École « Programmation hybride : une étape vers le many-coeurs ? »

Placement des threads et aspects mémoire

10 Octobre 2012

Autrans

Ludovic Saugé, PhD

Leader Technique Hardware
Ingénieur Applications Senior

Applications & Performance Team
Bull Extreme Computing BU

ludovic.sauge@bull.net

Plan

1

- Introduction
  - Hiérachie mémoire
  - Métriques
    - Latence
    - Débit
  - Architectures multi-processeurs
    - SMP
    - (cc-)NUMA
  - Exemples d'architectures NUMA
    - Effets NUMA
    - Effets NUMA IOs
  - Détermination des topologies

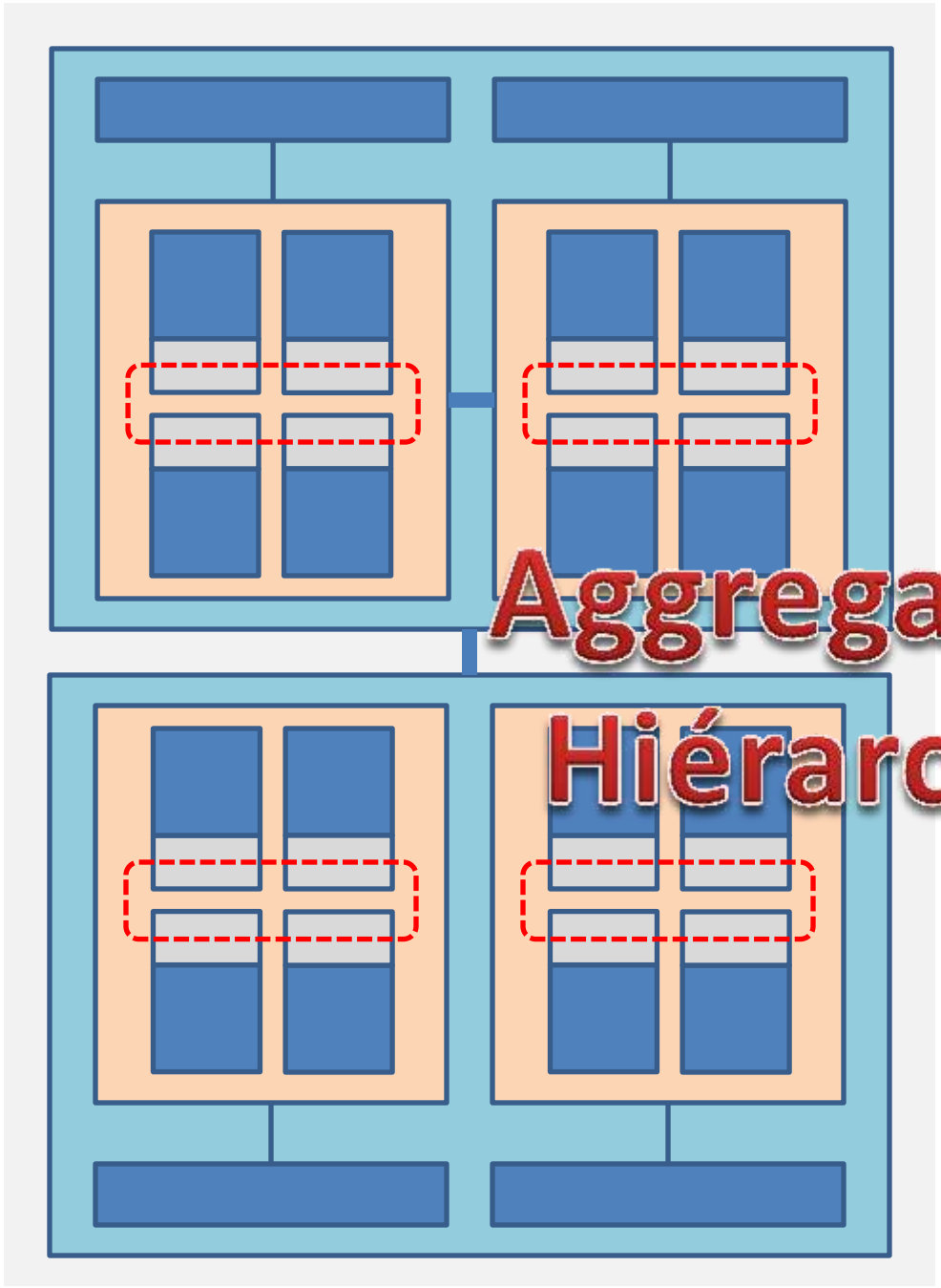
2

- Placement des threads & Optimisation des accès Intranoeuds
  - Interface bas niveau (noyau)
  - Placement et Attachement
    - Bitmask
  - Motivation
    - Switch de context
    - First Touch Policy
    - Cache IO
  - Méthodes
    - API Posix
    - taskset
    - libnuma
    - API
    - numactl
    - Placement des threads OpenMP
    - cpuset et confinement des processus
  - Huge pages

3

- Placement des processus MPI & Travaux Hybrides
  - Placement MPI (lanceur)
  - Distribution des rangs
  - Wrapper
  - Lib d'interposition

Introduction



Aggregation Hiérarchie

Cluster

Interconnect "classique"

Noeuds de calculs

Lien interprocesseur (HT, QPI)

Socket/die

Interconnect on chip

Coeurs
Caches

Accès aux données

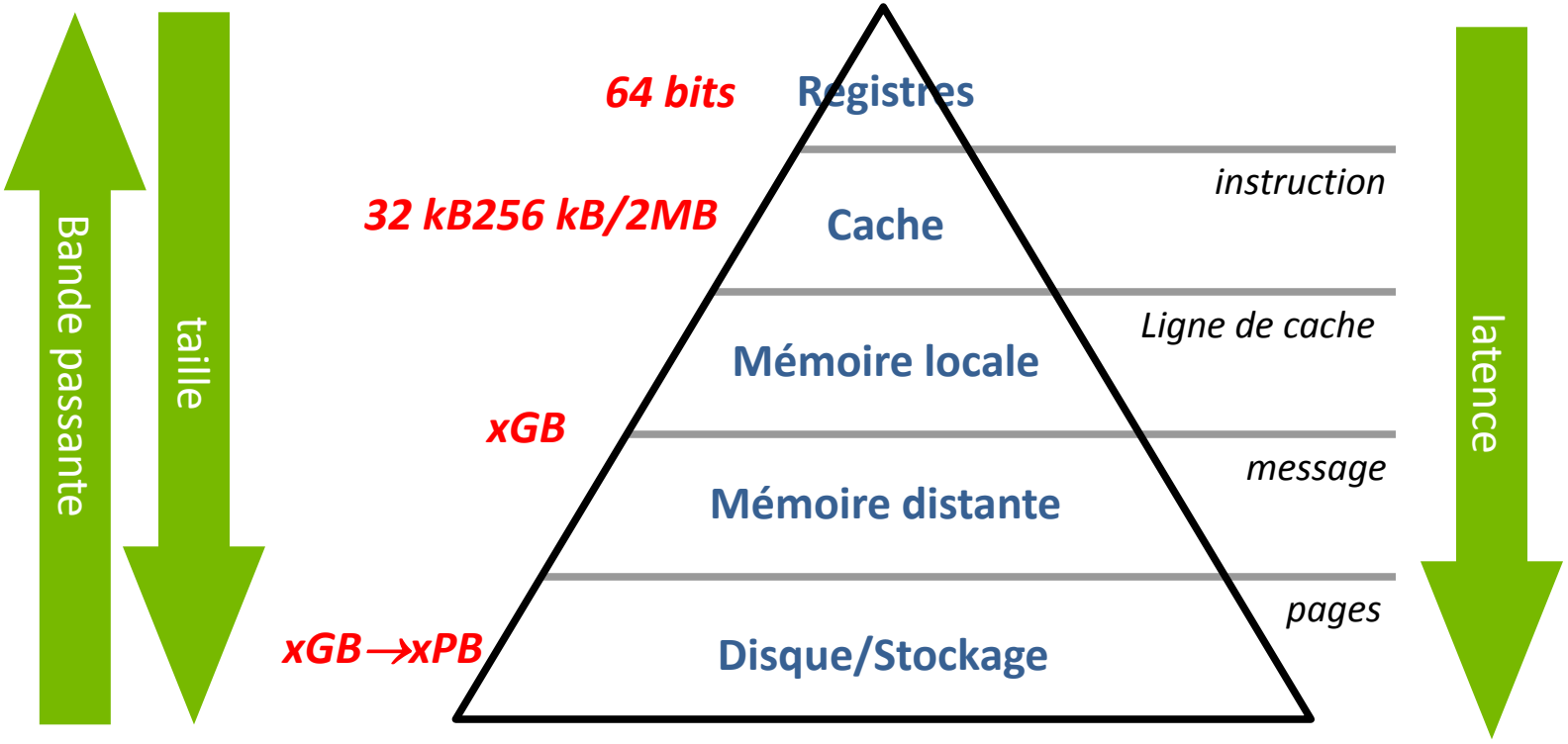
- Il est question des **accès aux données**
  - En mémoire
  - Au travers de périphériques (Interconnect, Accelérateur, Co-processeurs,…)

- Pour l'utilisateur, les métriques pertinentes sont :

  - **La latence** : le temps d'accès à une donnée (cycles processeur ou temps)
  - **Le débit** : la bande passante (GiB/secondes)

- **Quelle est la métrique la plus pertinente ?**
  - Ca dépend !
    - Par exemple, pour les accès mémoires
      - Accès aléatoires : latence
        - » Accès au cache
        - » Accès à la mémoire centrale
      - Accès séquentiels : débit
      - Mais avant ça, votre code est-il vraiment « **memory bound** » ?

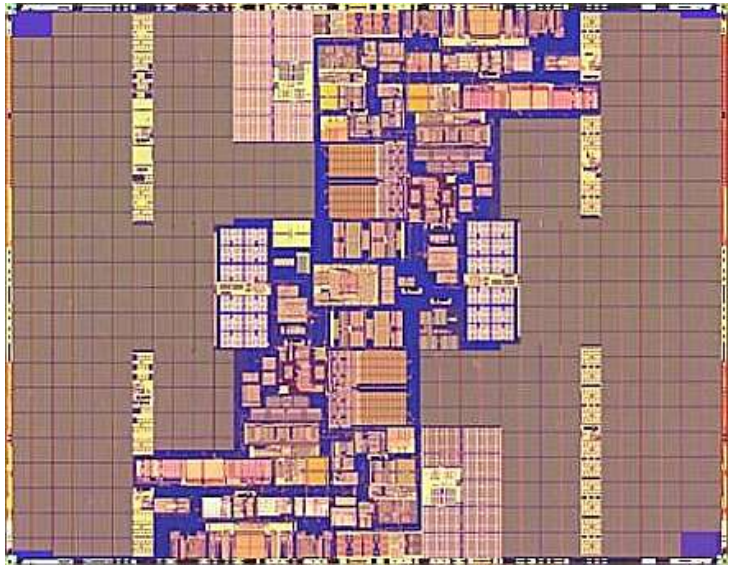
Hiérarchie mémoire



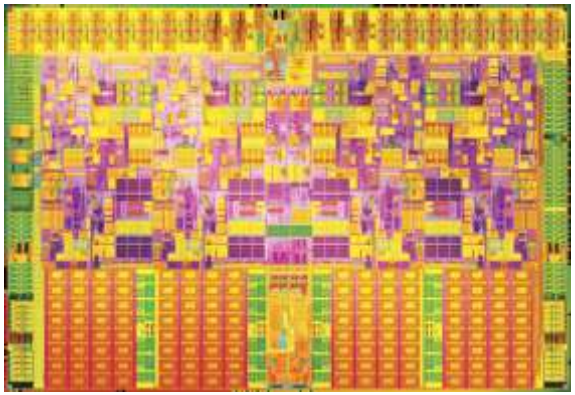
Cache

- Localité temporelle ou spatiale des données
- Les caches sont des éléments matériels complexes.
  - Cette complexité est cachée au programmeur.
- Par contre il existe des règles de bonne usage des caches
  - Il faut travailler sur la réutilisation des données
  - Leur localité
- Le programmeur peut lui même (à avoir) gérer la cohérence de cache
  - “Explicit cache control”
  - Utiliser par les compilateur dans les phases d’optimisation
  - Exemple d’instruction ou principe :
    - prefetching
    - “non temporal stores” ou “streaming stores”
    - fence
    - flush

- Design parameters
  - Capacity (size)
  - Line size
    - Banking
  - Coherency
    - Protocol (Ex. MESI), “snooping”
  - Associativity
    - Direct-mapped
    - Set-associative
    - Fully associative
  - Block replacement policy
    - LRU, LFU, FIFO, random
  - Write policy
    - Write-back, write-through (write buffer)
    - Allocate-on-write-miss policy
  - Victim buffer
  - Cache unification
  - Prefetching



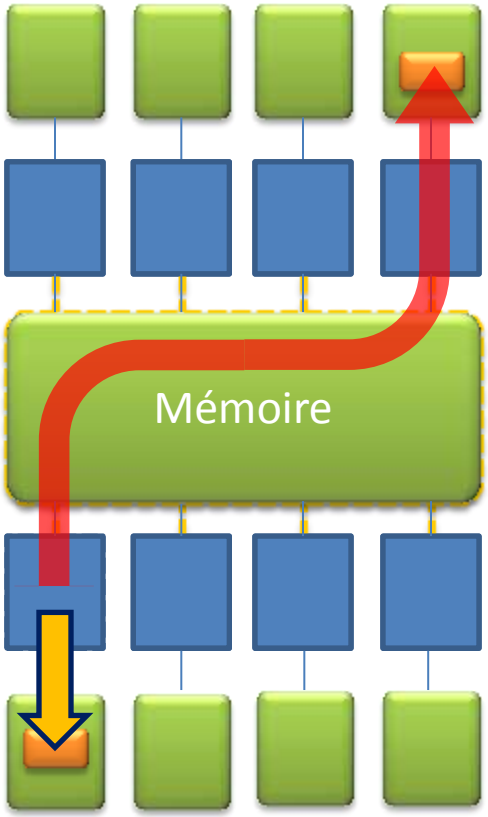
Intel IA64 Montecito (1.6 GHz)



Intel Xeon Nehalem (3.2 GHz)

Accès mémoire (intra-noeud) dans un environnement multiprocesseurs

- SMP: Symmetric Multi-Processing
  - Les accès depuis tous les processeurs se font au travers du même bus, avec la même distance
  - Scalabilité réduite …
- (cc-)NUMA: Non-Uniform Memory Access
  - Les processeurs sont regroupés en noeuds et il existe une hiérarchisation des accès mémoire:
    - Accès locaux, rapides (contrôleur mémoire attaché)
    - Accès distants (aux autres noeuds)
  - Améliore considérablement la bande passante globale du système (parallélisation) au détriment (généralement) de la latence
  - CC: Cache Coherent

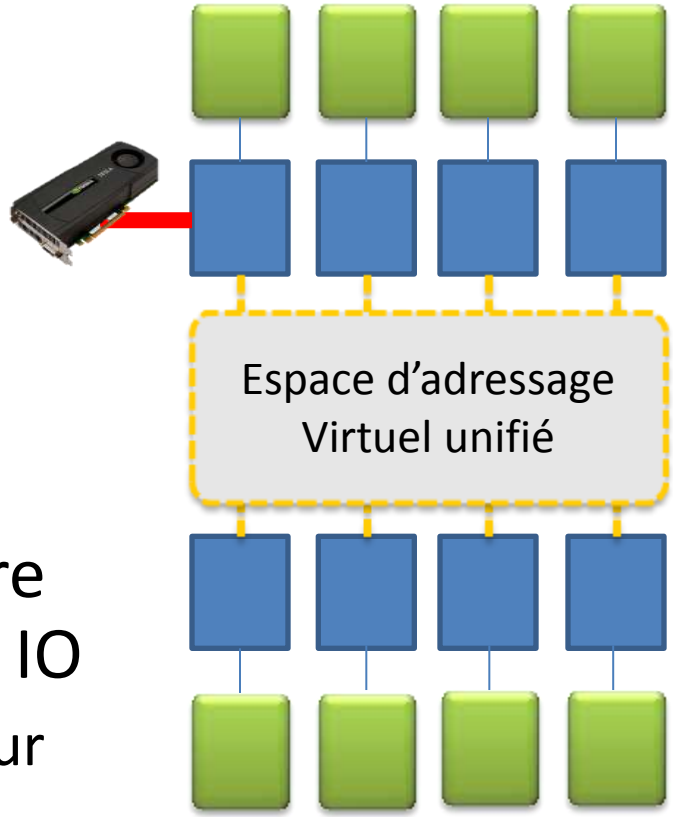


- Pour les processeurs multicoeurs actuels, un noeud NUMA isolé = SMP
- Linux supporte le mode SMP depuis sa version 2.0 (1996)

Accès mémoire (intra-noeud) dans un environnement multiprocesseurs

- La principale difficulté des architectures multiprocesseurs est la gestion de la cohérence des données
  - Protocole de cohérence (MESIF)
  - Opération atomique
  - Barrières …

  ⇒ **Modèle programmatif**
- Le “phénomène” NUMA peut être etendu aux accès aux ressources IO
  - En particuliers aux périphériques sur le bus PCIe
    - Réseau, Interconnect
    - Accélérateurs et coproceseurs …

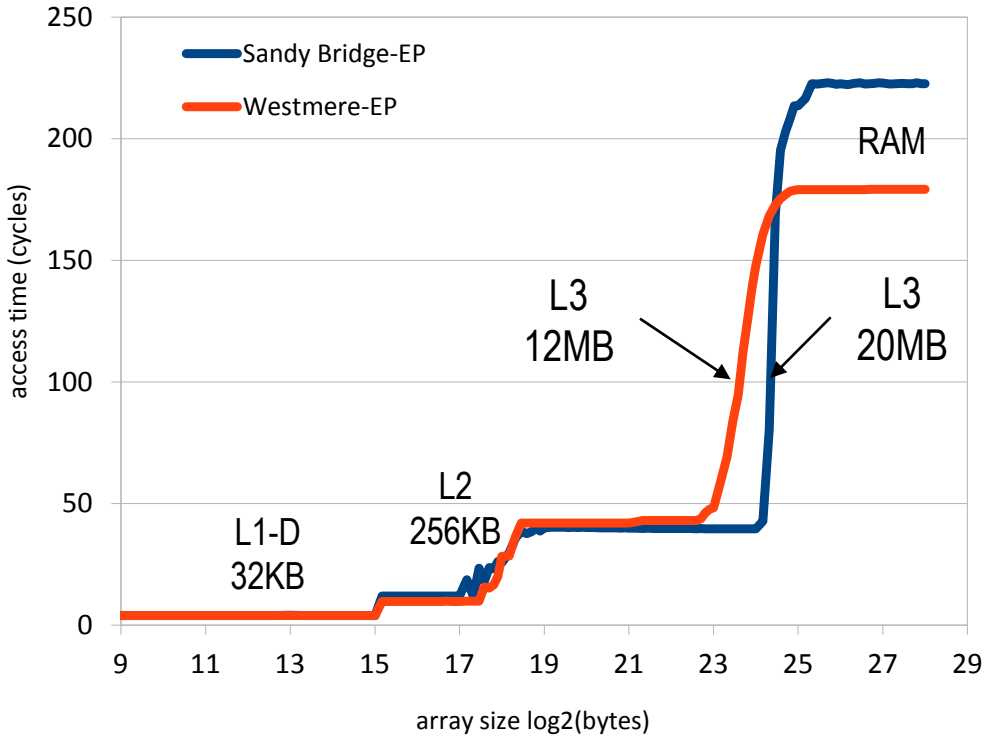


Débit mémoire (bande passante)

- **Estimation du débit théorique**
  - Paramètres:
    - Technologie et vitesse des barrettes (GT/sec) : 800, 1066, 1333, 1600 …
    - Nombre de canaux mémoire
    - Efficacité du contrôleur mémoire (50 à 80%)
    - Règles de population des slots mémoires

