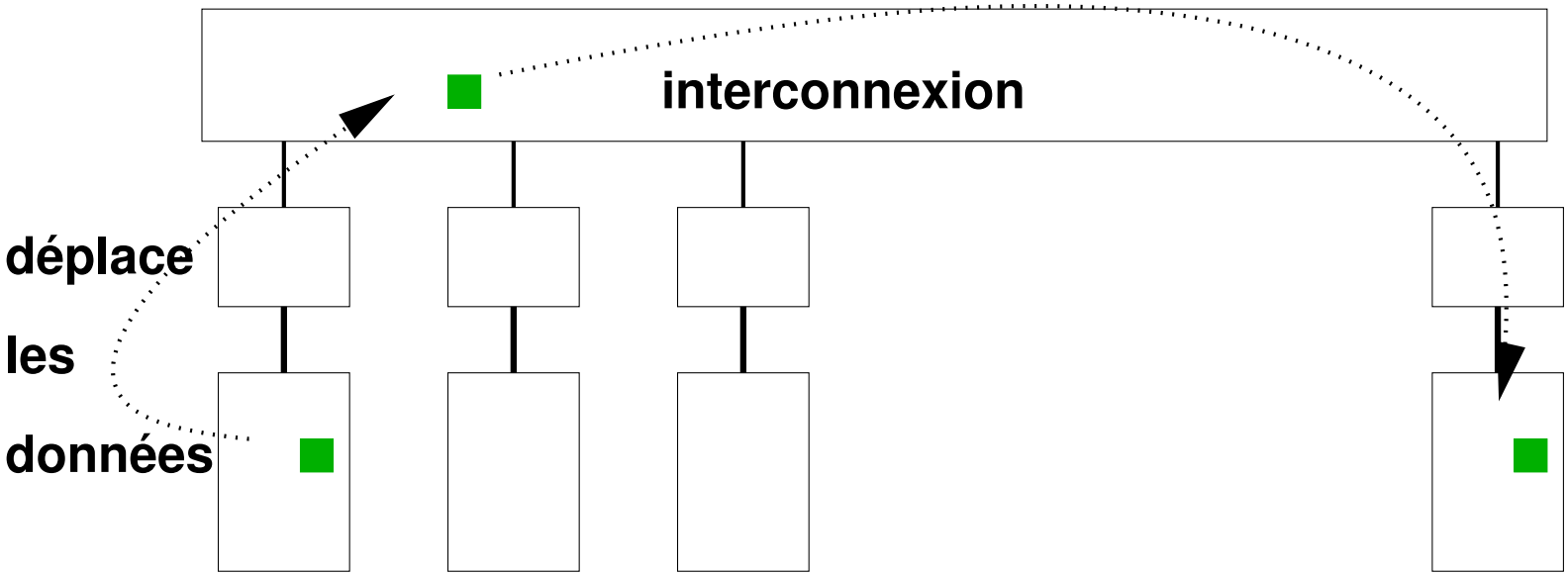
Ordinateur à mémoire distribuée

- Le système est programmé en utilisant l'échange de messages
- protocole réseau



Ordinateur à mémoire partagée

- Le mouvement des données est transparent pour l'utilisateur
- transfert mémoire



S.M.P. = Symmetric Multi-Processor
(tous les processeurs sont *égaux*)

MAIS

Attention aux accès vers la mémoire locale mais distante !

Types de parallélisme

- Parallélisme de tâches
  Plusieurs traitements
  Traitements indépendants
  $\Rightarrow$ Echanges de données, directives de contrôle
  Le parallélisme découle de la réalisation simultanée de différents traitements sur les données

- Parallélisme de données
  Données régulières et en grand nombre
  Traitement identique sur chaque donnée
  $\Rightarrow$ Programmation Data-parallèle
  Le parallélisme découle de l'application simultanée du même traitement sur des données différentes

Introduction

Parallélisme en mémoire distribuée : MPI
- Structures de données
- Communications point à point
- Communications collectives
- Diffusion / collecte
- Topologie
- Mise en œuvre de MPI

Parallélisme en mémoire partagée : OpenMP

Evolution

TP

MPI : Message-passing library interface specification

- C'est une norme définissant une bibliothèque de fonctions, initialement utilisable avec les langages C, C++ et Fortran
- En constante évolution depuis sa création en 1993-1994
- MPI 2.0 : bindings F90, C++ ; MPI I/O ; processus dynamiques
- MPI 3.0 : comm. coll. non bloquantes, extensions des one-sided operations
- Des implémentations existent pour ces langages ainsi que pour Python, Perl, Java, ...

Caractéristiques du passage de messages avec MPI

- Il repose sur l'échange de messages entre les processus pour le transfert de données, les synchronisations, les opérations globales
- La gestion de ces échanges est réalisée par MPI (Message Passing Interface)
- Chaque processus dispose de ses propres données, sans accès direct à celles des autres
- Explicite, cette technique est entièrement à la charge du développeur
- Ces échanges qui impliquent deux ou plusieurs processus se font dans un communicateur
- Chaque processus est identifié par son rang, au sein du groupe

Environnement

- Initialisation en début (MPI_Init)
- Finalisation en fin de programmme (MPI_Finalize)

```
int argc, rank, size;
char *argv[];
MPI_Init (&argc, &argv);
MPI_Comm_Size ( MPI_COMM_WORLD, &rank);
MPI_Comm_Rank ( MPI_COMM_WORLD, &size);
...
MPI_Finalize ();
```