- **Mesures**
  - Compteurs Hardware: “LA” véritable mesure …
  - `Cachebench` (`llcbench`)
    - Mesure de la bande passante pour différentes tailles de tableaux (256 bytes à plusieurs dizaines de MiBytes) et différents « pattern »
      - Découverte de la hiérarchie mémoire
    - http://icl.cs.utk.edu/projects/llcbench/
  - Mc Calpin (virginia) `Stream` benchmark
    - La référence

Latences



	Latence (cycles)		
	Theo. (1)	Measured	
DCU	4-5	4	1.5 ns
MLC	12	12	4.6 ns
LLC	41 (avg)	40	15.5 ns
RAM		222	85 ns

	Latence (cycles)		
	NHM	WST	SNB
DCU	4	4	4
MLC	10	10	12
LLC	38	43	40

- Benchmarks pertinents:
  - `lat_mem_rd` (LMBench)
    - http://www.bitmover.com/lmbench/get_lmbench.html
- Compteurs hardwares

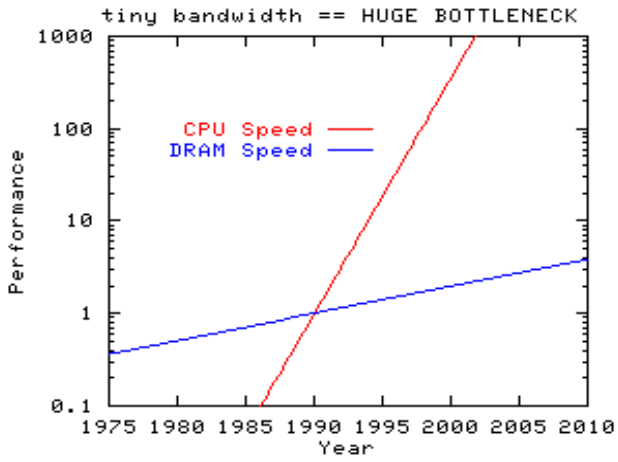
Débit mémoire | mesure, Stream benchmark

- http://www.cs.virginia.edu/stream/

« The STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels. »

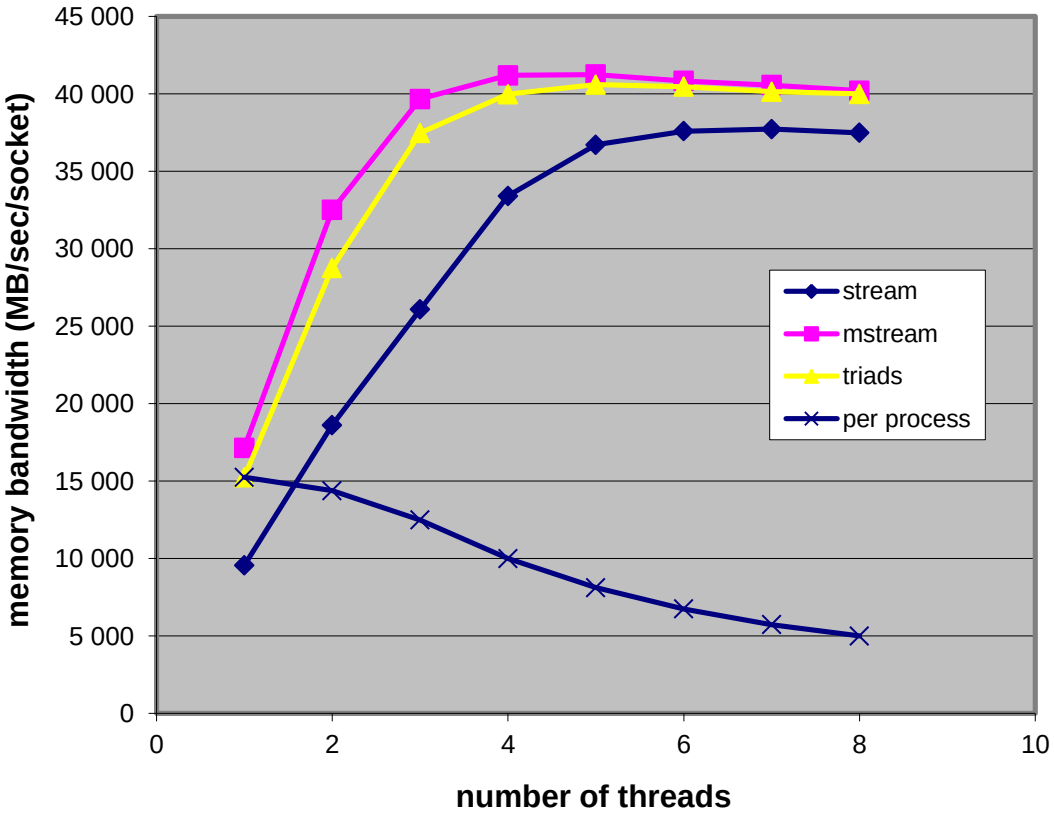
John McCalpin

- Benchmark accidentel ! Développé pour comprendre les différences de performance entre deux architectures pour un code météo.
- Indispensable dans sa trousse-à-outil
- Accès mémoires séquentiels
  - Portage :
    →Aucune difficulté
    →Fortran ou C (OpenMP)
  - Exécution : aucune difficulté
  - Auto-vérifiant (petite taille)



Débit mémoire | mesure, Stream benchmark

Memory bandwith (socket related) in MiB/sec		
threads	**triads**	**per process**
8	40 000	5 000
7	40 150	5 736
6	40 462	6 744
5	40 600	8 120
4	39 977	9 994
3	37 480	12 493
2	28 760	14 380
1	15 250	15 250



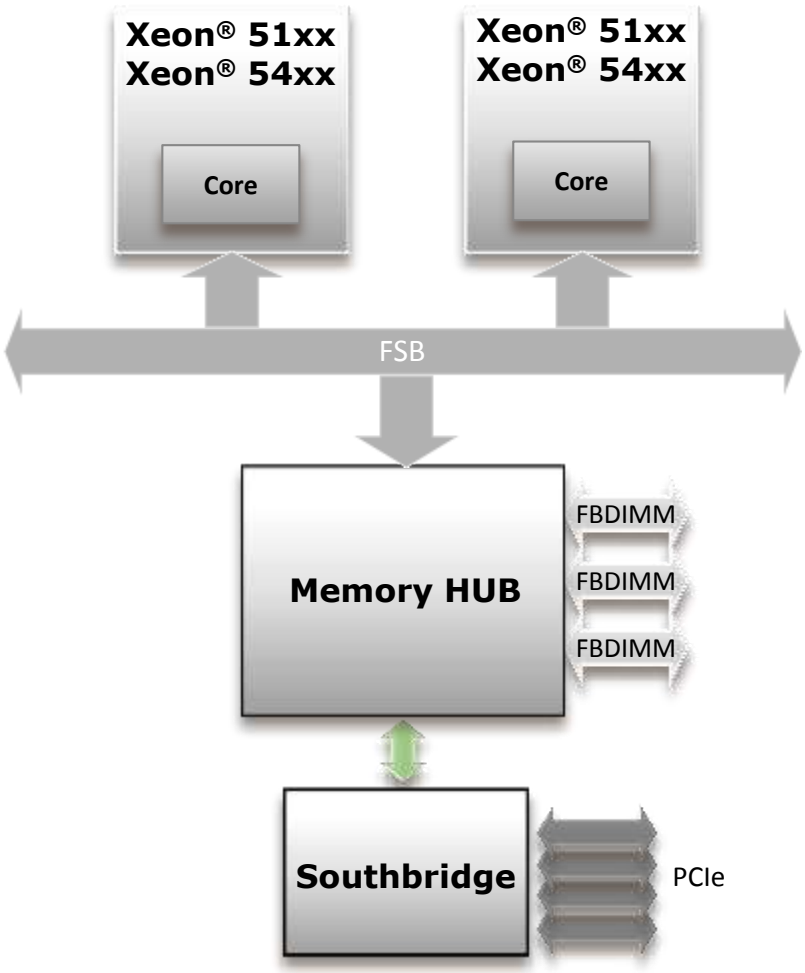
Intel ® Xeon ® “Sandy Bridge” E5-2690

Topologie NUMA

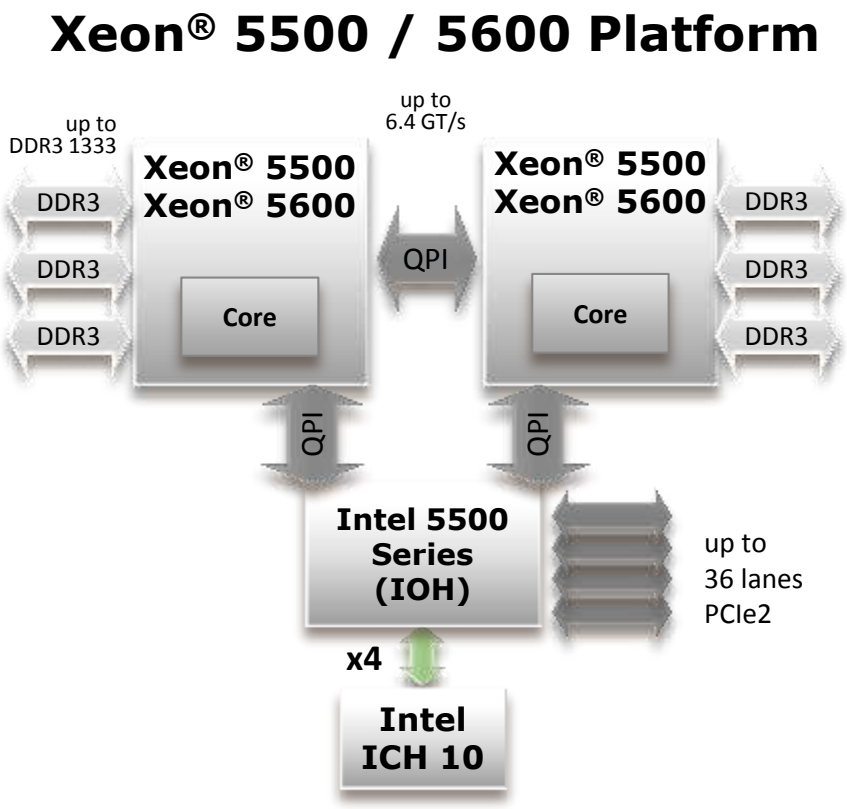
- La topologie du noeud de calcul est impactante au même titre que la topologie globale du calculateur

- Quelle topologie ? Quels impacts ?
- Comment connaître cette topologie ?
- Comment faire au mieux ?

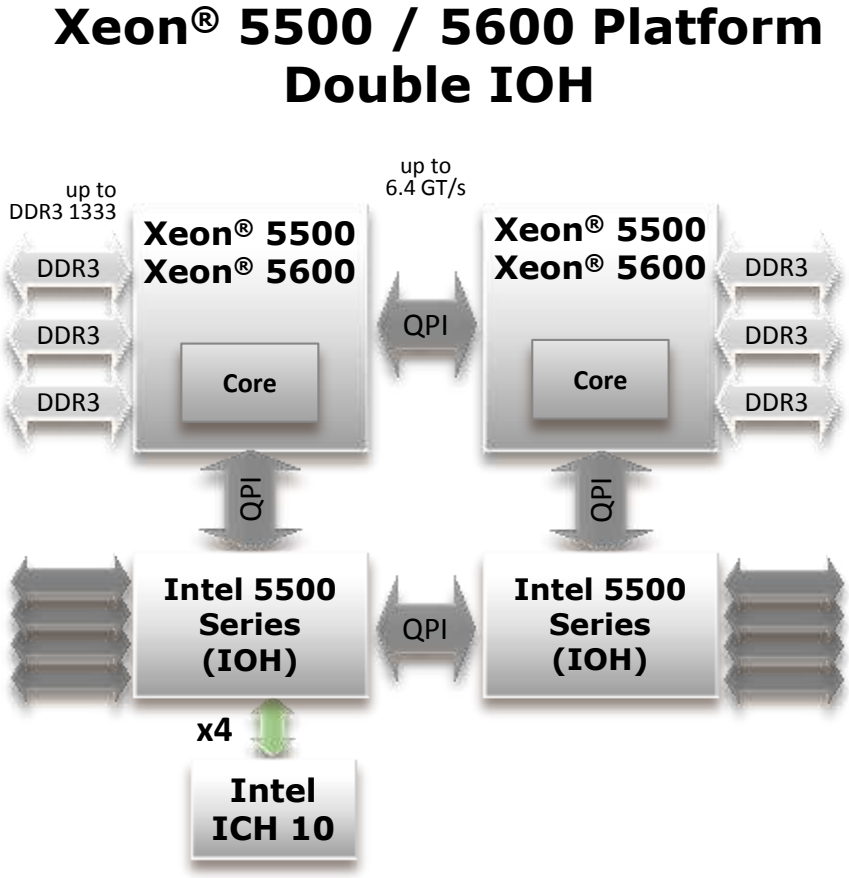
Configuration non-NUMA avec FSB



Configuration NUMA

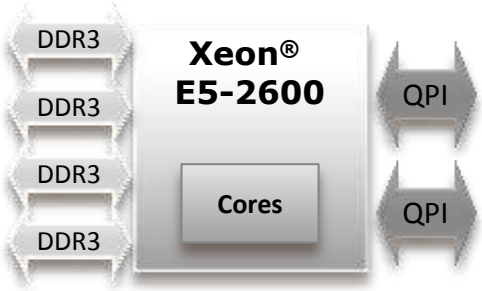


Configuration NUMA



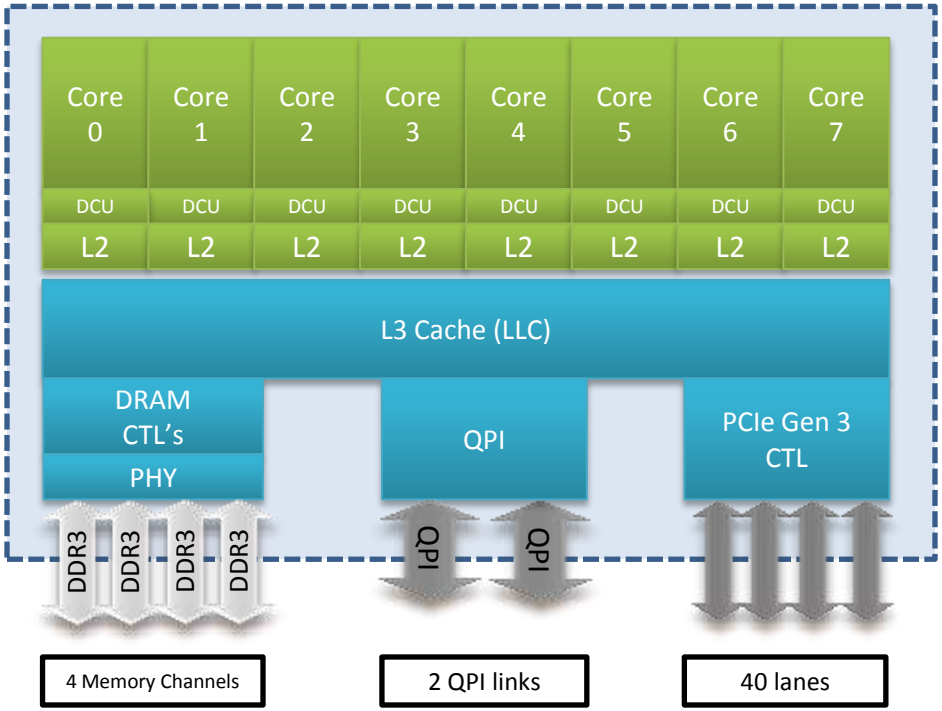
Configuration NUMA

Intel “Sandy Bridge”

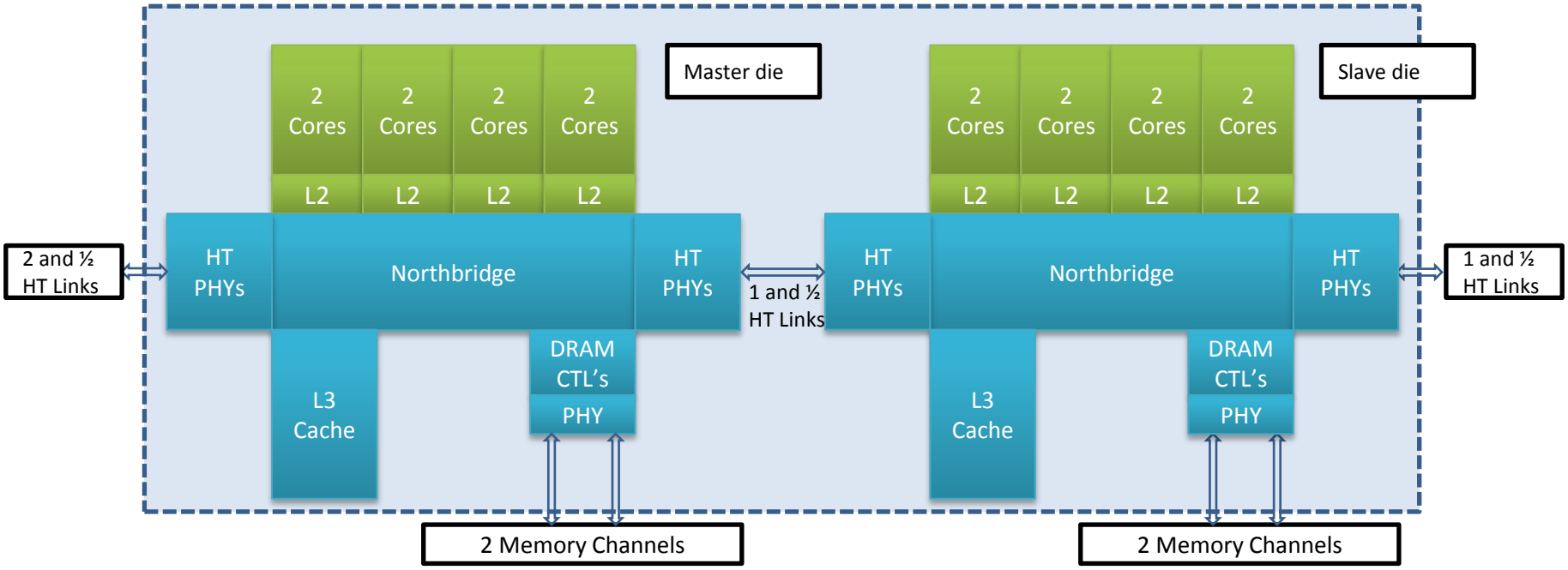


AMD “Interlagos”

Architecture Intel ® Jaketown



Architecture AMD - G34

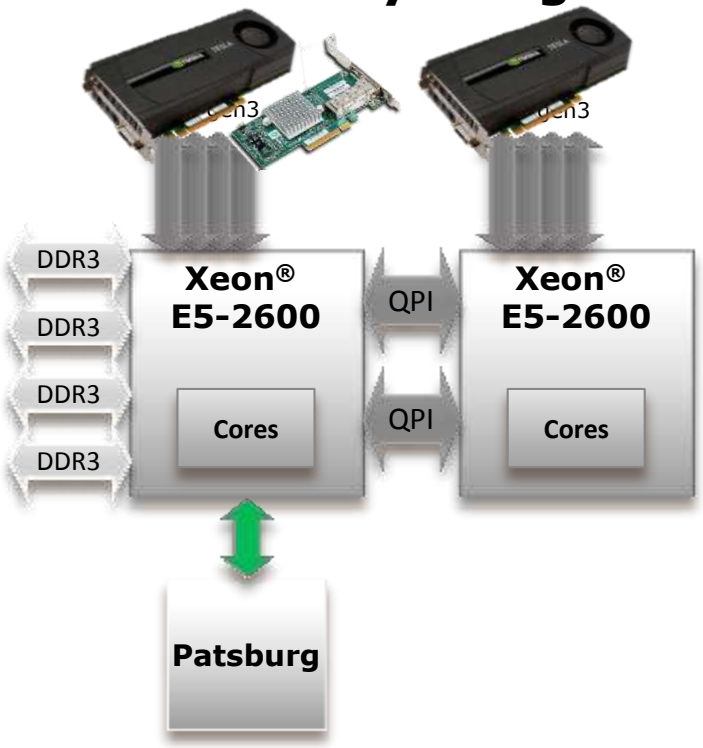


A Hierarchical system

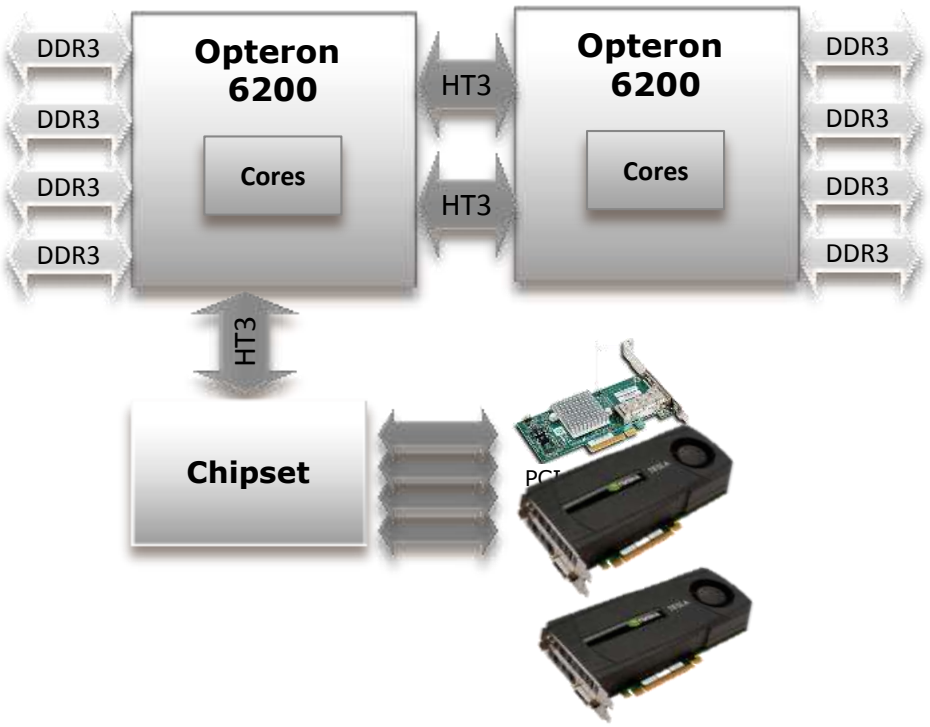
- Platforms are composed of 2 or 4 "G34" **processor packages**
- Each "G34" is composed of 2 **"Interlagos" chip** (or die)
- Each "Interlagos" die is composed of 4, 6 or 8 **"bullodozer" modules**
- Each "Bulldozer" comprise **2 integer cores** sharing **1 Flex FP (FPUnit)**

Configuration NUMA

Intel "Sandy Bridge"

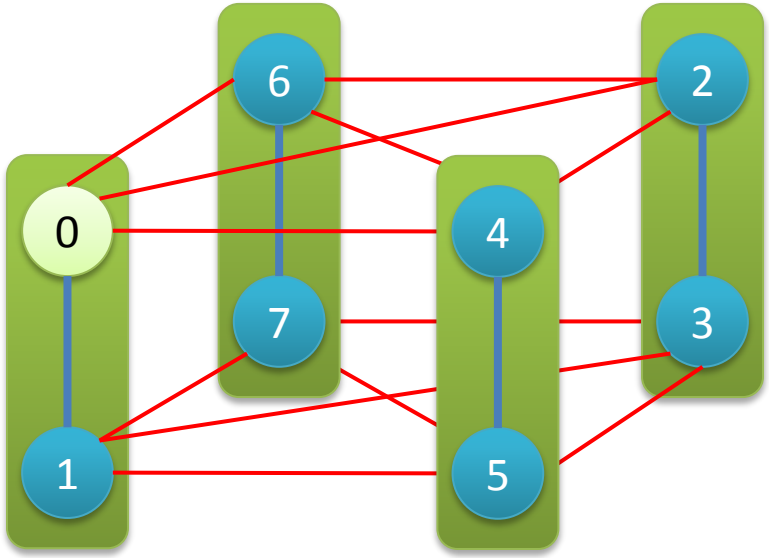


AMD "Interlagos"



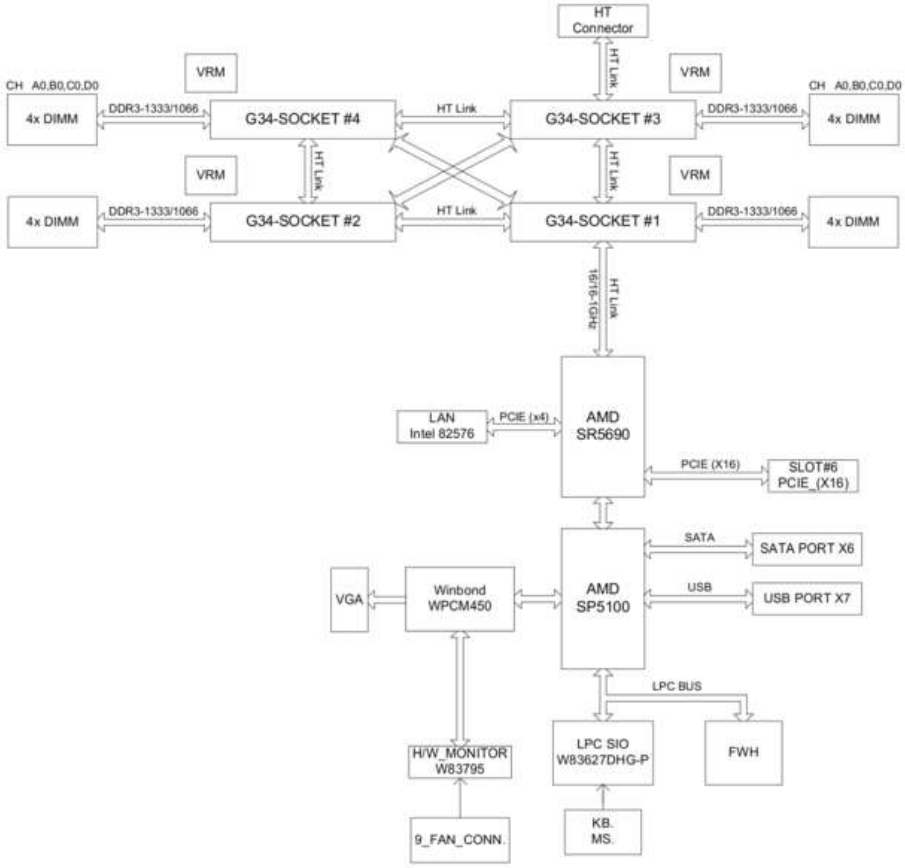
AMD 4P platform architecture

- Débit mémoire mesuré depuis le nœud 0

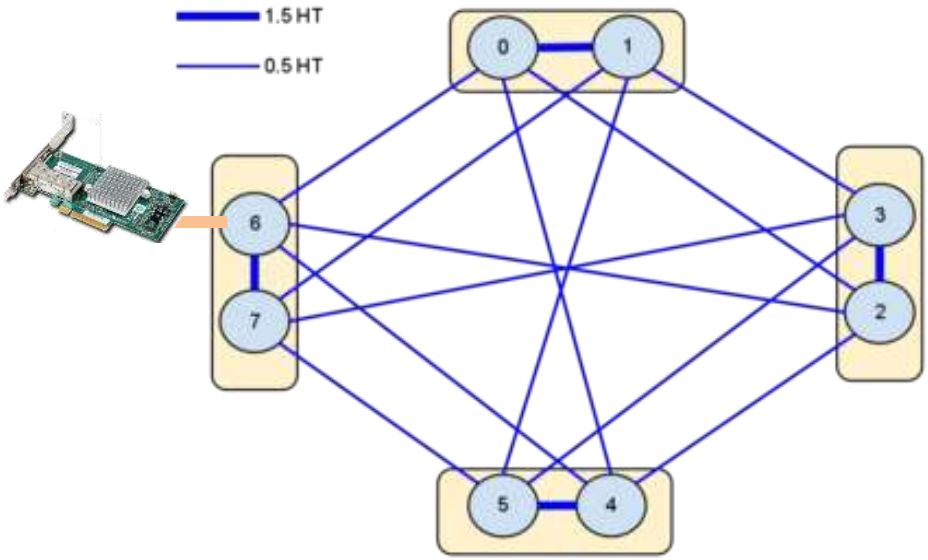


0	9 800
1	5 600
2	4 950
3	4 180
4	4 100
5	2 828
6	4 000
7	2 756

Performance InfiniBand sur G34-4P

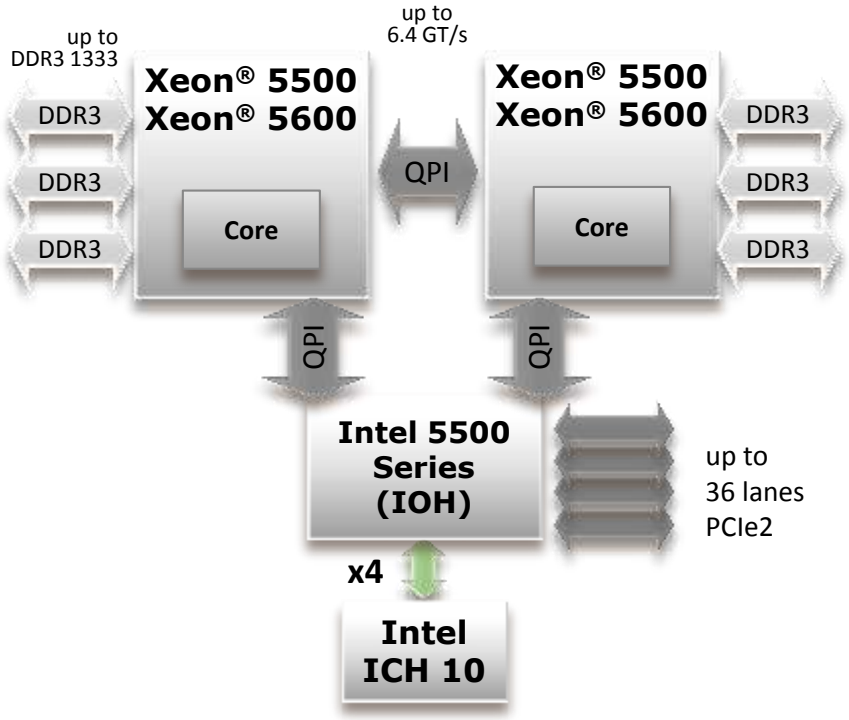


package	NUMA node	core	Latency (us)	BW (MB/sec)
0	0	1	1.99	2 275
	1	8	2.11	1 364
1	2	16	1.97	2 227
	3	24	2.12	1 348
2	4	32	1.96	2 878
	5	40	2.11	1 971
3	6	48	1.86	2 893
	7	56	1.95	2 887

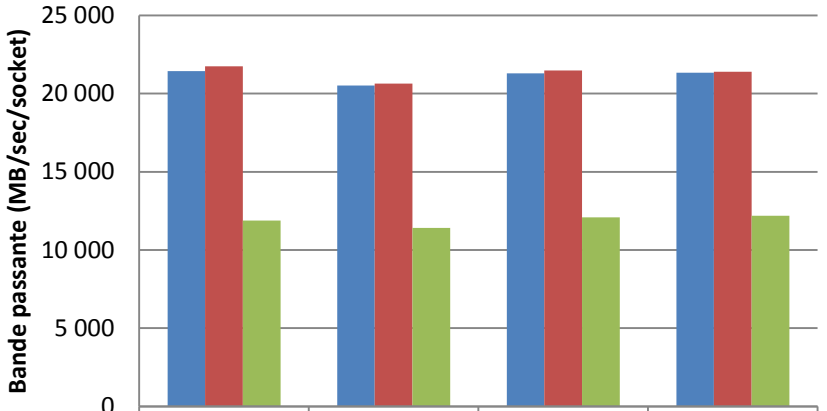


Effets “NUMA”

Xeon® 5500 / 5600 Platform



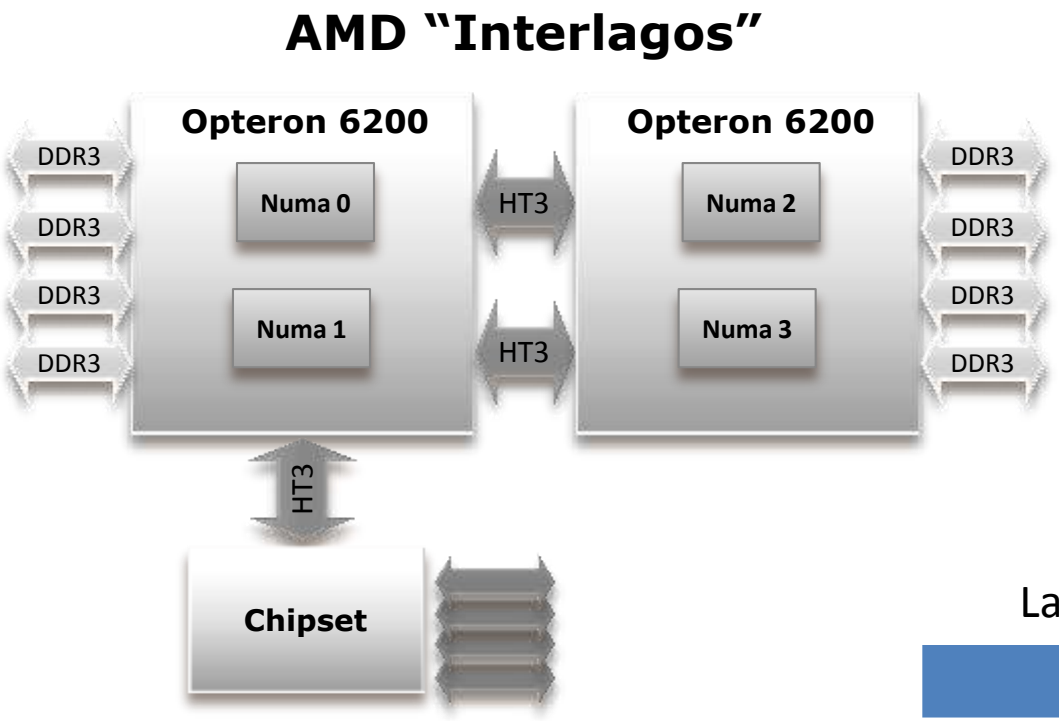
Memory Stream (X5660, 1333 MHz)



	Copy	Scale	Add	Triad
0-0	21 444	20 511	21 292	21 338
1-1	21 749	20 644	21 482	21 385
0-1	11 891	11 408	12 080	12 192

- Débit théorique (/socket):
  - 3x1333x(64/8) = 32GB/sec
  - IMC eff. : 21.7/32 ≈ 68%
- Effet NUMA ≈ 1:2

Effets “NUMA”

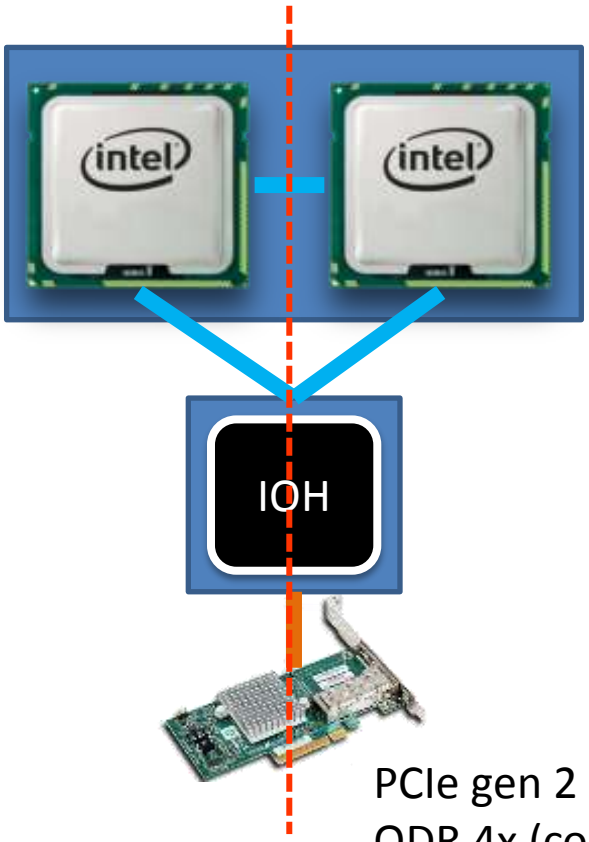


Latences d’accès (cycles proc.)

	0	1	2	3
0	95	151	163	155
1	151	95	155	161
2	163	155	95	150
3	155	161	151	96

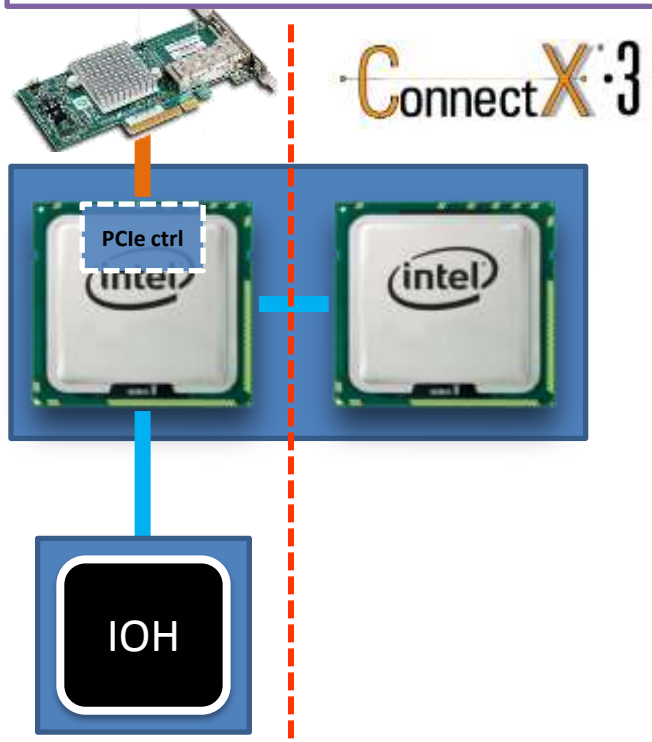
Effets « NUMA IOs »

Architecture Intel ® Nehalem



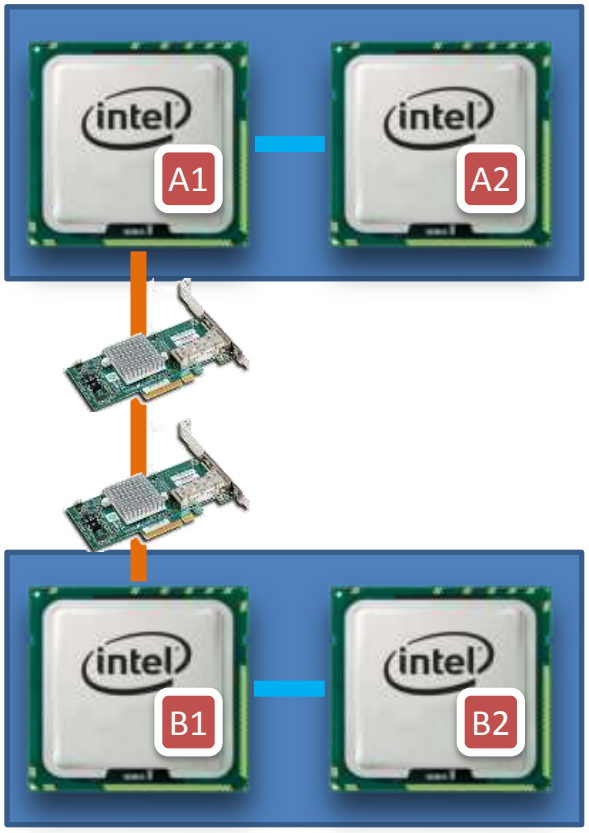
PCIe gen 2
QDR 4x (connectX-2)