- Pour réaliser des opérations impliquant des données d'autres processus, il est nécessaire d'échanger ces informations aux travers de messages
- Ces messages se font sous la forme de communications impliquant au moins deux processus
- On peut faire une analogie avec le courrier électronique

Structures de données

- Les données transmises sont typées
- Types prédéfinis : `MPI_INT`, `MPI_DOUBLE`, ...
- Type homogène :
  - données contigües : fonction `MPI_Type_Contiguous`
    ⇒ ligne de matrice en C
  - données distantes d'un pas constant : fonctions
    `MPI_Type_Vector` ou `MPI_Type_HVector`
    ⇒ ligne ou bloc d'une matrice
  - données distantes d'un pas variable : fonctions
    `MPI_Type_Indexed` ou `MPI_TYPE_HIndexed`
    ⇒ triangle dans une matrice
  - portion de tableau multi-dimensionnel : fonction
    `MPI_Type_Create_Subarray`
- Type hétérogène :
  ⇒ Construction d'une structure : fonctions `MPI_Type_Struct`
- Validation d'un type : fonction `MPI_Type_Commit`
- Destruction d'un type : fonction `MPI_Type_Free`

Communications point à point

- La communication point à point est une communication entre deux processus :
  $\Rightarrow$ expéditeur et destinataire
- Composition d'un message :
  le communicateur (comm)
  les deux identifiants (src et dest)
  la donnée (buf), son type (datatype) et sa taille (count)
  une étiquette (tag) qui permet au programme de distinguer différents messages

Communications bloquantes (il existe des variantes ...) :

```
int MPI_Send (const void* buf, int count, MPI_Datatype
        datatype, int dest, int tag, comm, MPI_Comm)

int MPI_Recv (void* buf, int count, MPI_Datatype,
        datatype, int src, int tag, MPI_Comm comm,
        MPI_Status *status)
```

Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <math.h>

int main(argc, argv)
int argc;
char *argv[];
{
   int myid, numprocs;
   int tag, source, destination, count;
   int buffer, rc;
   MPI_Status status;

   MPI_Init (&argc, &argv);
   MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
   MPI_Comm_rank (MPI_COMM_WORLD, &myid);
   tag = 1234;
   source = 0;
   destination = 1;
   count = 1;
   if (myid == source) {
      buffer = 5678;
      rc = MPI_Send (&buffer, count, MPI_INT, destination, tag, MPI_COMM_WORLD);
      printf ("processor %d sent %d\n", myid, buffer);
   }
   if (myid == destination) {
      rc = MPI_Recv (&buffer, count, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
      printf ("processor %d got %d\n", myid, buffer);
   }
   MPI_Finalize ();

}
```

Communications collectives

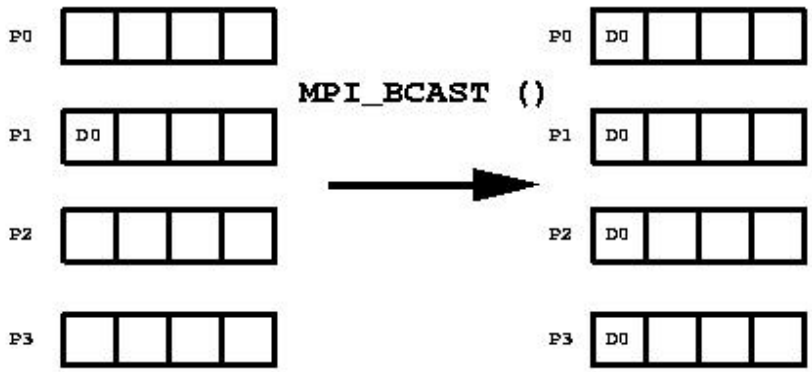
- La communication collective est une communication qui implique un ensemble de processus qui l'effectuent tous
- **les synchronisations globales** : une barrière de synchronisation qui agit sur l'ensemble des membres d'un communicateur.

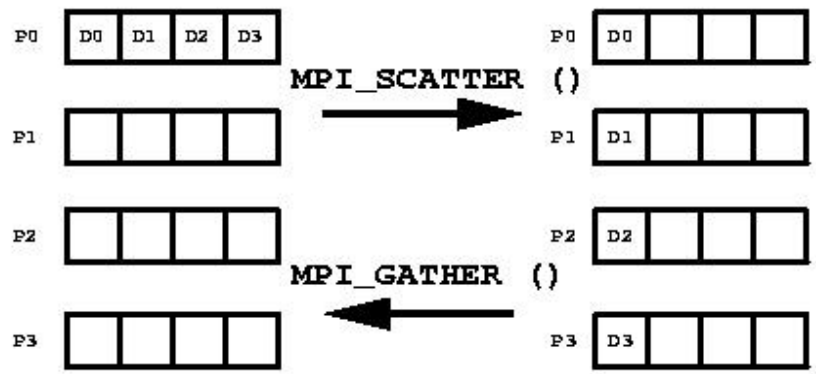
```
int MPI_Barrier (MPI_Comm comm)
```

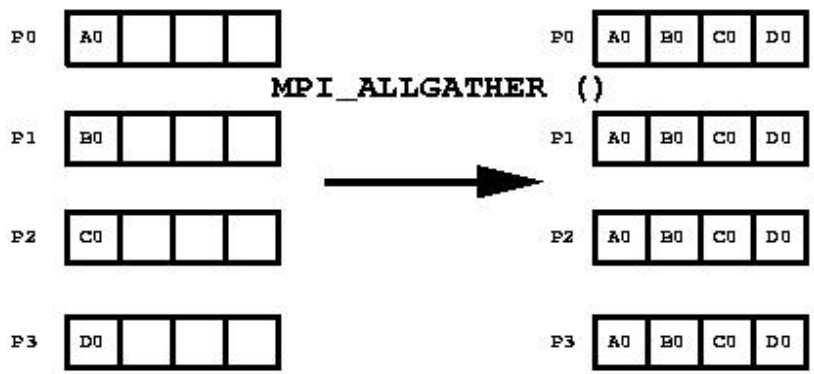
- **les opérations de réduction sur des données réparties** : somme, produit, maximum ... et le résultat peut être ensuite redistribué (`MPI_AllReduce`) ou pas (`MPI_Reduce`).

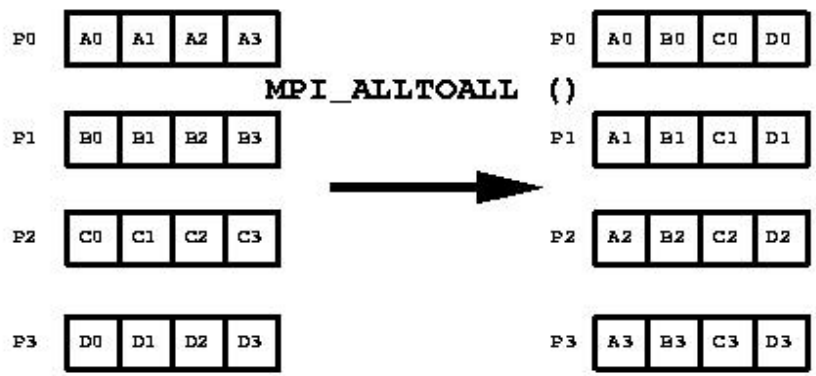
```
int MPI_Reduce (const void* sbuf, void* rbuf,
                int count, MPI_Datatype datatype,
                MPI_Op oper, int root, MPI_Comm comm)