Architecture Intel ® Sandy Bridge



PCIe gen 3
FDR 4x (connectX-3)

Effet « NUMA IOs »



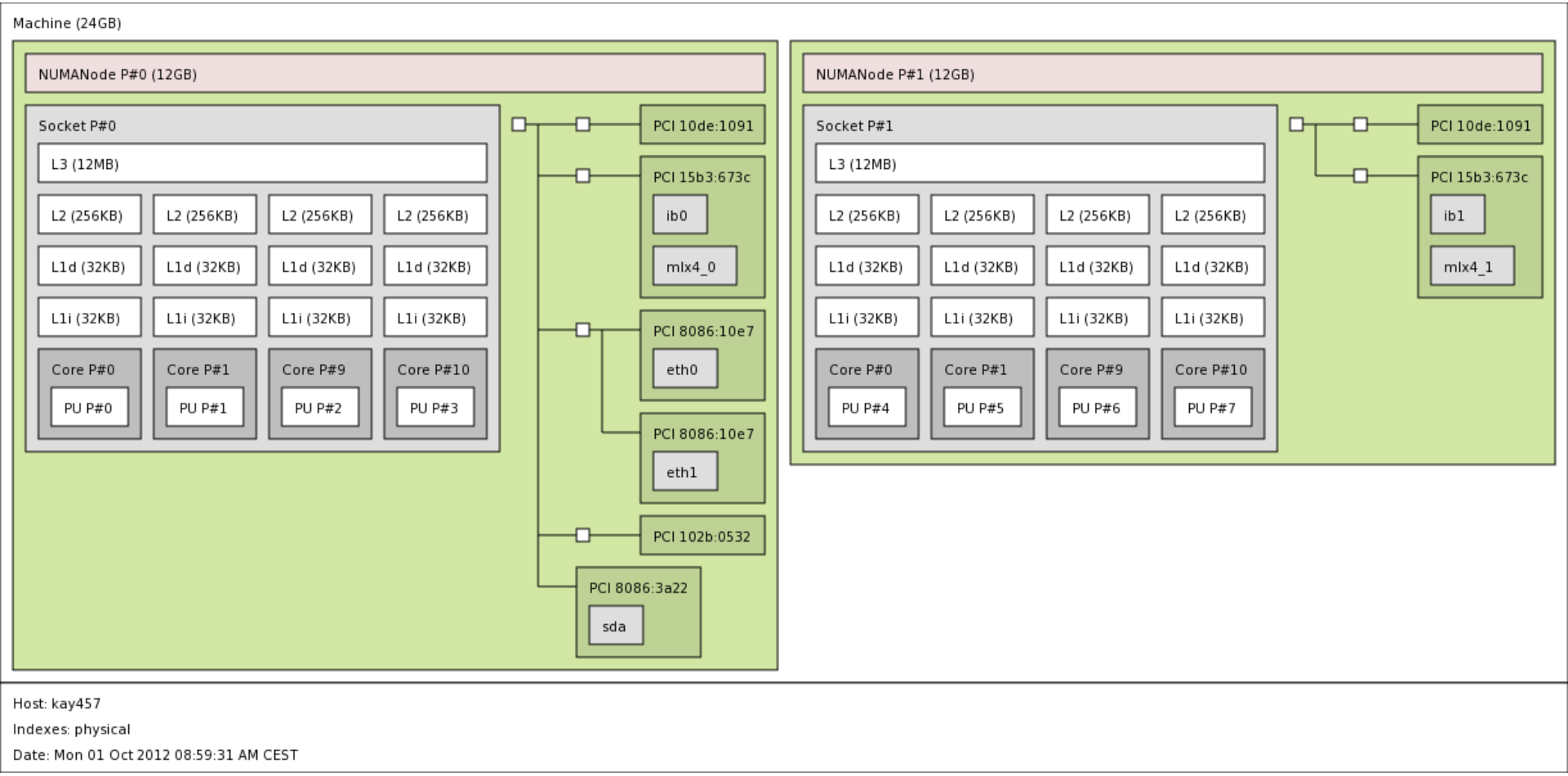
- Rappel: Westmere, 1.44 usec

Sandy Bridge FDR – Conf1		Target	
		B1	B2
Initiator	A1	**1.16**	1.63 (+40%)
	A2	1.63	**2.11** (+82%)

Sandy Bridge QDR – openmp 1.4.4		Target	
		B1	B2
Initiator	A1	**1.02**	1.25 (+22%)
	A2	1.25	**1.97** (+93%)

Topologie des machines

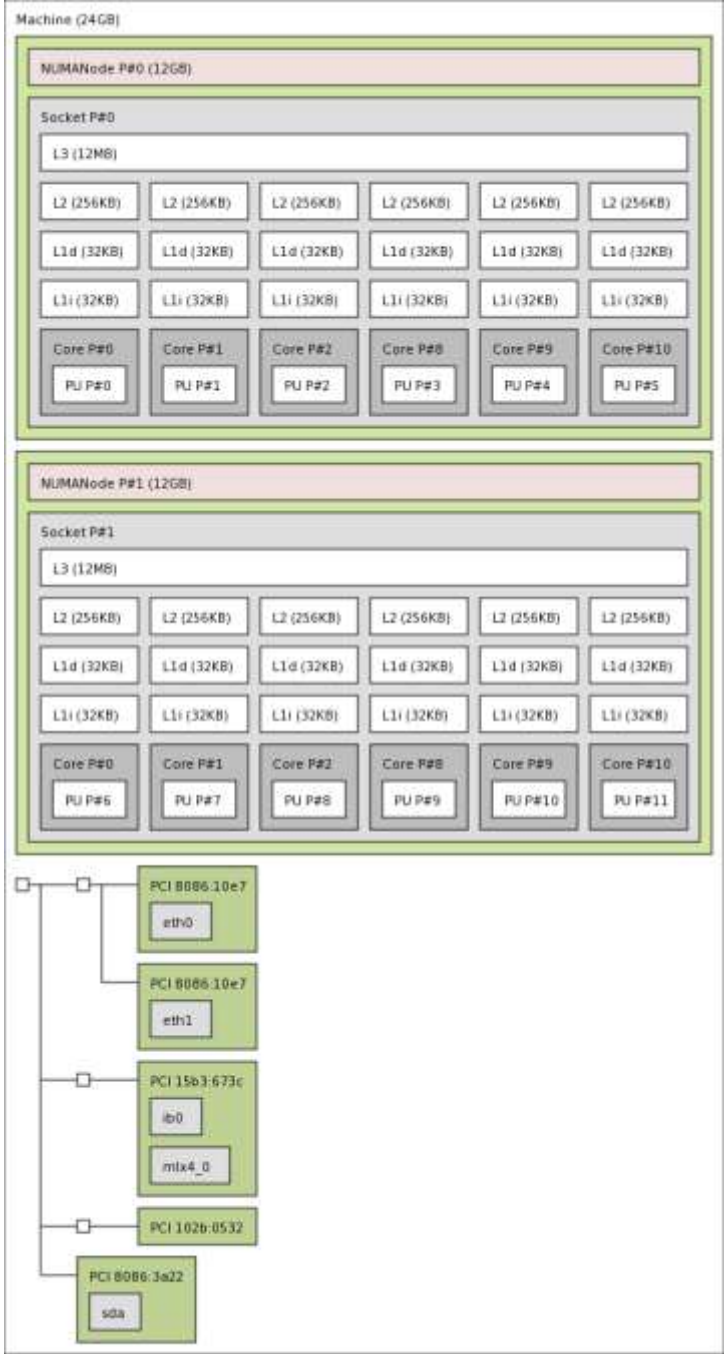
- **Hiérarchie des “CPUs” et mémoire**
  - `cpuid`
    - Très bas niveau: instruction assembleur …
  - `/proc` et `/sys`
    - `/proc/cpuinfo`
    - `/sys/devices/system/{cpu,node}/`
  - `hwloc`
  - `numactl` …
  - `cpuinfo`
- **I/O**
  - `lspci`
  - `hwloc`

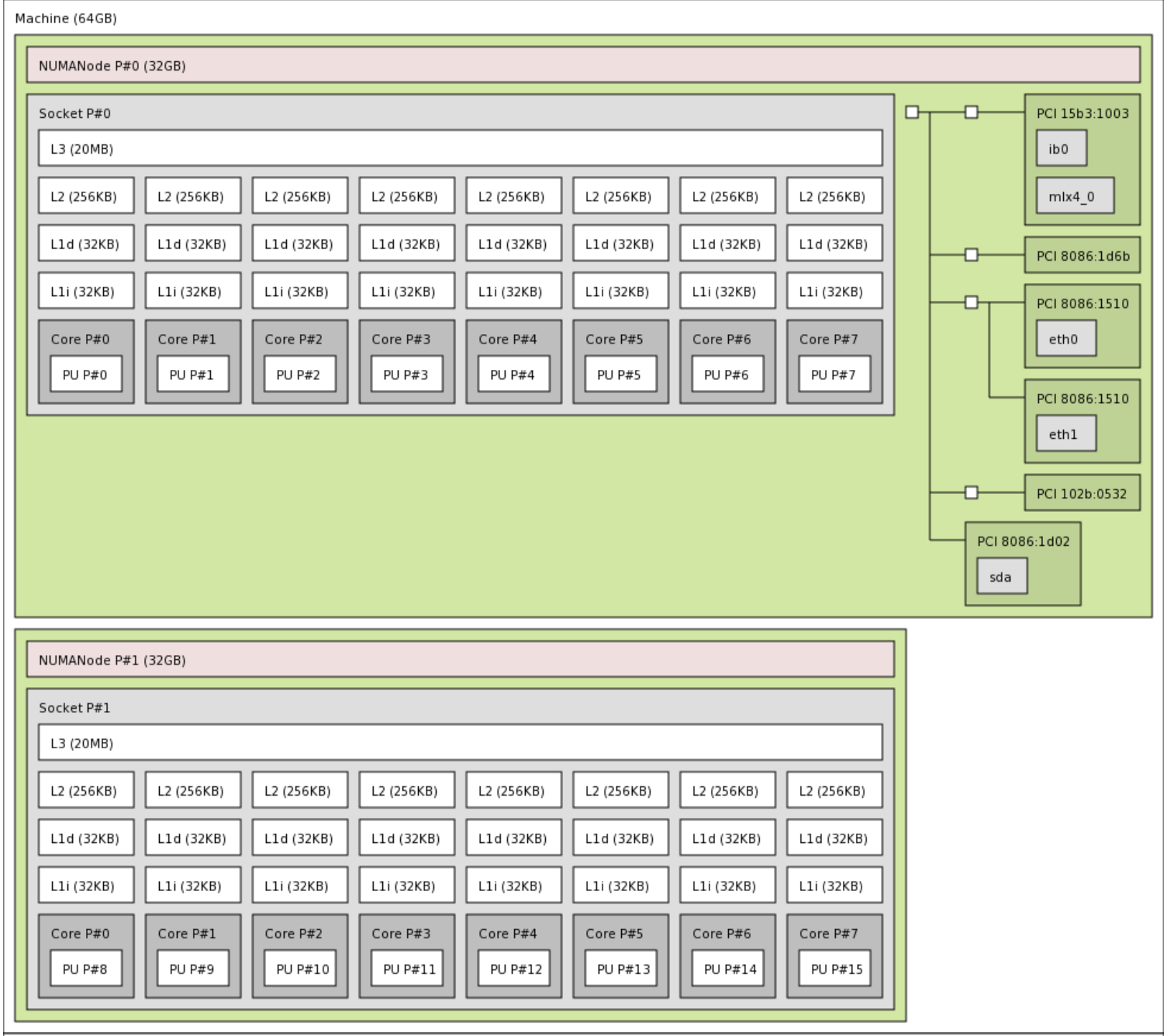


Host: kay457

Indexes: physical

Date: Mon 01 Oct 2012 08:59:31 AM CEST


Machine (24GB)
NUMANode P#0 (12GB)
Socket P#0
L3 (12MB)
L2 (256KB)
L1d (32KB)
L1i (32KB)
Core P#0
PU P#0
Core P#1
PU P#1
Core P#2
PU P#2
Core P#8
PU P#3
Core P#9
PU P#4
Core P#10
PU P#5
NUMANode P#1 (12GB)
Socket P#1
Core P#0
PU P#6
Core P#1
PU P#7
Core P#2
PU P#8
Core P#8
PU P#9
Core P#9
PU P#10
Core P#10
PU P#11
PCI 8086:10e7
eth0
PCI 8086:10e7
eth1
PCI 15b3:673c
ib0
mlx4_0
PCI 102b:0532
PCI 8086:3a22
sda
Host: kay374
Indexes: physical
Date: Mon 01 Oct 2012 09:29:16 AM CEST



Host: kay301

Indexes: physical

Date: Mon 01 Oct 2012 09:29:46 AM CEST

Alignement des données

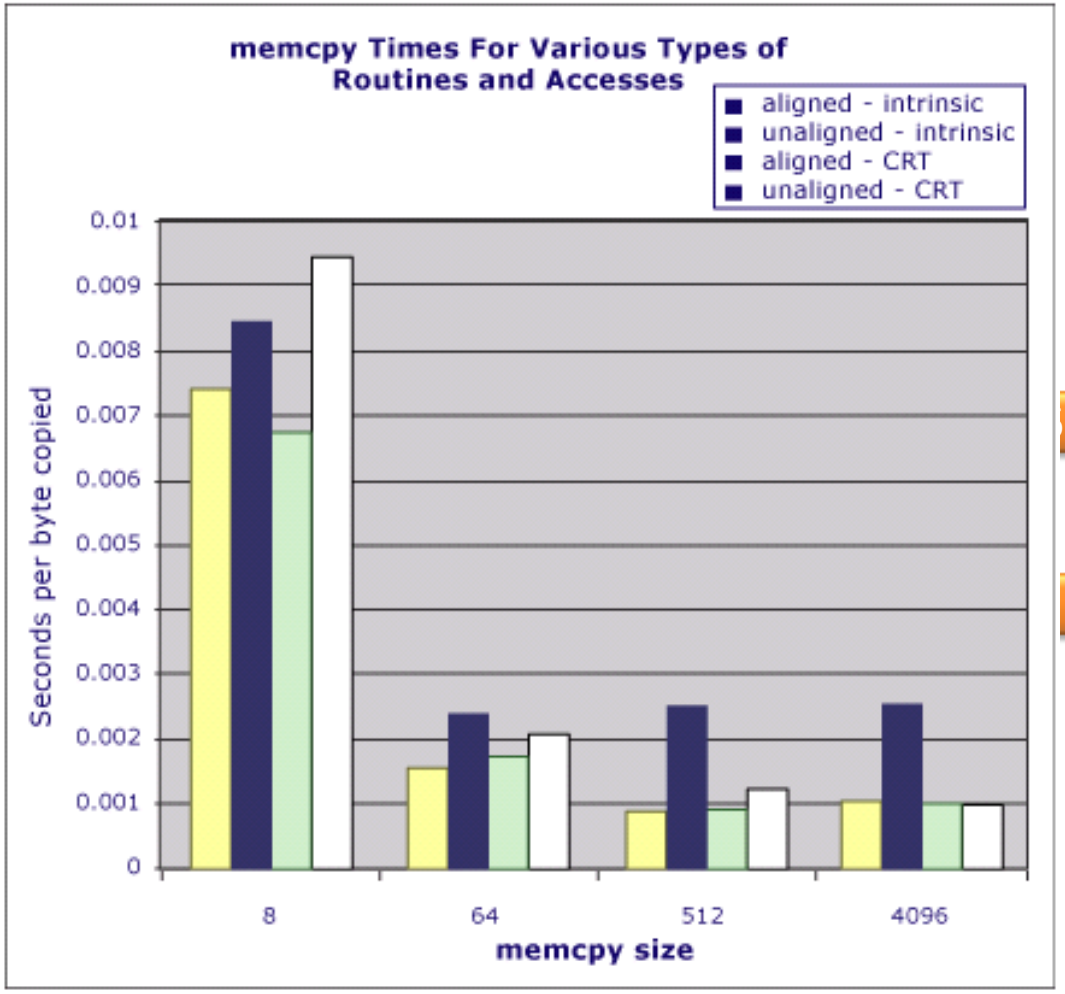
Lignes de

Ali

Non-ali

Ali

Non-ali



16

Placement des threads et optimisations des accès intranoeuds

Le chef d'orchestre: le noyau

- Le noyau est informé très tôt dans le processus de boot de la topologie de la machine par l'intermédiaire de l'ACPI et en particulier des tables:
  - SRAT : Static Resource Affinity Table
  - SLIT : System Locality Information Table
  - Existence de 'proximity domain'

```
Linux version 2.6.32-220.7.1.bl6.Bull.28.8.x86_64 (efix@atlas.frec.bull.fr) (gcc version 4.4.6
20110731 (Bull 4.4.6-3) (GCC) ) #1 SMP Fri Apr 27 13:48:52 CEST 2012
…
SRAT: PXM 0 -> APIC 0 -> Node 0
SRAT: PXM 0 -> APIC 2 -> Node 0
SRAT: PXM 0 -> APIC 4 -> Node 0
SRAT: PXM 0 -> APIC 6 -> Node 0
SRAT: PXM 1 -> APIC 16 -> Node 1
SRAT: PXM 1 -> APIC 18 -> Node 1
SRAT: PXM 1 -> APIC 20 -> Node 1
SRAT: PXM 1 -> APIC 22 -> Node 1
SRAT: Node 0 PXM 0 0-a0000
SRAT: Node 0 PXM 0 100000-c0000000
SRAT: Node 0 PXM 0 100000000-340000000
SRAT: Node 1 PXM 1 340000000-640000000
NUMA: Allocated memnodemap from 28040 - 34880
NUMA: Using 20 for the hash shift.
…
```

Le chef d'orchestre: le noyau

- Le noyau dispose d'un petit nombre d'appels systèmes lui permettant de gérer le placement/l'attachement des processus et la gestion mémoire
  - **`#include <sched.h>`**
    - `sched_setaffinity`
    - `sched_getaffinity`
    - `sched_getcpu` (glibc > 2.6)
  - **`#include <numaif.h>`**
    - `mbind`
    - `set_mempolicy`
- Existe des interfaces noyau de plus haut niveau (`cpuset`)

#define _GNU_SOURCE
#include <sched.h>

Fonction	Description	Remarque
sched_getaffinity sched_setaffinity	Définir et obtenir le masque d'affinité CPU d'un processus	CPU_CLR CPU_ISSET CPU_SET CPU_ZERO
sched_getcpu	Déterminer le processeur et le nœud NUMA sur lesquels le thread appelant est en cours d'exécution	

#include <numaif.h>

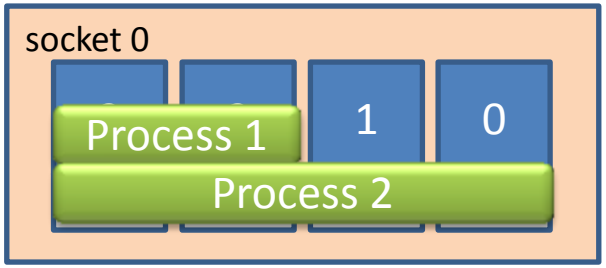
Fonction	Description	Remarque
mbind	Configurer la politique mémoire pour une zone de mémoire	MPOL_DEFAULT MPOL_BIND MPOL_INTERLEAVE MPOL_PREFERRED
set_mempolicy get_mempolicy	Configurer/Lire la politique de la mémoire NUMA par défaut pour un processus et ces fils	
move_pages	Déplacer des pages individuelles d'un processus sur un autre nœud	

Différente politique de gestion mémoire (NUMA)

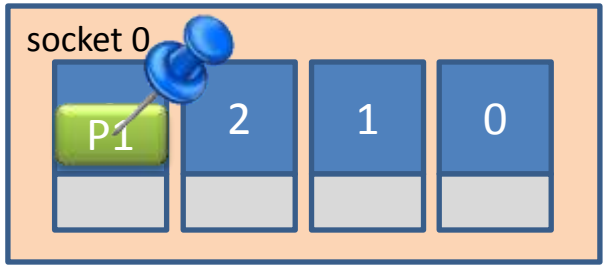
Politique	Description	FLAGS afférent
Default	Allocation sur le noeud local (là où le processus est en cours d'exécution)	MPOL_DEFAULT
Bind	Politique d'allocation stricte des pages mémoires sur un ensemble de noeuds.	MPOL_BIND
Interleave	Les pages sont distribuées	MPOL_INTERLEAVE
Preferred	Définit l'allocation prioritairement sur un noeud. Si ce n'est pas possible, les noeuds les plus proches seront mis à contribution (fallback)	MPOL_PREFERRED

Ordonnancement des processus

- **Placement**
  - action de restreindre l'enveloppe des processeurs éligibles lors de l'ordonnancement d'un processus
    ⇒ **notion de bitmask**



- **Attachement** (binding/pinning)
  - Action de restreindre cette enveloppe à un seul et unique processeur logique



- **Lors de la création d'un processus, le fils hérite du 'bitmask' du père**

- Pourquoi c'est important ?