int MPI_AllReduce (const void* sbuf, void* rbuf,
                int count, MPI_Datatype datatype,
                MPI_Op oper, MPI_Comm comm)
```

Diffusion / collecte



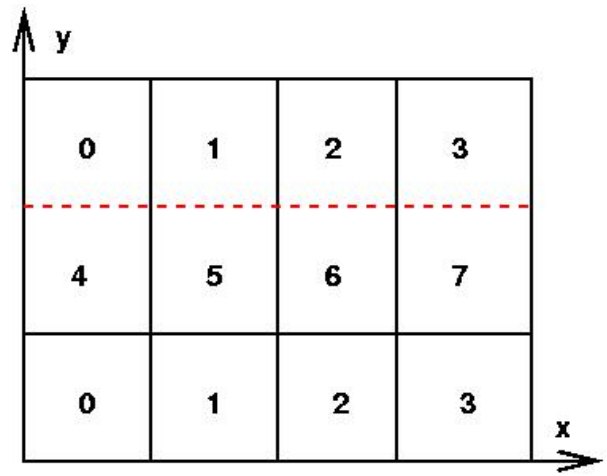






Topologie

- Ordonner intelligemment ses processus et s'y retrouver !
- Topologie cartésienne : grille de processus
- Nombre de proc. par dimension d'espace : `MPI_Dims_Create`
- Création de la grille : `MPI_Cart_Create`
  $\Rightarrow$ périodicité ou non des conditions aux limites
  $\Rightarrow$ création d'un nouveau communicateur



- Recherche des voisins dans chaque dimension : `MPI_Cart_Shift`
- Coordonnées / rang : `MPI_Cart_Coords` et `MPI_Cart_Rank`

Implémentation OpenMPI : installation

- https../Graphics/www.open-mpi.org est **une** implémentation

- Installation classique : **configure**, **make**, **make install**

- Par exemple
  ./configure CC=gcc CXX=g++ F77=gfortran FC=gfortran –prefix=/logiciels/openmpi/1.10.1/gnu **–enable-mpi-thread-multiple –enable-static=**yes **–with-slurm –enable-orterun-prefix-by-default**

- **–enable-mpi-thread-multiple** : parallélisation hybride
- **–with-slurm** : interaction avec le gestionnaire de ressources
- **–enable-orterun-prefix-by-default** : chemins complets vers les fichiers

- Utiliser les modules ! ($\rightarrow$ Bernard !)

Implémentation OpenMPI : utilisation

- Utilisation
  - wrapper : **mpicc**, **mpif90**, ...
  - include : **mpi.h**, **mpif.h**, ...
  - bibliothèque : **libmpi.so**, ...
- Exécution
  - **mpirun -np 4 ./a.out**
  - **ompi_info** (options) : fournit des renseignements sur le paramétrage
- Voir le petit TP d'utilisation
- Placement des processus : important, essentiel pour la parallélisation hybride.
  - **–bind-to <foo>** slot, core, L3cache, socket, node, ppr
- Numérotation des processus
  - **–rank-by <foo>** : slot, core, L2cache, socket, node
- Voir la manpage de **mpirun** pour les détails

Introduction

Parallélisme en mémoire distribuée : MPI

Parallélisme en mémoire partagée : OpenMP
- Principe d'OpenMP
- Directives
- Visibilité des variables
- Autres objets OpenMP
- Partage du travail
- Sérialisation et synchronisations
- Implémentation OpenMP

Evolution

TP

OpenMP : qu'est-ce que c'est ?

OpenMP est :

- une API (Application Program Interface) qui permet de faire du parallélisme en mémoire partagée, multi-threadé
- composé de 3 éléments de base :
  - des directives de compilation
  - une bibliothèque de routines spécialisées
  - des variables d'environnement
- portable :
  - l'API est supportée par le Fortran (77, 9x), le C/C++
  - l'essentiel des plates-formes le supportent
- standardisé
  - défini et approuvé conjointement par l'essentiel des constructeurs informatiques et éditeurs de logiciels
  - peut-être bientôt un standard ANSI
- Que signifie le nom OpenMP ?
  - version courte : Open specifications for MultiProcessing
  - version longue : spécifications ouvertes pour le multi-processing, définies par les mondes de l'Industrie et de la Recherche

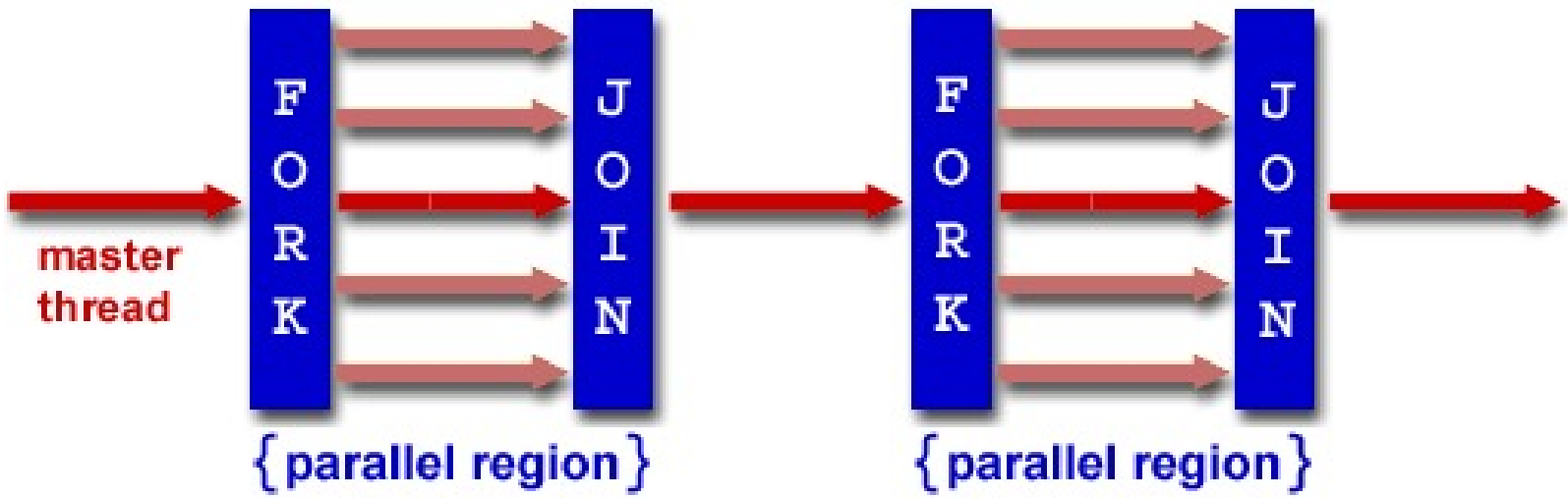
Historique d'OpenMP

- Au début des années 90, chaque constructeur informatique avait son jeu de directives pour le multitâche sur ses supercalculateurs à mémoire partagée
- Les implémentations étaient toutes fonctionnelles mais totalement différentes et incompatibles
- Une première tentative de standardisation (ANSI X3H5) n'a jamais été adoptée
- La spécification du standard OpenMP date du printemps 1997
- Évolution des différentes normes :

Oct 1997	Fortran 1.0	Nov 2000	Fortran 2.0	Mai 2008	OpenMP 3.0
Oct 1998	C/C++ 1.0	Mar 2002	C/C++ 2.0	Juil 2011	OpenMP 3.1
Nov 1999	Fortran 1.1	Mai 2005	OpenMP 2.5	Juil 2013	OpenMP 4.0

Principe d'OpenMP (1/3)

- Un programme OpenMP est exécuté par un processus unique
- Ce processus active des processus légers (threads) à l'entrée d'une région parallèle



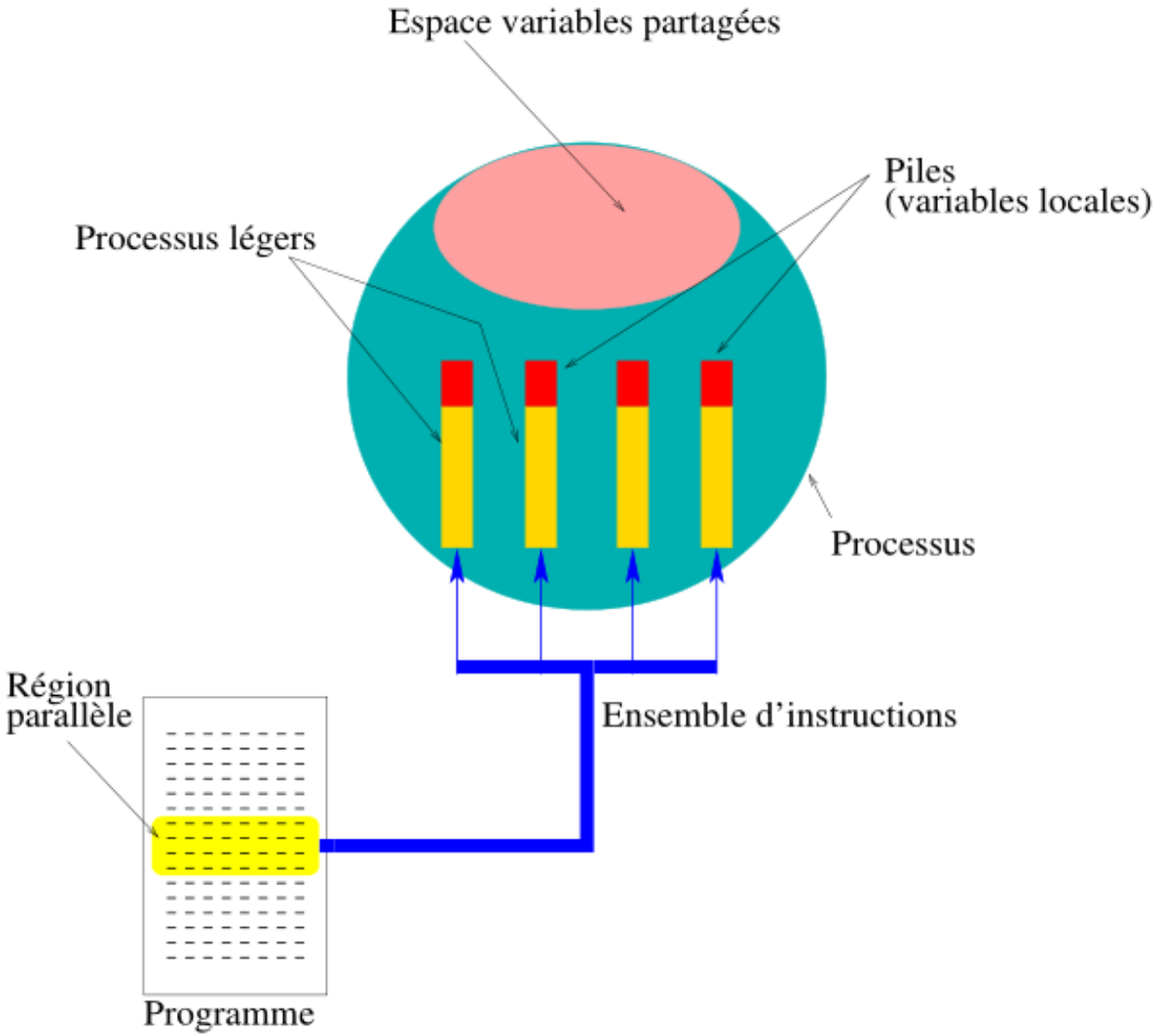
(LLNL)

- Un programme OpenMP est une alternance de régions séquentielles et parallèles

Principe d'OpenMP (2/3)

- Une région séquentielle est toujours exécutée par une seule tâche, la tâche maître
- Une région parallèle peut être exécutée par plusieurs tâches à la fois
- Les tâches peuvent se partager le travail contenu dans la région parallèle
- Le nombre de tâches veut varier d'une région parallèle à l'autre
- OpenMP ne spécifie rien concernant les I/O. C'est au développeur de faire en sorte que les I/O se déroulent correctement dans un contexte multi-tâche

Principe d'OpenMP (3/3)



(IDRIS)

Directives

La parallélisation se fait en insérant des directives dans le code séquentiel pour construire des régions parallèles :