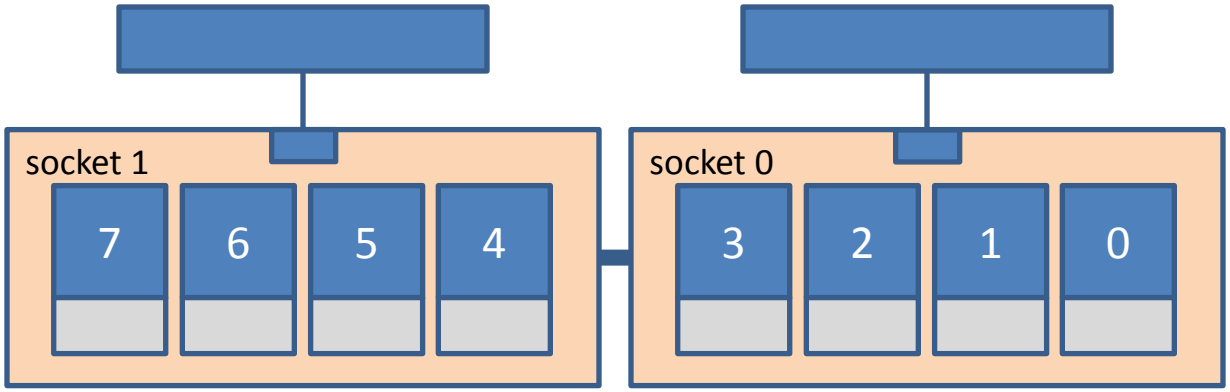
Assurer la localité des données

Minimiser les temps d'accès

Maximiser le débit mémoire

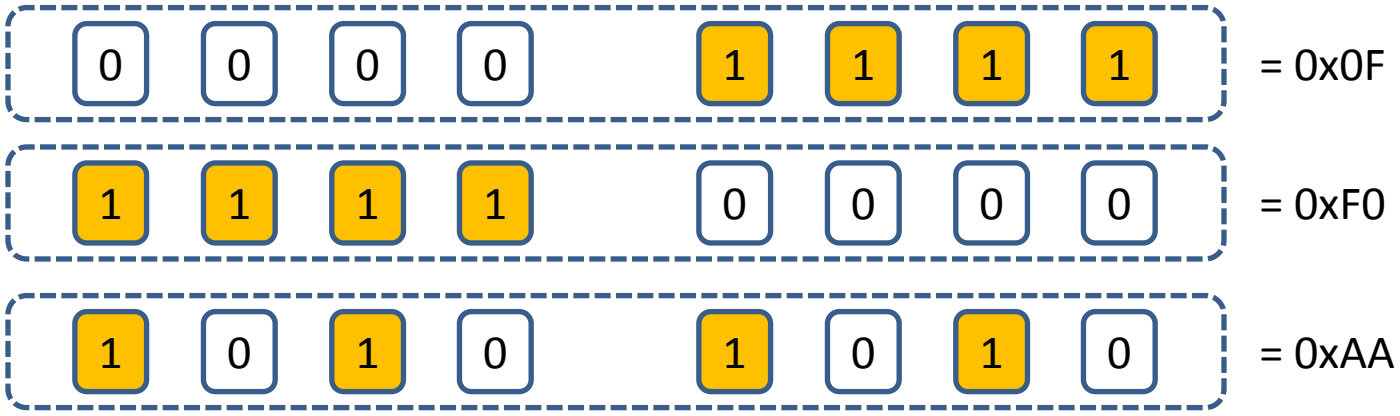
OPTIMISER

Bitmasking

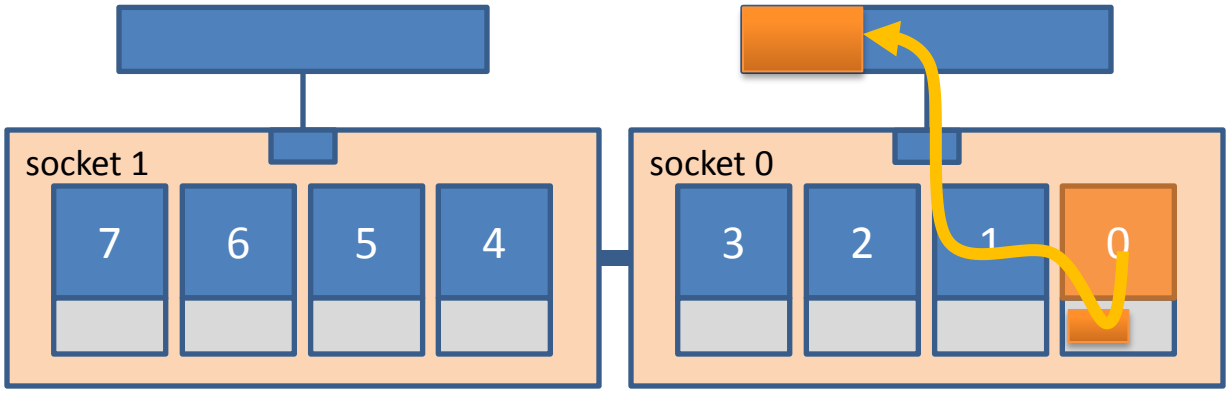


bitmask

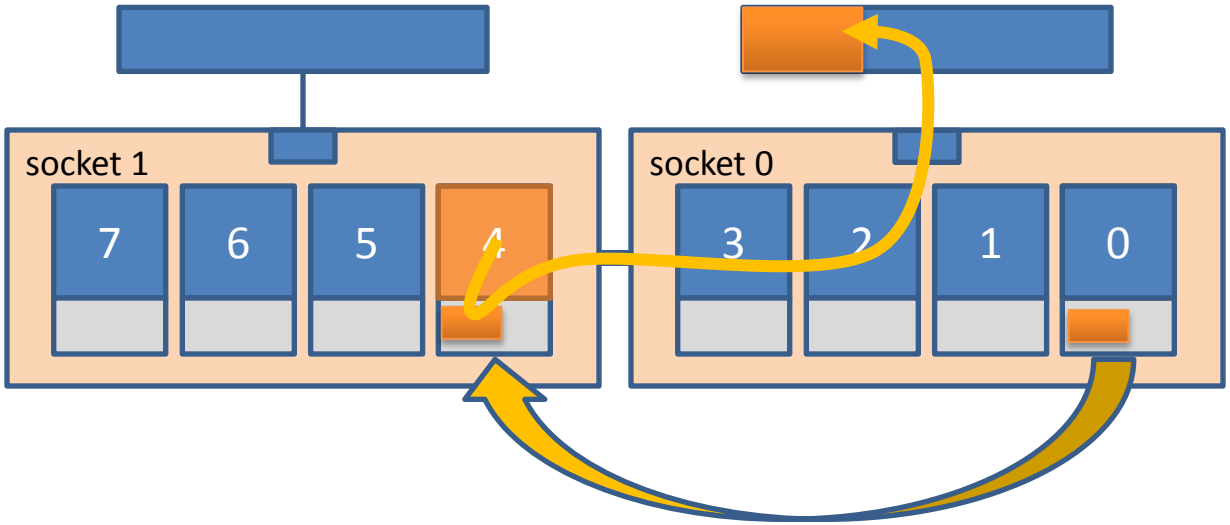
Exemples:



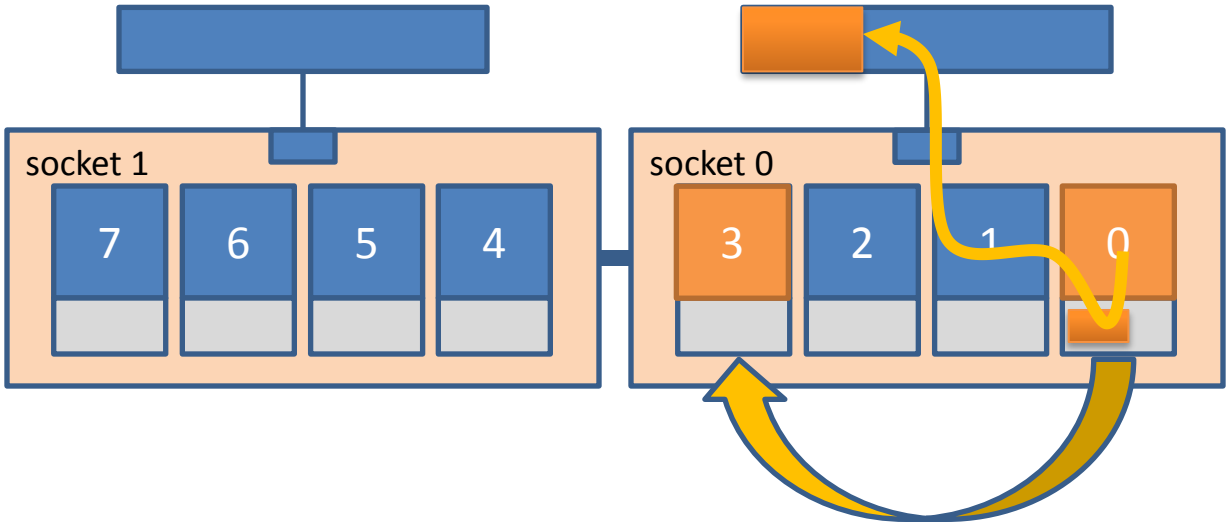
Switch de contexte



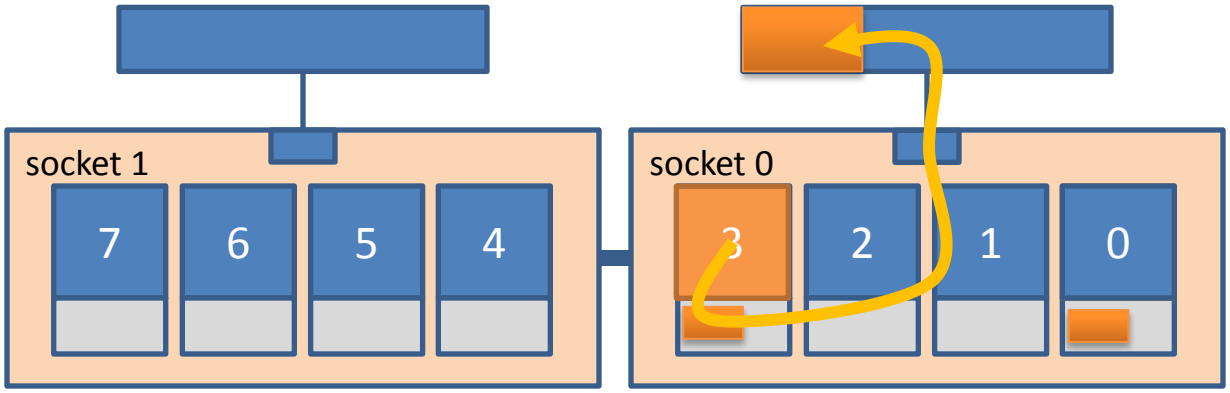
Switch de contexte



Switch de contexte



Switch de contexte



First touch policy

- Allocation se fait par page (4K généralement)
  - Les processeurs supportent aussi les hugepages
- Les allocations sont gérées à bas niveau de le kernel pour être optimisées
- Un `malloc` (voire un `calloc`) alloue la mémoire au niveau virtuelle pas physique
- La page doit être “touchée” pour être physiquement allouée
  - Permet l’overcommit !

First touch policy

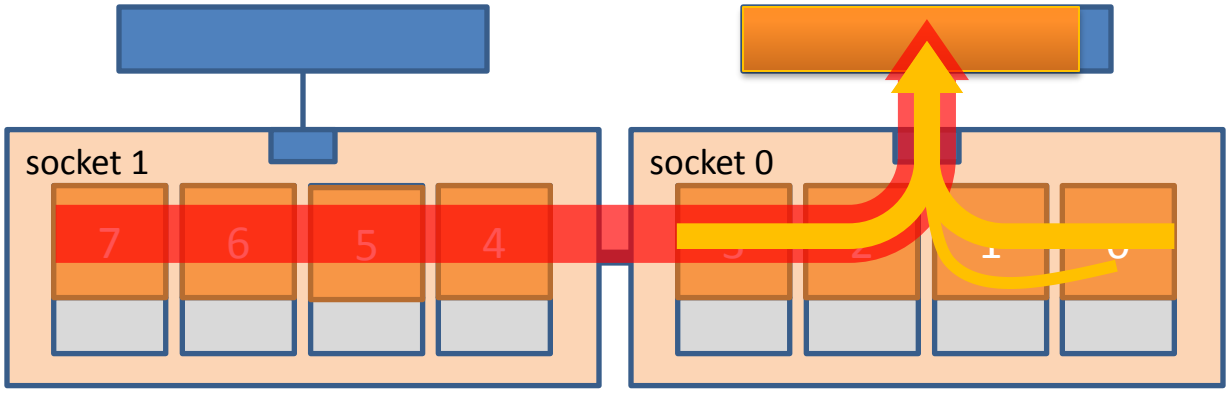
- Démonstration !

```
> ./fillmem 32g
NUMA support enabled. Max number of nodes: 2
32.00 Gbytes = 8388608 pages (page size=4096 byte)
Allocating memory...Done in 6e-06 sec
All requested memory have been allocated and touched.
Elapsed time        : 15.65 sec
  Node 00 - 32726 Mbytes, free 14962 Mbytes
  Node 01 - 32768 Mbytes, free 15607 Mbytes
```

- Le code alloue 32 GB de mémoire via un malloc en $6x10^{-6}$ sec
  - **Débit de** 32/ $6x10^{-6}$ = **5.3 10^6 GB/sec**

First touch policy

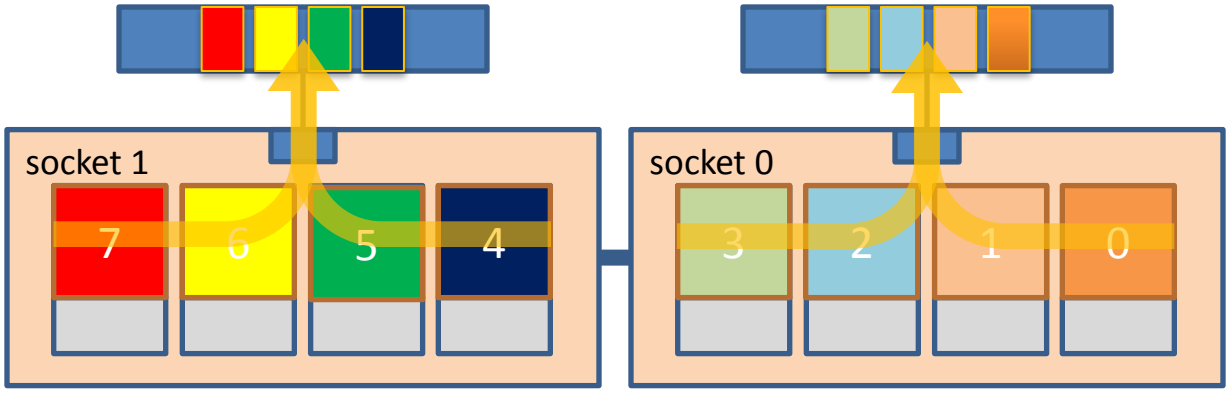
```
// initialisation des données
for(i=0; i<N; i++)
    for(j=0; j<M; j++) { … }

#pragma omp parallel for private(j)
for(i=0; i<N; i++)
    for(j=0; j<M; j++) { … }
```



La moitié des accès mémoires est distante !

First touch policy

```
#pragma omp parallel for private(j)
// initialisation des données
for(i=0; i<N; i++)
    for(j=0; j<M; j++) { … }

#pragma omp parallel for private(j)
for(i=0; i<N; i++)
    for(j=0; j<M; j++) { … }
```

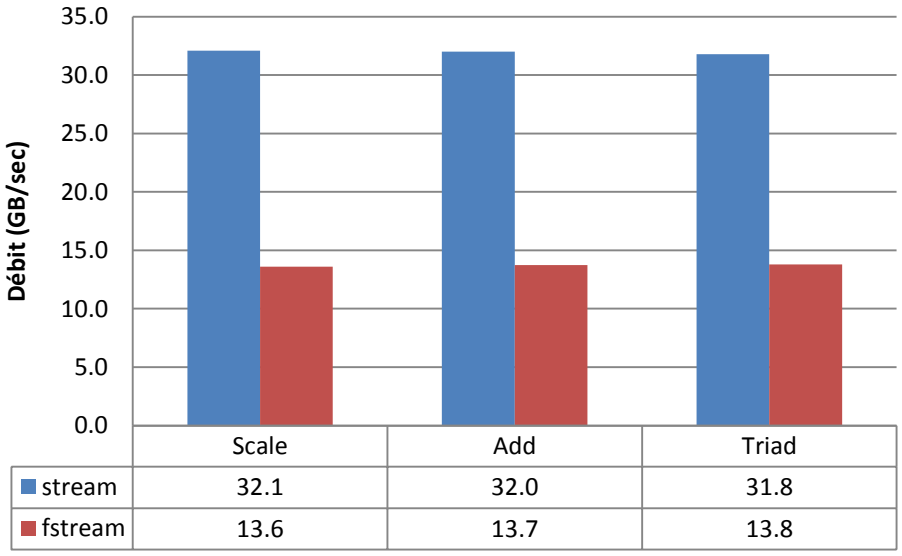


Tous les accès mémoires sont maintenant locaux !