```
#pragma omp parallel [clause [,clause] ...] new-line
```

Où :

`#pragma omp` en C/C++	est une sentinelle, obligatoire comme préfixe des directives
`parallel`	est une directive
`[clause [,clause] ...]`	sont les clauses, optionnelles, qui permettent de décrire la visibilité des variables

Visibilité des variables

La visibité des données se définit à l'aide de clauses : soit les variables sont partagées, soit elles sont privées.

- Variables partagées :
  - les variables le sont par défaut
  - tous les threads accèdent à la même instance de la variable (attention aux conflits !)
- Variables privées :
  - elles n'existent qu'au sein de leur région parallèle
  - elles peuvent êtrer initialisées, persistantes, ..

Autres objets OpenMP

D'autres objets existent :

- clauses : `firstprivate`, `lastprivate`, `copyprivate`, `copyin`,...

- directives : `threadprivate`, ...

- sous-programmes : `omp_set_num_threads()`, `omp_get_wtime()`, `omp_get_num_threads()`, `omp_get_thread_num()`, ...

- variables d'environnement : `OMP_NUM_THREADS`, `OMP_SCHEDULE`, `OMP_DYNAMIC`, ...

- un fichier d'en-tête pour le C/C++ `omp.h`, permettent de définir les prototypes des fonctions OpenMP

Partage du travail 1/2

Les constructions parallèles permettent de partager le travail entre les threads.

Boucle dans une région parallèle :