First touch policy | exemple

- Un exemple:
  - Code stream modifié

```
…
/* Get initial value for system clock. */
#pragma omp parallel for
    for (j=0; j<N; j++) {
         a[j] = 1.0;
         b[j] = 2.0;
         c[j] = 0.0;
         }
…
```



	Scale	Add	Triad
stream	32.1	32.0	31.8
fstream	13.6	13.7	13.8

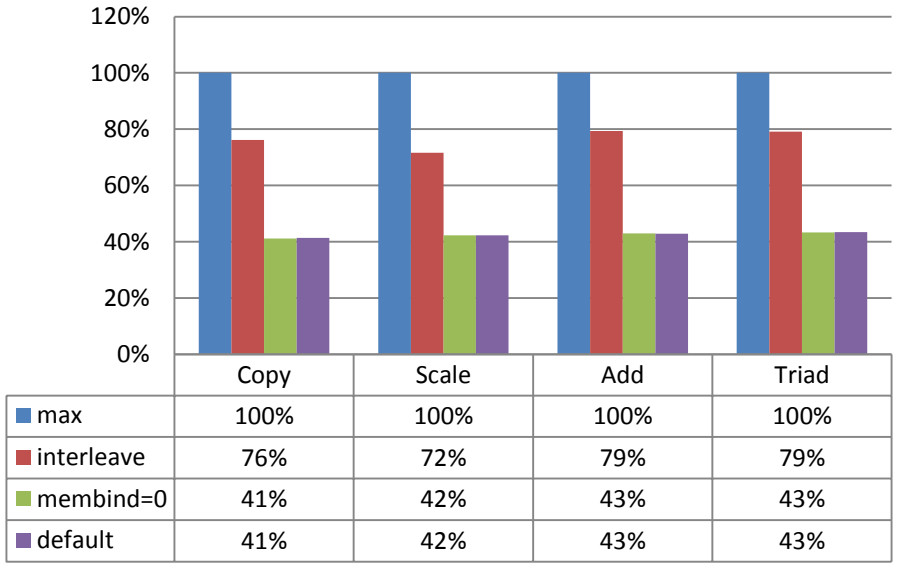
First touch policy | interleave

- Un exemple:
  - Code stream modifié
- Comment résoudre le problème:
  - Corriger le code si c'est possible (sources)
  - Si ce n'est pas possible, envisager l'**interleaving**
- Politique d'interleaving:
  - L'allocateur distribue les pages en "round-robin" sur les différents noeuds (ceux qu'on lui spécifie).
  - On augmente la probabilité de Hits
  - `numactl` est votre ami …

```
…
/* Get initial value for system clock. */
#pragma omp parallel for
    for (j=0; j<N; j++) {
         a[j] = 1.0;
         b[j] = 2.0;
         c[j] = 0.0;
         }
…
```

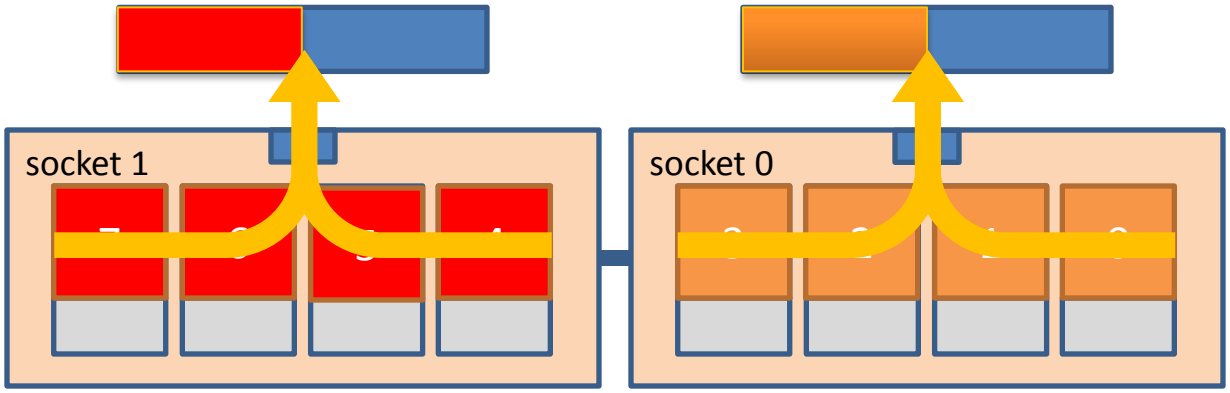
First touch policy | interleave

	numactl --interleave=0-1 ./fstream	numactl –membind=0 ./fstream	./fstream
	node0 node1 numa_hit 587631 587649 numa_miss 0 0 numa_foreign 0 0 interleave_hit **587581** **587574** local_node 491 **587199** other_node **587140** 450	node0 node1 numa_hit **1175180** 23 numa_miss 0 0 numa_foreign 0 0 interleave_hit 0 0 local_node 274 23 other_node **1174906** 0	node0 node1 numa_hit **1174947** 102 numa_miss 0 0 numa_foreign 0 0 interleave_hit 0 0 local_node **1174947** 102 other_node 0 0
Copy	18515	10015	10057
Scale	23000	13574	13593
Add	25392	13749	13732
Triad	25166	13794	13797

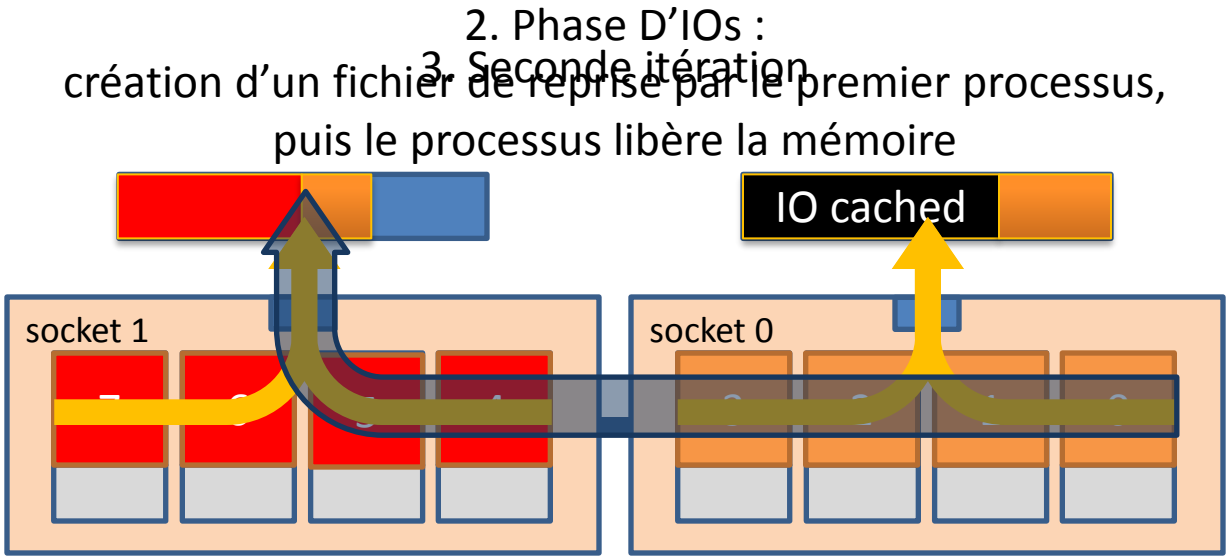


Gestion du cache IO et effets de bords

1. Première Itération



Gestion du cache IO et effets de bords



- Par défaut, linux alloue la mémoire sur le noeud local et sur un noeud distant si la mémoire locale est occupée par des entrées du cache IOs
- Cette politique est ajustable via l'entrée du noyau `/proc/sys/vm/zone_reclaim_mode` ou `vm.zone_reclaim_mode`.

Gestion du cache IO et effets de bords

- Zone Reclaim
  - Ajustement
    - À la volée:
      `echo 3 > /proc/sys/vm/zone_reclaim_mode`
    - Persistent
      - Ajouter dans le fichier `sysctl.conf`
        `vm.zone_reclaim_mode=3`
      - Rebooter ou forcer la reactivation des paramètres : `sysctl -p`
- On peu
  - `ech`
    Dif
    •
  - Pro
    •
    •

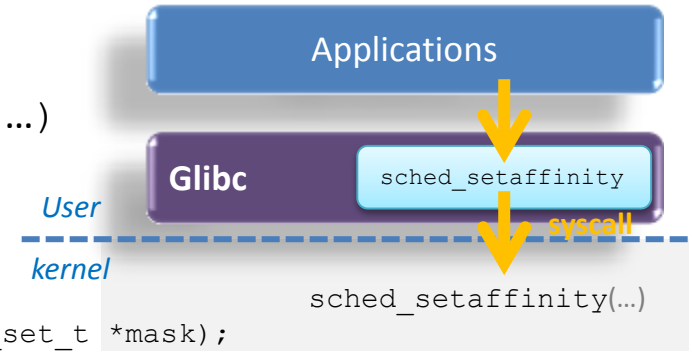
mode	Description
0	Désactive le ‘mode reclaim’. Généralement la politique par défaut dans beaucoup d’OS. A du sens sur les serveur de données …
1	Active le ‘mode reclaim’
2	Force l’écriture des pages marquées ‘dirty’
4	Swaps les pages

e travaux

e puis free.

Posix | sched_{get,set}affinity(…)

- La glibc (>2.3) supporte les wrappers vers les appels systèmes à sched_{get,set}affinity(…)
- Facile à mettre en oeuvre




```
#define _GNU_SOURCE
#include <sched.h>
int sched_setaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);
int sched_getaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);
```

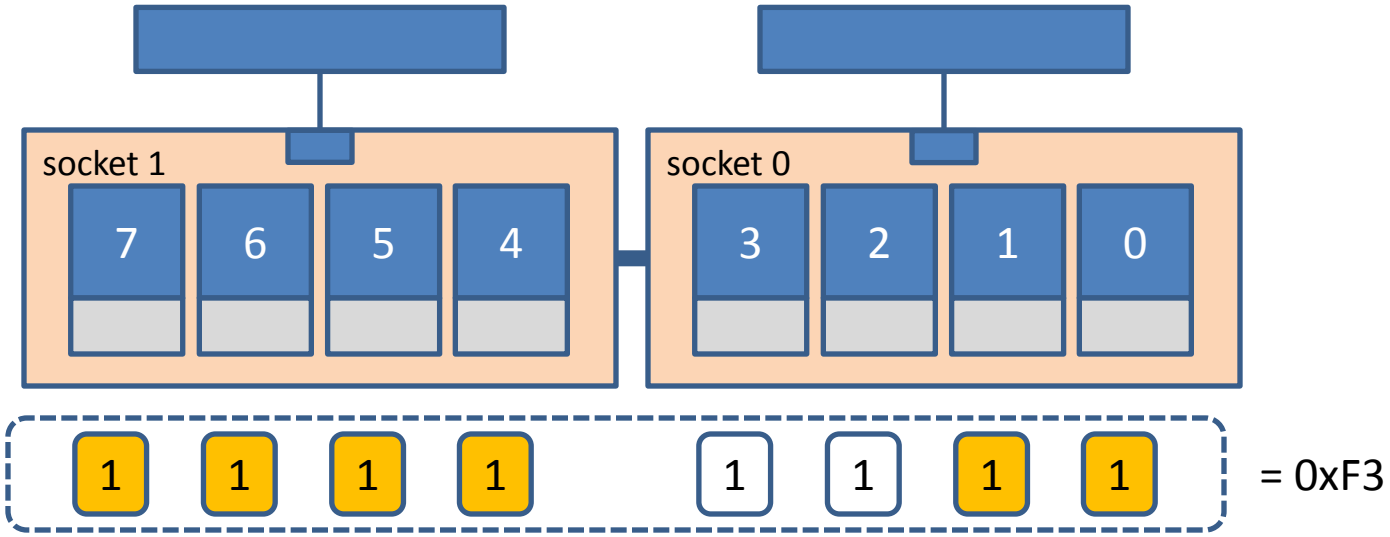
```
#define _GNU_SOURCE
…
#include <sched.h>

char * getcpumask(char *buffer, size_t size)
{
   cpu_set_t     mask;
   unsigned      ncpus;
   unsigned      i;

   memset(buffer,'\0',1024);

   ncpus = sysconf(_SC_NP

   sched_getaffinity(0,si
   for(i=0;i<ncpus;i++) {
        if(CPU_ISSET(i,&m
            sprintf(buffe
        }
   }
   return (buffer);
}
```

```
…
char buffer[1024];
…
#pragma omp parallel private(tid,buffer)
{
     tid = omp_get_thread_num();
     printf("%d running on %s\n",tid,getcpumask(buffer,1024));
}
…
```

taskset

- Commande la plus simple pour récupérer ou ajuster l'affinité CPU d'un processus
- Permet d'assurer le placement
  - D'un nouveau processus lors de son exécution
  - D'un processus existant à partir de son PID
- Permet de lire le mask d'un processus (via son PID)

taskset




```
> sleep 120 &
[1] 11659

> taskset -pc 11659
pid 11659's current affinity list: 0-7

> taskset -p 11659
pid 11659's current affinity mask: ff

> taskset -pc 0,1,4-7 11659
pid 11659's current affinity list: 0-7
pid 11659's new affinity list: 0,1,4-5

> taskset -p 11659
pid 11849's current affinity mask: f3

> kill 11659
```

libnuma

- Bilbiothèque openSource proposant une API (simple) pour la gestion de la politique NUMA supportée par le kernel linux

```
#include <numa.h>
libnuma.so
```

- http://oss.sgi.com/projects/libnuma/
- Reference
  - http://developer.amd.com/assets/LibNUMA-WP-fv1.pdf

Exemple d'utilisation de l'API libnuma

```
#include <numa.h>

int  numa_avail = -1;
int  max_node = -1;
char *ptr = NULL;

if((numa_avail=numa_available())==0) {
    printf("NUMA support enabled. Max number of nodes: %d\n",
           (max_node=numa_max_node())+1
          );
    } else puts("NUMA support is not available");

printf("Allocating memory...");
if((numa_avail==0)&&(getenv("FILLMEM_NO_NUMA")==NULL)) {
     ptr=(char*)numa_alloc_interleaved(size);
} else {
     ptr=(char*)malloc(size);
}

if(numa_avail==0) {
    for(i=0;i<=max_node;i++) {
        long node_free;
        long node_size = numa_node_size(i,&node_free);
        printf("  Node %02d - %ld Mibytes, free %ld Mibytes\n",
               i,node_size>>20,node_free>>20);
    }
}
```

libnuma | `numactl`

- “Livrée” avec la libnuma
- `numactl` [*options*] `command` [*args*] ...

Description	options

Politique et placement mémoire	`--localalloc`
	`--preferered=<`*nodes*`>`
	`--interleave=<`*nodes*`>`
	`--membind=<`*nodes*`>`

Placement CPU	`--physcpubind==<`*cpus*`>` `--cpunodebind`
Infos	`--hardware` `--show`

```
> numactl --hardware
available: 2 nodes (0-1)
node 0 size: 12092 MB
node 0 free: 270 MB
node 1 size: 12120 MB
node 1 free: 564 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

Statistiques numa

- Disponibles via:
  - `/sys/devices/system/node/node*/numastat`
  - la commande `numastat`
    - L'unité est la page (4K)
  - Tout ça mérite d'être enrobé !
    - CF. TP

```
> numastat
                           node0           node1
numa_hit            37270443889     41299449921
numa_miss            1738099296      3204337700
numa_foreign         3204337700      1738099296
interleave_hit          1347000         1243207
local_node          37266097532     41259383644
other_node           1742445653      3244403977
```

variable	Description
`numa_hit`	Number of pages allocated from the node the process wanted
`numa_miss`	Number of pages allocated from this node, but the process preferred another node
`numa_foreign`	Number of pages allocated another node, but the process preferred this node
`interleave_hit`	Number of pages allocated from this node while the process was running locally
`local_node`	Number of pages allocated from this node while the process was running remotely (on another node)
`other_node`	Number of pages allocated successfully with the interleave strategy

Placement des threads openmp

- L'utilisation de `taskset` et `numactl` sont un bon point de départ

- Specification openmp

  `OMP_PROC_BIND=[true|false]`

  Assure le binding des processus openmp sur un processeur

- Placement:
  - GNU

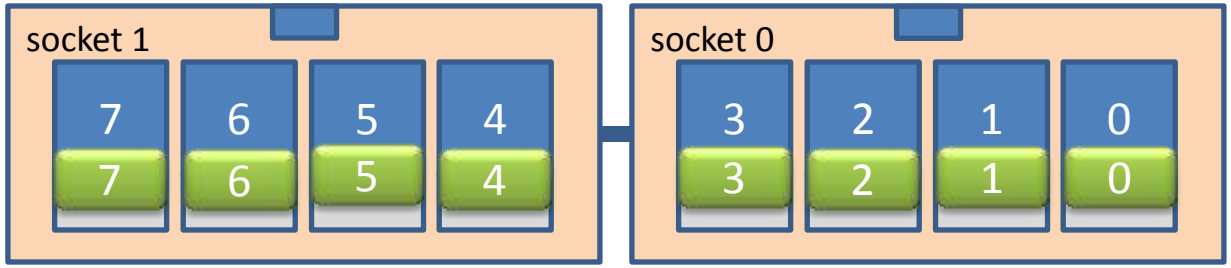
    `GOMP_CPU_AFFINITY=M0-N0:S0,M1-N1:S1,…`
  - Intel

    `KMP_AFFINITY`

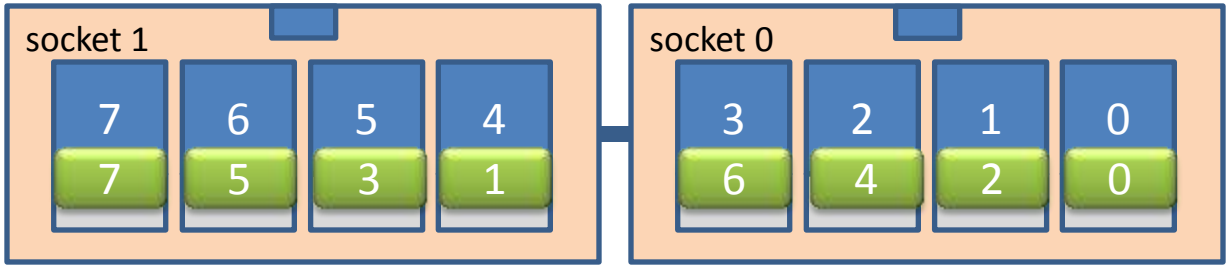
    - Détection de la topologie du noeud.
    - Interface plus ou moins 'user friendly' !
    - Très nombreuses options. Consulter la doc !
    - Existe aussi une API pour contrôler l'affinité directement dans le code

Placement des threads openmp | KMP_AFFINITY

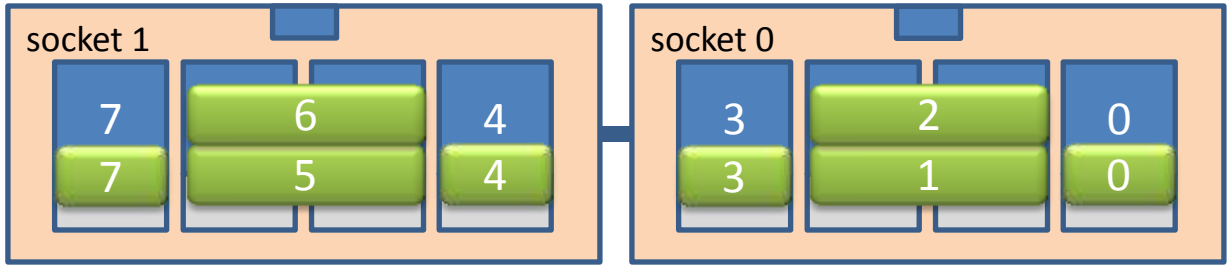
- `KMP_AFFINITY=verbose,compact`



- `KMP_AFFINITY=verbose,scatter`



- `KMP_AFFINITY="verbose,granularity=fine,proclist=[0,{1,2},3,4,{5,6},{5,6},7],explicit"`



Placement des threads openmp | KMP_AFFINITY

- Les librairies Intel ® fournissent une API de haut niveau pour la gestion des threads

```
int main() {
    #pragma omp parallel
    {
        int tmax = omp_get_max_threads();
        int tnum = omp_get_thread_num();
        int nproc = omp_get_num_procs();
        int ncores = nproc / 2;
        int i;
        kmp_affinity_mask_t mask;

        kmp_create_affinity_mask(&mask);
        for (i = tnum % ncores; i < tmax; i += ncores)
        {
              kmp_set_affinity_mask_proc(i, &mask);
        }

        if (kmp_set_affinity(&mask) != 0)
        {
           …
        }
…
}
```

cpusets et confinement des processus

- Confinement des processus à des sous-ensembles de processeurs et de nœuds mémoire
  - On parle parfois d'enveloppe
- Se présente sous la forme d'un pseudo système de fichiers
  - Habituellement monté dans `/dev/cpuset`
  - Permet d'accéder au mécanisme du noyau du même nom
    - `mbind`, `sched{get,set}affinity` se cachent derrière
- Utiliser par les gestionnaires de ressources
- Intégrés dans cgroups
  - Gestion et partage des ressources

```
> mkdir /dev/cpuset
> mount -t cpuset cpuset /dev/cpuset
```

```
> cd /dev/cpuset
> mkdir moncpuset
> cd moncpuset/
> echo 0-7 > cpus
> echo 0 > mems
> cat /proc/self/cpuset
/
> cat /proc/self/status
…
Cpus_allowed:   ffff
Cpus_allowed_list:
Mems_allowed:   000000
Mems_allowed_list:
…

> echo $$ > tasks
> cat /proc/self/cpuse
/moncpuset
> cat /proc/self/statu
…
Cpus_allowed:   00ff
Cpus_allowed_list:
Mems_allowed:   000000
Mems_allowed_list:
…
> taskset -c 0 date
Mon Oct  8 10:25:28 CE
> taskset -c 8 date
sched_setaffinity: Invalid argument
failed to set pid 0's affinity.
```

```
> ls -1
cgroup.procs
cpu_exclusive
cpus
mem_exclusive
mem_hardwall
memory_migrate
memory_pressure
memory_spread_page
memory_spread_slab
mems
notify_on_release
placement
sched_load_balance
sched_relax_domain_level
tasks
virtualize
```

Huge pages

- Réduire les “TLB Misses”
- Les xeons supportent les HPs de 2MB ou 1GB
  - Linux supporte les HPs de 2MB
- Par forcément facile de mettre en place même si la complexité a diminué avec libhugetlbfs
- Peut se faire de manière transparente (Transparent Huge Pages):
  - Depuis RHEL6
  - “System wide”
  - pas forcément conseillée en environnement de prod

Mise en oeuvre des huge pages

- A l'aide de la libraire `libhugetlbfs`
  - Interface via un pseudo système de fichiers
  - Prérequis

```
> mkdir /libhugetlbfs
> groupadd libhp
> chgrp libhp /libhugetlbfs
> chmod 770 /libhugetlbfs
> usermod moi -G libhp
> mount -t hugetlbfs hugetlbfs /libhugetlbfs
```

  - Méthode 1 : relinker votre application

```
> gcc -B $HOME/local/lib/libhugetlbfs/ -Wl,--hugetlbfs-link=BDT  mon_programme.c
> ./a.out
```

  - Méthode 2 : interposition (`LD_PRELOAD`)

```
> export LD_PRELOAD=/usr/lib64/libhugetlbfs.so
> export HUGETLB_MORECORE=yes
> ./a.out
```

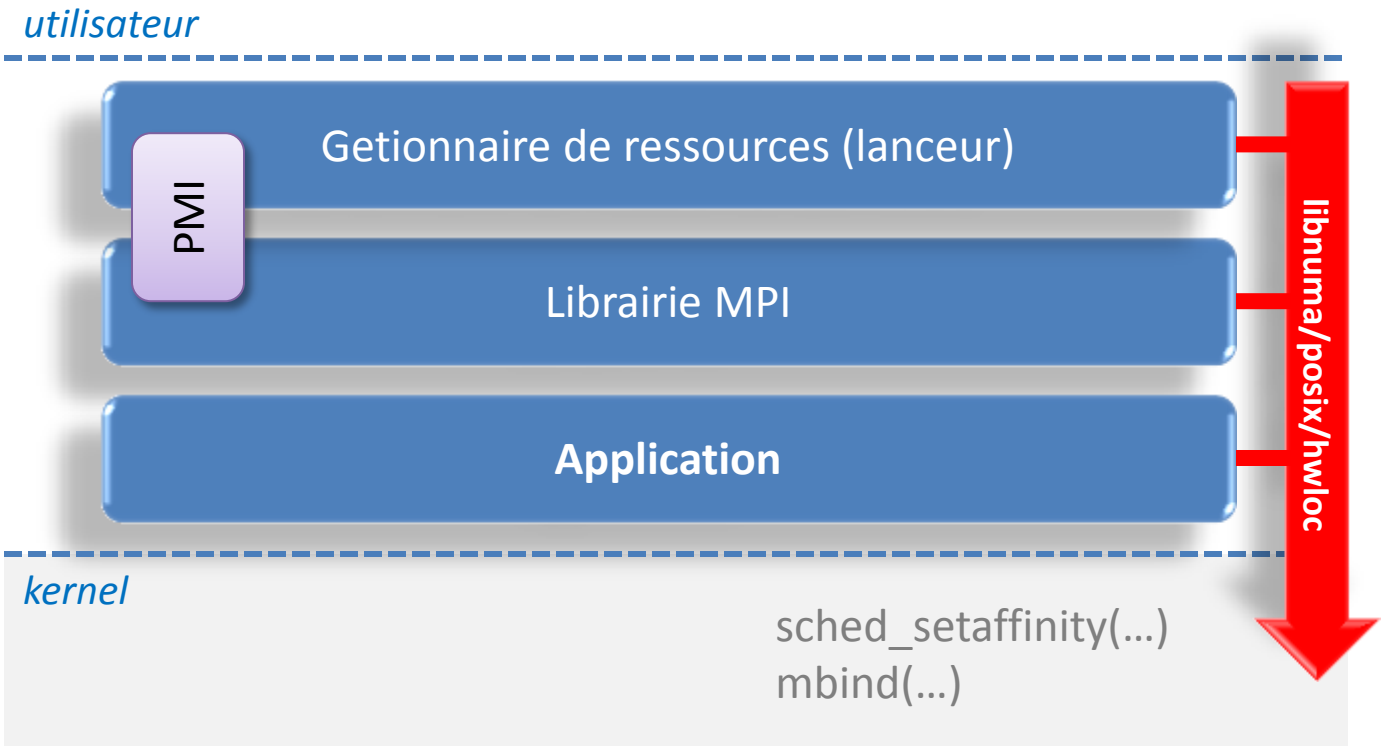
- Transparent huge pages (THP)

```
> echo "always" >/sys/kernel/mm/redhat_transparent_hugepage/enabled
```

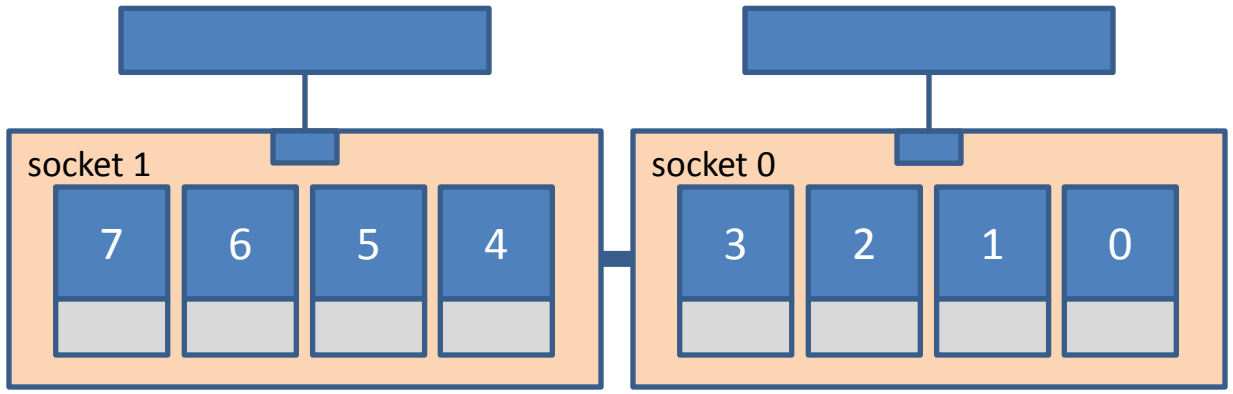
Placement des processus MPI et travaux hybrides

Qui attache quoi et quand ? Et Où ? …

```
srun … ./monapp_mpi
```

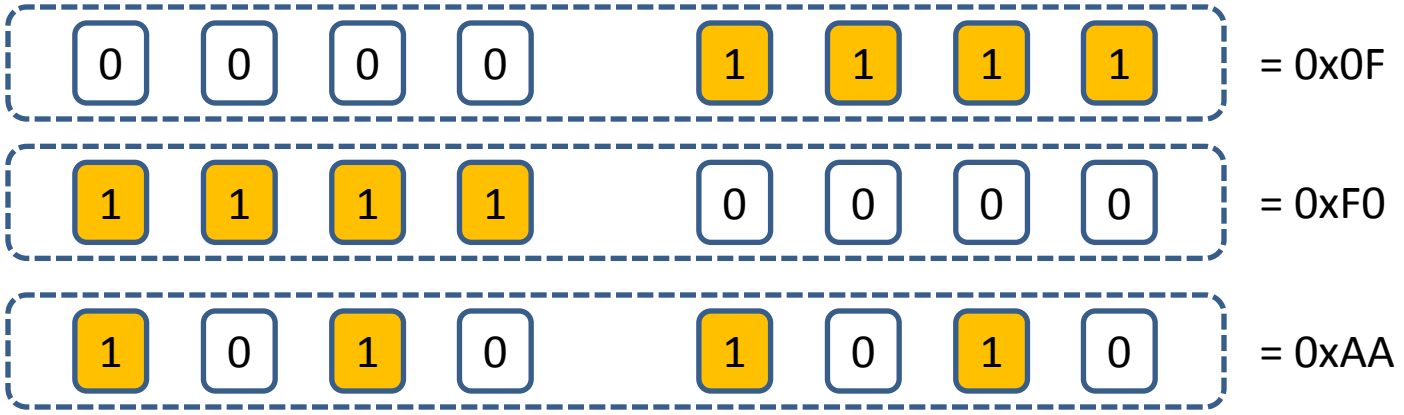


Placement via le lanceur | slurm

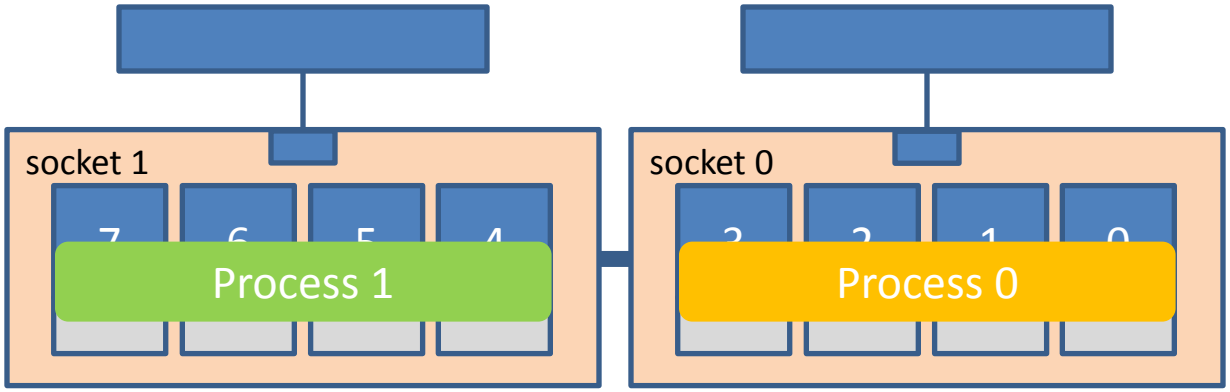


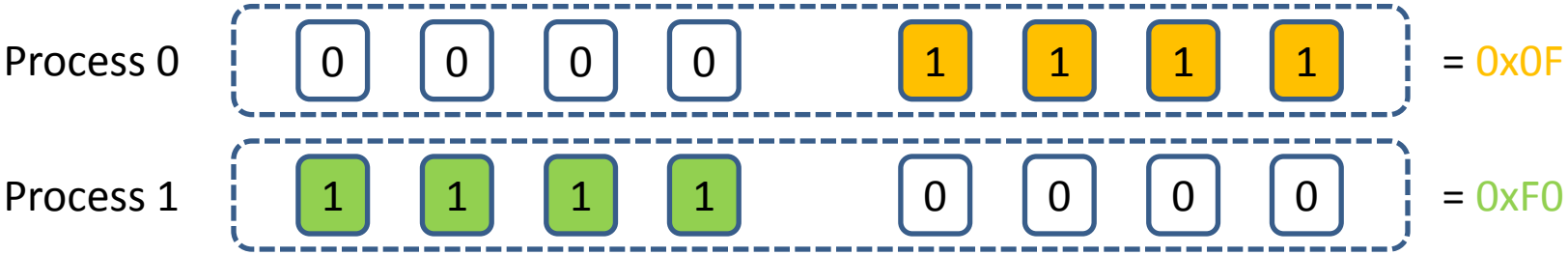
bitmask

Exemples:



Placement via le lanceur | slurm

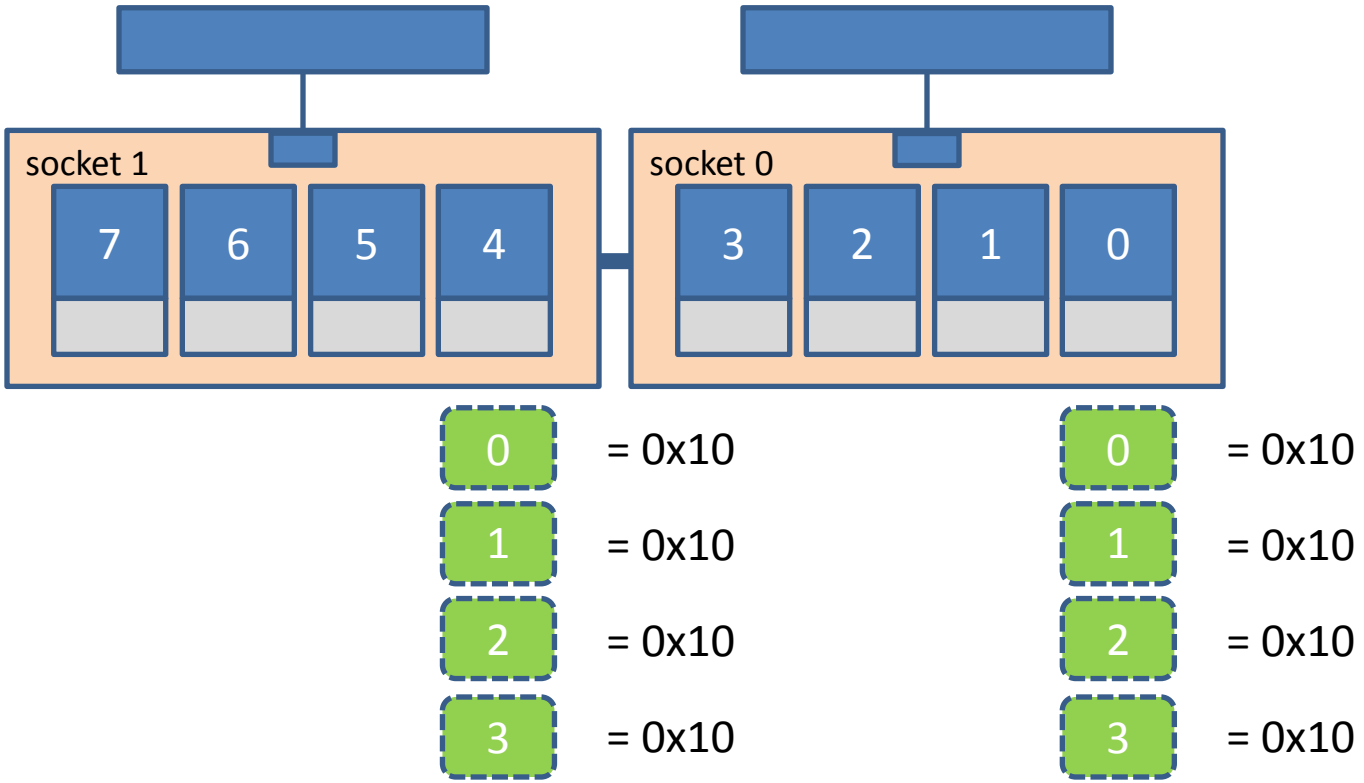