```
#pragma omp parallel
...
#pragma omp for
for (i=1; i<n; i++)
   mywork(i);
```

Construction parallèle restreinte à une triple boucle :

```
#pragma omp parallel for collapse(3) \
   default(none) \
   shared(a,nx,ny,nz) private(i,j,k)
for (k=1; k<=nz; k++)
  for (j=1; j<=ny; j++)
     for (i=1; i<=nx; i++)
        bar(a,i,j,k);
```

Partage du travail 2/2

Une section est une portion de code exécutée par un seul thread.

```
#pragma omp parallel sections default(none) \
        shared(x,y,z,nx,ny,nz)
{
   #pragma omp section
      xaxis(x,nx);

   #pragma omp section
      yaxis(y,ny);

   #pragma omp section
      zaxis(z,nz);
}
```

Attention : méthode non scalable

Sérialisation et synchronisations

Il y a plusieurs constructions qui permettent de restreindre ou spécifier l'ordre d'accès à des données partagées

- directive `master { ... }`
  accès pour le thread de rang 0 uniquement

- directive `single [clause] { ... }`
  accès pour un seul thread, non déterminé à l'avance

- directive `atomic`
  instruction d'affectation suivante réalisée de manière atomique

- directive `critical [nom] { ... }`
  accès pour un seul thread à la fois

- directive `barrier`
  barrière de synchronisation globale

Implémentation OpenMP

- Elle est incluse dans les compilateurs les plus courants
- Son activation se fait au travers d'une option de compilation
  - **GNU** : **-fopenmp**
  - **Intel** : **-openmp**
  - **PGI** : **-mp**
- On peut binder les threads sur les cœurs, à l'aide de variables d'environnement
  - **OpenMP** : **OMP_BIND_PROC**
  - **GNU** : **GOMP_OMP_AFFINITY**
  - **Intel** : **KMP_AFFINITY**
  - **PGI** : **MP_BIND**

Implémentation OpenMP

top - 11 :04 :31 up 70 days, 20 :18, 2 users, load average : 12.54, 3.91, 1.45
Mem : 193663M total, 31495M used, 162168M free, 191M buffers
Swap : 24575M total, 76M used, 24499M free, 27626M cached

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
34247	gmoebs	20	0	1141m	1.0g	1520	R	100	0.5	0 :41.16	C3Dgc_1omp.
34245	gmoebs	20	0	1141m	1.0g	1520	R	100	0.5	0 :41.18	C3Dgc_1omp.
34248	gmoebs	20	0	1141m	1.0g	1520	R	100	0.5	0 :41.17	C3Dgc_1omp.
34250	gmoebs	20	0	1141m	1.0g	1520	R	100	0.5	0 :41.17	C3Dgc_1omp.
34251	gmoebs	20	0	1141m	1.0g	1520	R	100	0.5	0 :41.17	C3Dgc_1omp.
34252	gmoebs	20	0	1141m	1.0g	1520	R	100	0.5	0 :41.17	C3Dgc_1omp.
34253	gmoebs	20	0	1141m	1.0g	1520	R	100	0.5	0 :41.17	C3Dgc_1omp.
34255	gmoebs	20	0	1141m	1.0g	1520	R	100	0.5	0 :41.17	C3Dgc_1omp.
34258	gmoebs	20	0	1141m	1.0g	1520	R	100	0.5	0 :41.17	C3Dgc_1omp.
34260	gmoebs	20	0	1141m	1.0g	1520	R	100	0.5	0 :41.17	C3Dgc_1omp.
34266	gmoebs	20	0	9320	1580	920	R	1	0.0	0 :00.43	top

...
...

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
34245	gmoebs	20	0	1141m	1.0g	1520	R	1996	0.5	23 :00.29	C3Dgc_1omp.
34266	gmoebs	20	0	9320	1580	920	R	1	0.0	0 :00.68	top

Passage de messages ou mémoire partagée ?

$\Rightarrow$ quelques observations ...

Passage de Messages	Mémoire partagée
Demande plus de temps à implémenter	Plus facile à implémenter
Plus de détails à prendre en compte	Le système gère pas mal de détails
Augmente la taille du code	Faible accroissement de la taille du code
Plus de maintenance	Peu de maintenance additionnelle
Complexe à déboguer et à optimiser	Plus facile à déboguer et à optimiser
Augmente le volume mémoire	Usage efficace de la mémoire
Meilleur pour une scalabilité optimale	Scalable, mais ....
Parallélisme visible	Parallélisme traité par le compilateur

Introduction

Parallélisme en mémoire distribuée : MPI

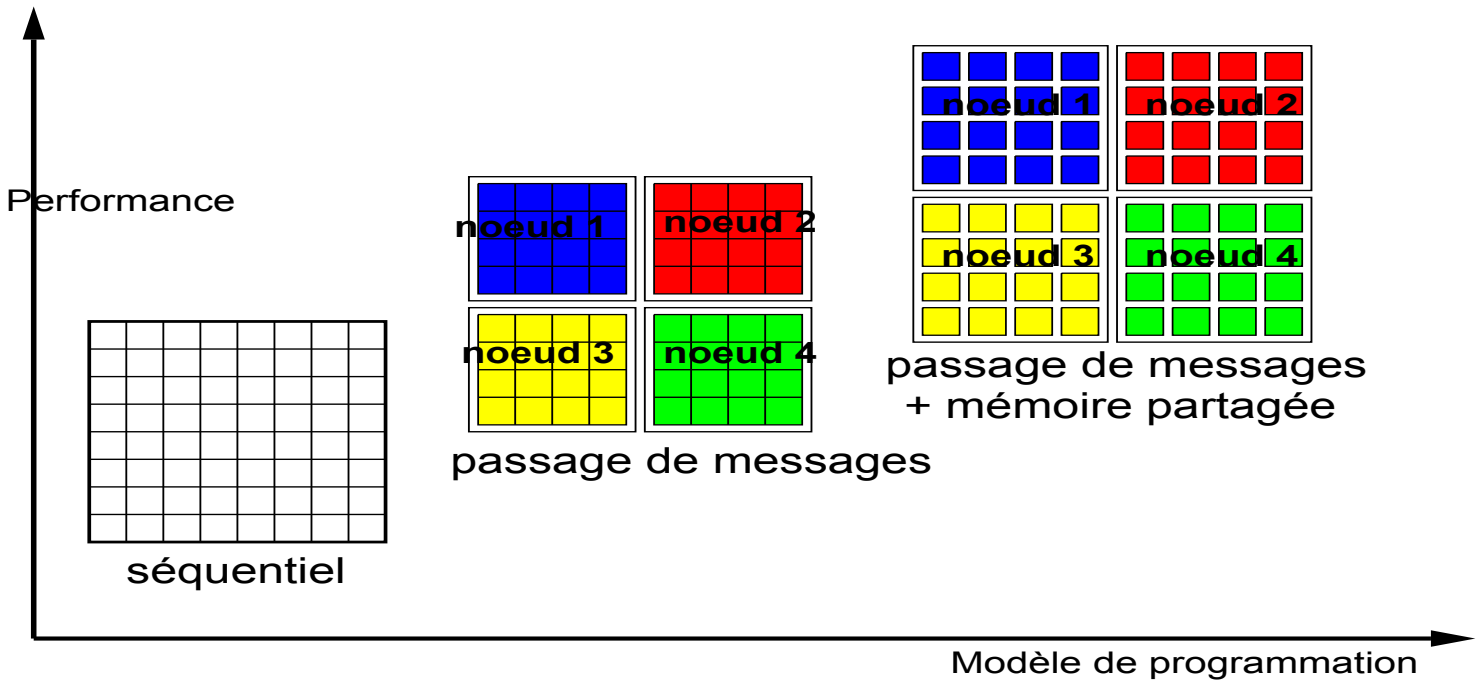
Parallélisme en mémoire partagée : OpenMP

Evolution

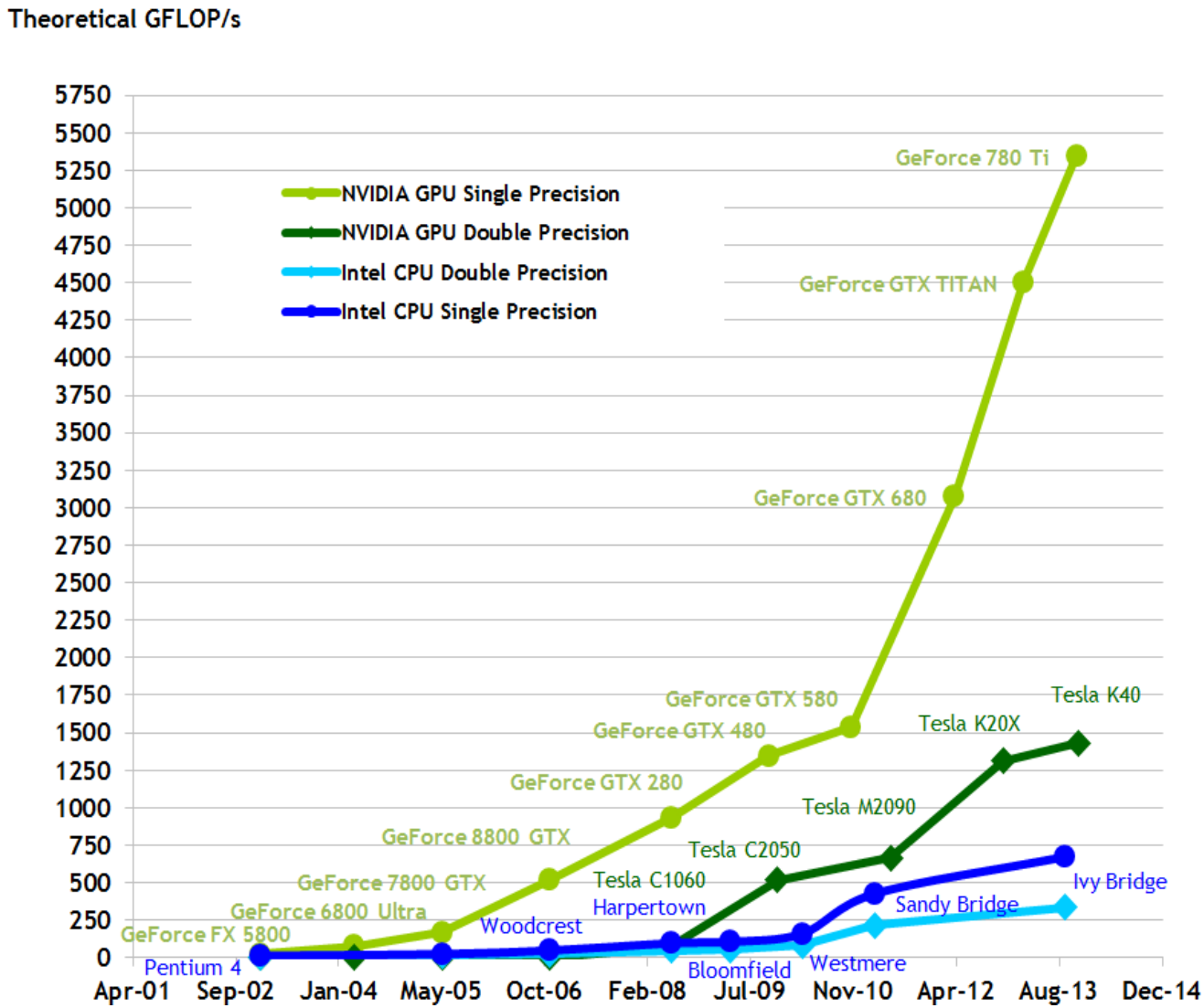
TP

Evolution logicielle : modèles hybrides

- L'hybride MPI+OpenMP est de plus en plus utilisé, il est notamment bien adapté aux architectures type cluster de multiprocesseurs



Evolution matérielle : cartes accélératrices



Introduction

Parallélisme en mémoire distribuée : MPI

Parallélisme en mémoire partagée : OpenMP

Evolution

TP

TP sur les performances réseau

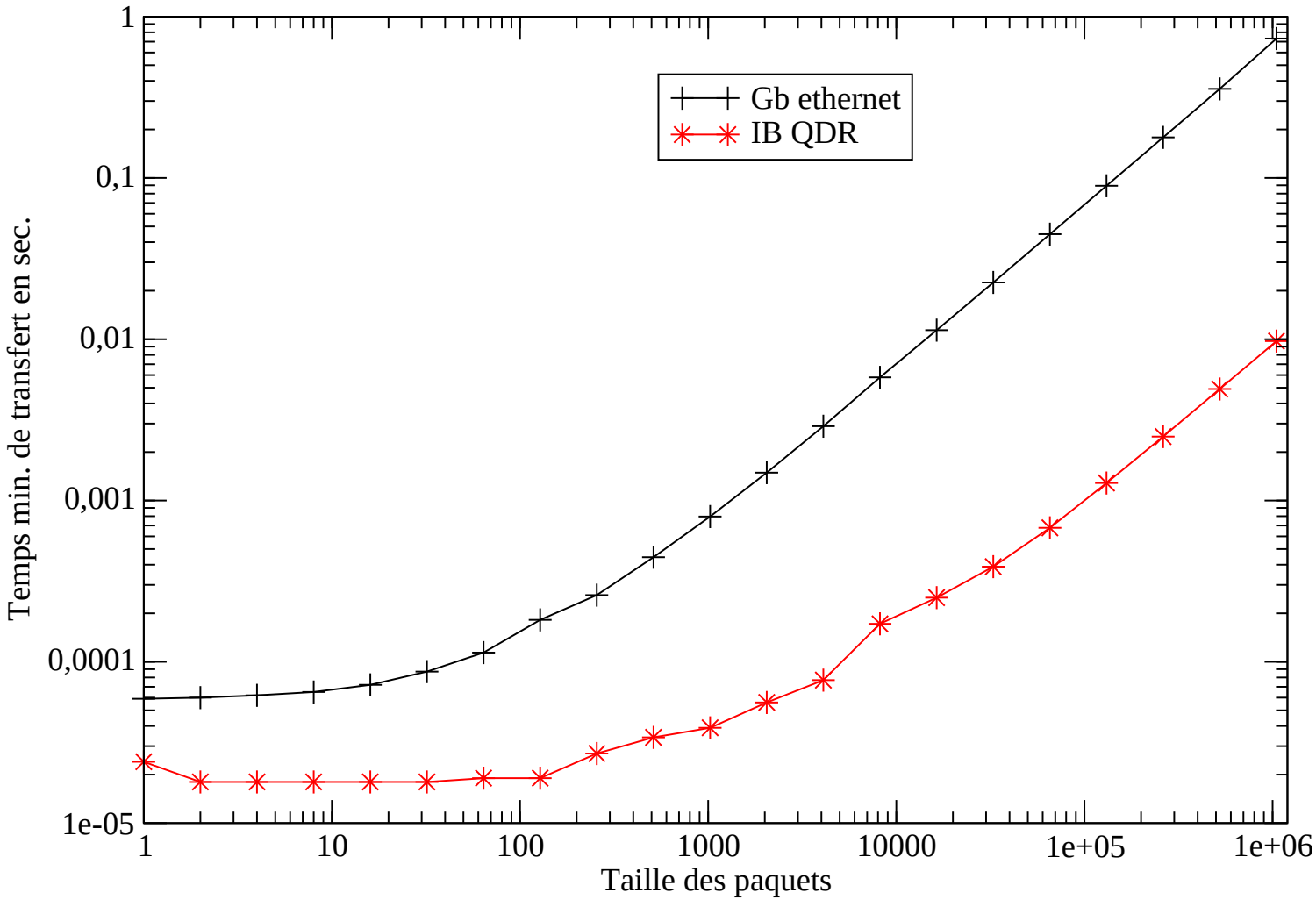
- Il s'agit d'un petit programme qui fait du ping-pong entre deux nœuds et prend l'un ou l'autre des réseaux.
  (source : `https://www.hpc2n.umu.se/node/230`)

- Compilation : **mpicc -O2 hello.c**

- Exécution :
  mpirun -np 2 –mca btl tcp,self -host goulaine01,goulaine02
  –mca btl_tcp_if_include ib0 ./a.out

  mpirun -np 2 –mca btl tcp,self -host goulaine01,goulaine02
  –mca btl_tcp_if_include em1 ./a.out

Comparaison des temps de transfert



Comparaison des débits