```
srun --cpu_bind=mask_cpu:0x0F,0xF0 -N …
```

Placement via le lanceur | slurm

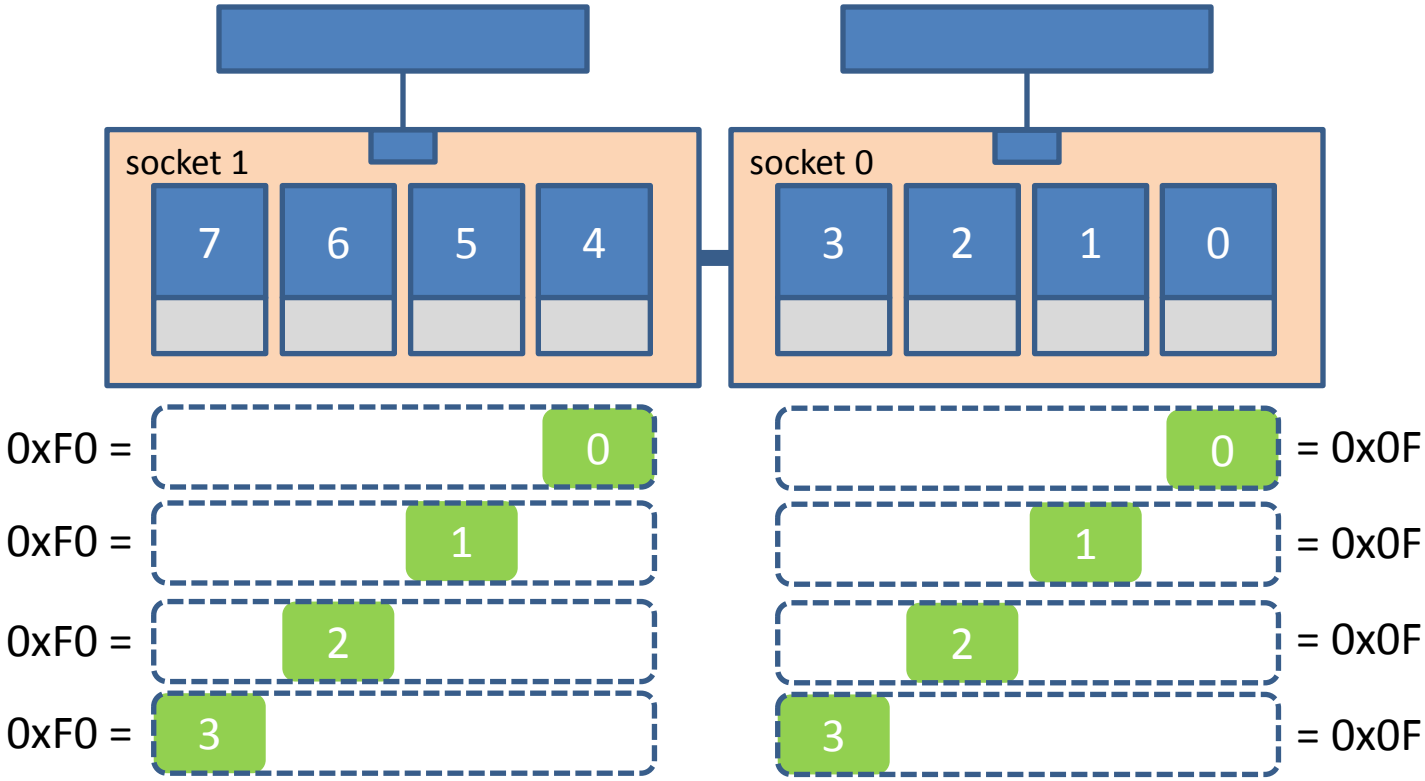
```
export OMP_NUM_THREADS=4
srun --cpu_bind=map_cpu=0,4 -N 1 -n 2 prg_omp
```



Ne pas oublier:
les fils héritent du bitmask du père …

Placement via le lanceur | slurm

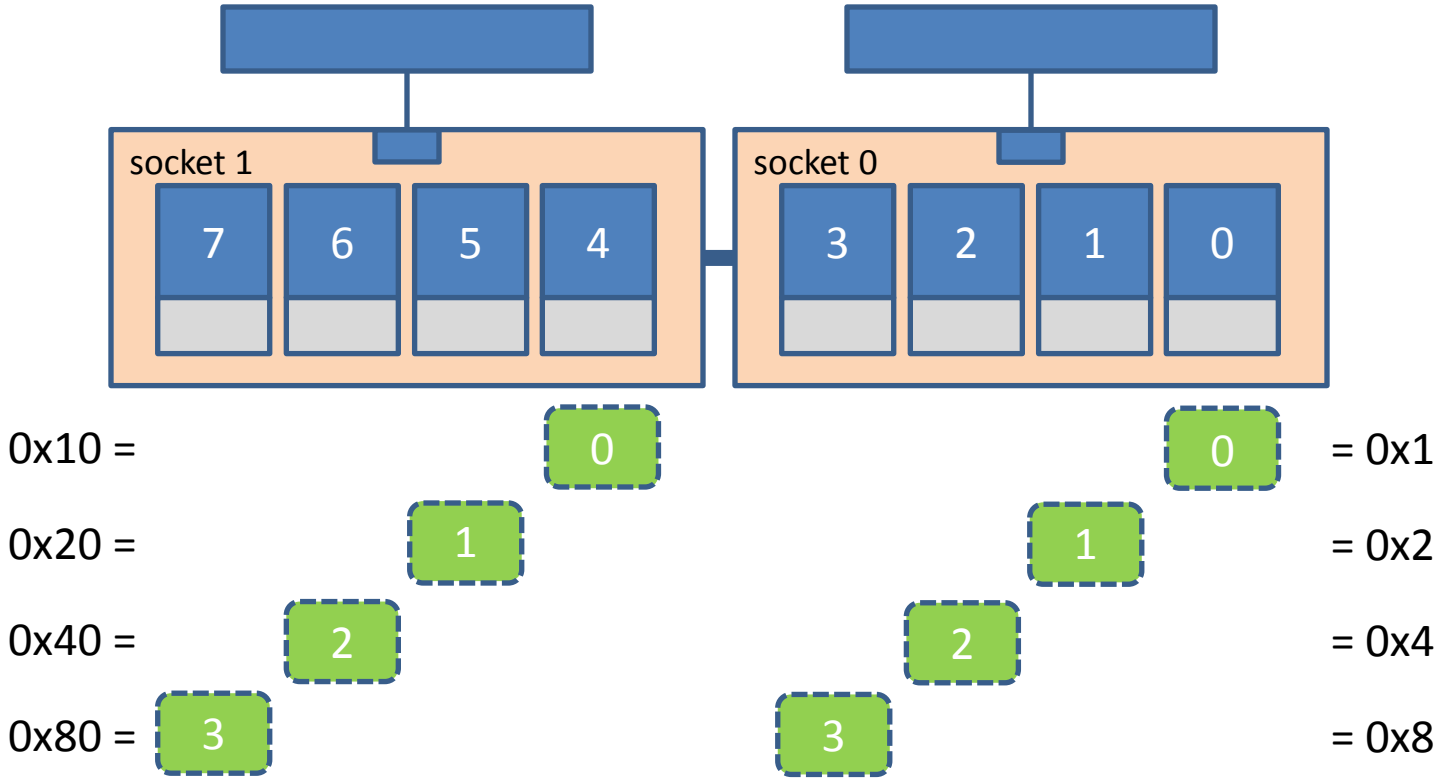
```
export OMP_NUM_THREADS=4
srun --cpu_bind=mask_cpu:0xF,0xF0 -N 1 -n 2 prg_omp
```



Linux fera le reste …

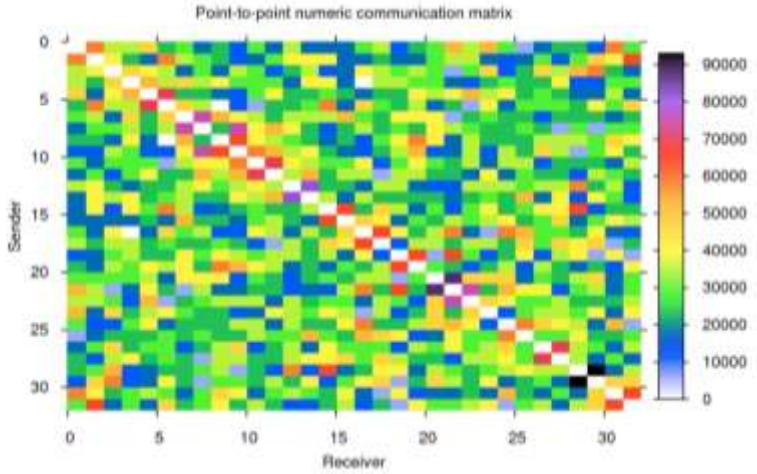
Placement via le lanceur | slurm

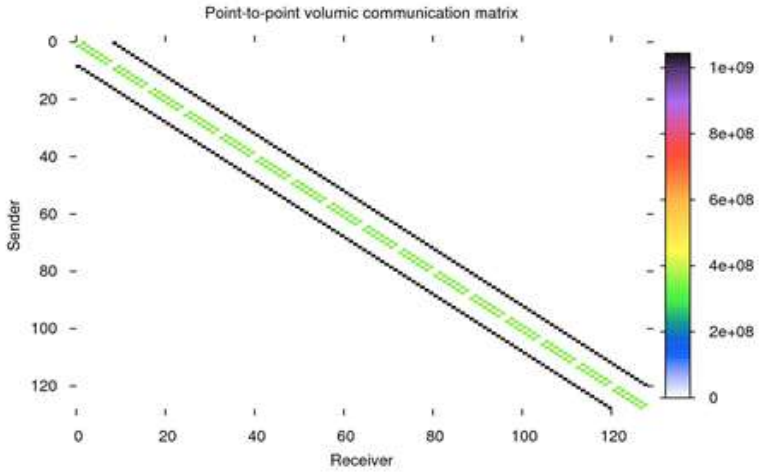
```
export OMP_NUM_THREADS=4
export OMP_PROC_BIND=true
srun --cpu_bind=mask_cpu:0xF,0xF0 -N 1 -n 2 prg_omp
```



Placement des process MPI

- Pas de recette absolue
- Le placement optimal dépend de nombreux facteurs
  - Architecture Réseaux / Interconnect
    - Topologie (type, niveau de blocage = prunning)
      - Clos, fat-tree, tore 2D/3D …
  - Le routage (technologie)
  - Et bien évidemment, du “pattern de comm” de l’application **ET** du cas test employé : **PROFILING**

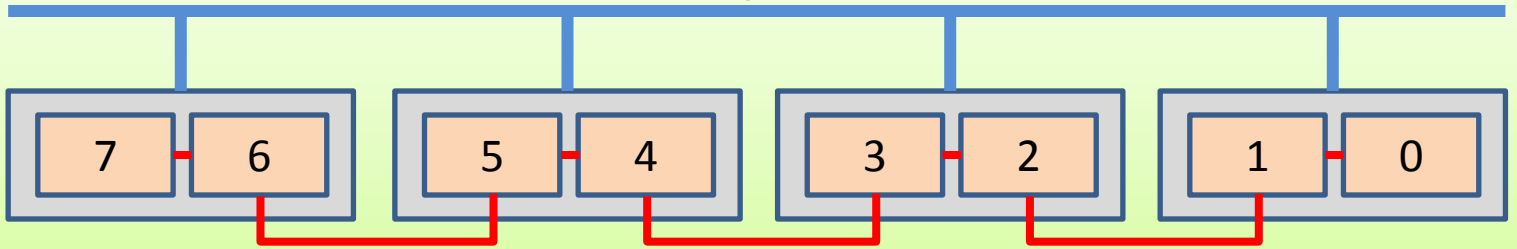




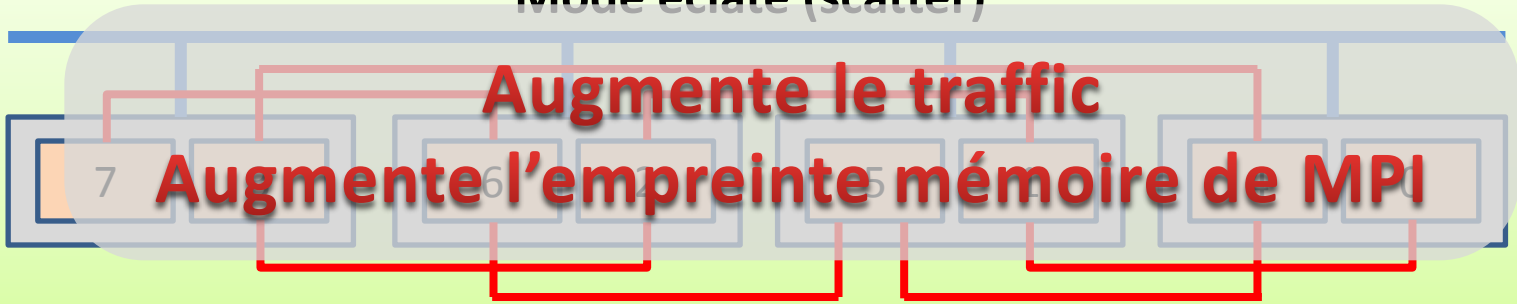
Distribution des rangs MPI

- Exemple:
  - Job hybride MPI/OpenMP (4 noeuds, 8 tâches, PPN=2, OMP=4)
  - Decomposition de domaine 2D. Découpage en sous-domaine (socket)

Mode Compact (bunch)



Mode éclaté (scatter)



Augmente le traffic

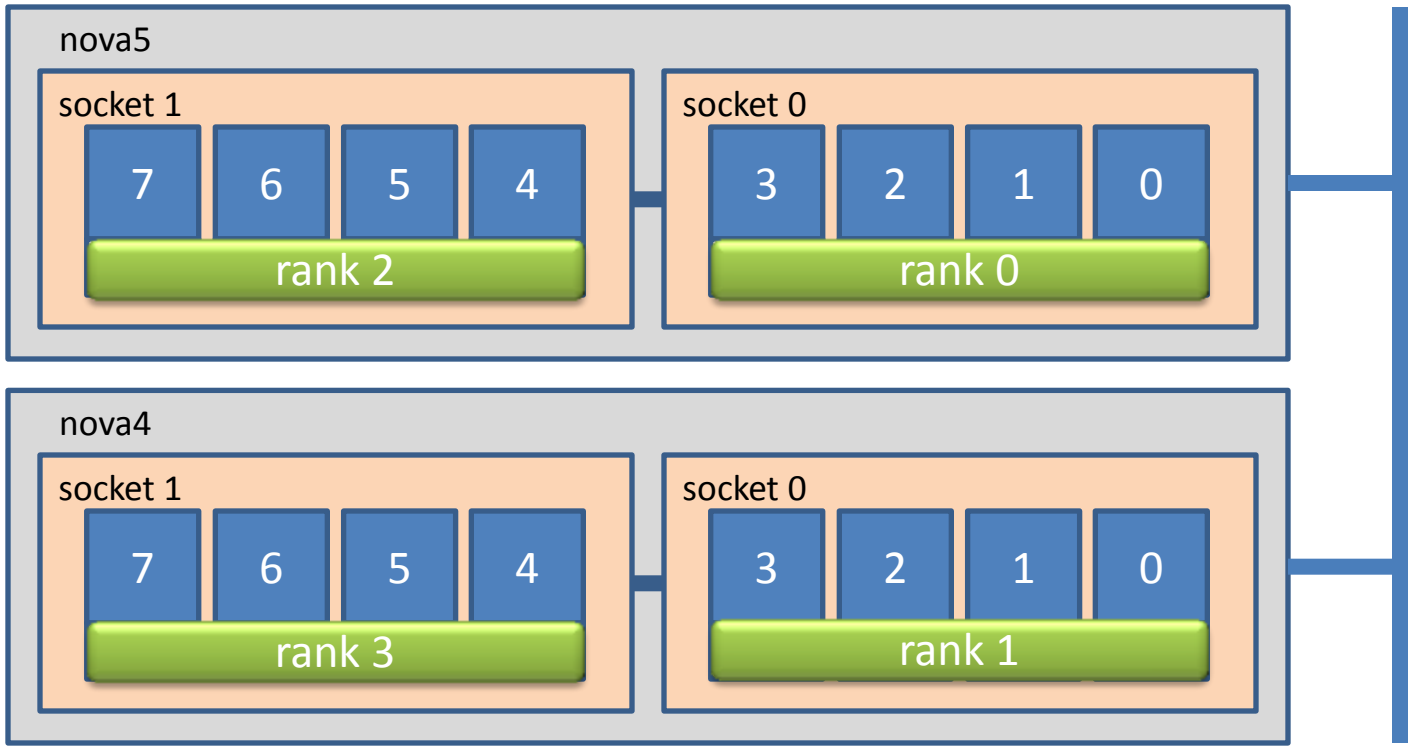
Augmente l'empreinte mémoire de MPI

Placement fin des rangs MPI avec OpenMPI

- Utilisation des “rankfile”
  - Permet de distribuer finement (explicitement) les rangs MPI sur les différents noeuds et d’assurer le placement

```
…
rank <n>=<nom du noeud> slot=<socket>:<cores>
…
```

- OpenMPI >= 1.3
- Mise en oeuvre pas forcément évidente dans un environnement de batch
  - Construction du fichier `rankfile` à partir de la liste de noeuds alloués

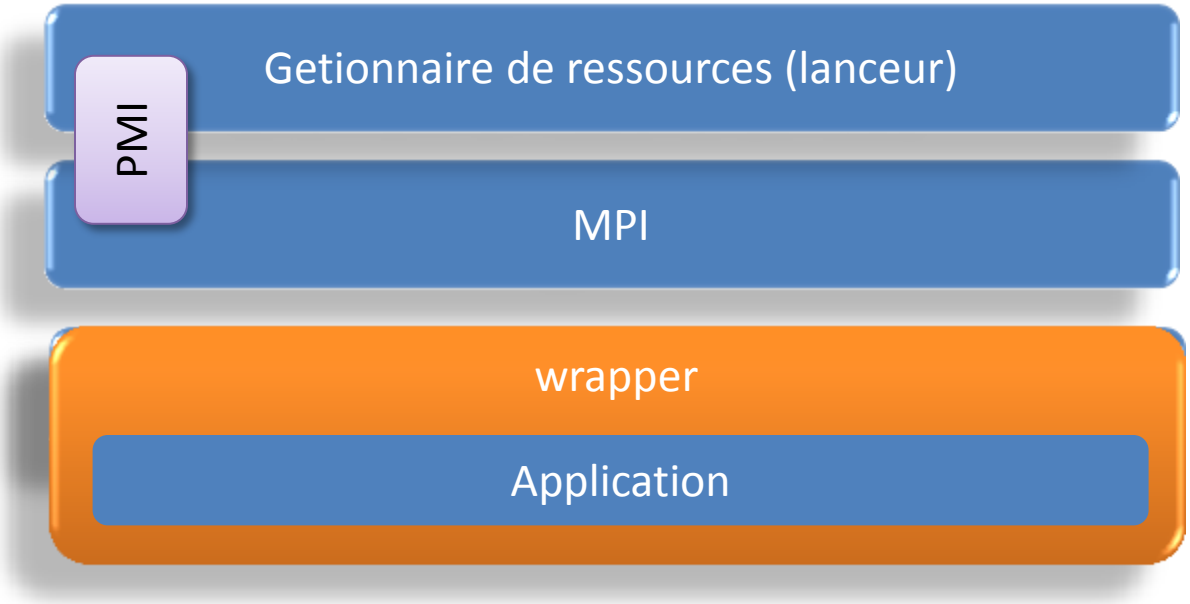



```
> cat > rankfile.txt << EOF
rank 0=nova4 slot=0:0-3
rank 1=nova5 slot=0:0-3
rank 2=nova4 slot=1:0-3
rank 3=nova5 slot=1:0-3
EOF

> mpirun --prefix -host nova4,nova5 --rankfile ./rankfile.txt -n 4 ./hw

processor per node (check on rank 0): 8
proc   0/  4 says Hello,World! (from node nova4) 0 1 2 3
proc   2/  4 says Hello,World! (from node nova4) 4 5 6 7
proc   1/  4 says Hello,World! (from node nova5) 0 1 2 3
proc   3/  4 says Hello,World! (from node nova5) 4 5 6 7
```

```
> cat > rankfile.txt << EOF
rank 0=nova4 slot=0:0-3
rank 1=nova5 slot=0:0-3
rank 2=nova4 slot=1:0-3
rank 3=nova5 slot=1:0-3
EOF

> mpirun --prefix -x OMP_NUM_THREADS=4 -x KMP_AFFINITY=compact -host nova4,nova5 --
rankfile ./rankfile.txt -n 4 ./hw-omp

processor per node (check on rank 0): 8
proc   0/  4 says Hello,World! (from node nova4) 0 1 2 3
proc   2/  4 says Hello,World! (from node nova4) 4 5 6 7
proc   1/  4 says Hello,World! (from node nova5) 0 1 2 3
proc   3/  4 says Hello,World! (from node nova5) 4 5 6 7
hello, was launched by process (0,0) and I am running on logical core: 0
hello, was launched by process (2,0) and I am running on logical core: 4
hello, was launched by process (0,3) and I am running on logical core: 3
hello, was launched by process (0,2) and I am running on logical core: 2
hello, was launched by process (1,0) and I am running on logical core: 0
hello, was launched by process (1,1) and I am running on logical core: 1
hello, was launched by process (1,3) and I am running on logical core: 3
hello, was launched by process (3,0) and I am running on logical core: 4
hello, was launched by process (3,3) and I am running on logical core: 7
hello, was launched by process (3,2) and I am running on logical core: 6
hello, was launched by process (1,2) and I am running on logical core: 2
hello, was launched by process (3,1) and I am running on logical core: 5
hello, was launched by process (0,1) and I am running on logical core: 1
hello, was launched by process (2,2) and I am running on logical core: 6
hello, was launched by process (2,3) and I am running on logical core: 7
hello, was launched by process (2,1) and I am running on logical core: 5
```

Technique du wrapper

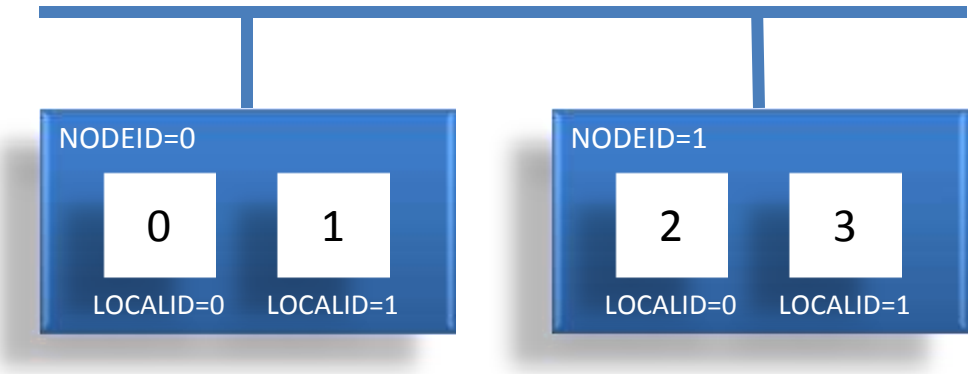
- Script d'enrobage (généralement en bash) destiné à effectuer certaines opérations avant (et/ou après) l'exécution de l'application à proprement dite.



- Permet des réglages "fins"

Technique du wrapper | variables d'environnement exportées

```
# srun -p NH24 -N 2 -n 4 -c 4 env
…
SLURM_NODEID=0
SLURM_PROCID=0
SLURM_LOCALID=0
…
SLURM_NODEID=0
SLURM_PROCID=1
SLURM_LOCALID=1
…
SLURM_NODEID=1
SLURM_PROCID=2
SLURM_LOCALID=0
…
SLURM_NODEID=1
SLURM_PROCID=3
SLURM_LOCALID=1
…
```



Description	slurm	openMPI/bullxmpi	Intel ® MPI
Nombre total de processus MPI		OMPI_COMM_WORLD_SIZE	PMI_SIZE
Rang absolu du noeud	SLURM_NODEID	OMPI_COMM_WORLD_NODE_RANK	
Rang absolu du processus MPI	SLURM_PROCID	OMPI_COMM_WORLD_RANK	PMI_RANK
Nombre de processus MPI sur le noeud		OMPI_COMM_WORLD_LOCAL_SIZE	
Rang local	SLURM_LOCALID	OMPI_COMM_WORLD_LOCAL_RANK	

Technique du wrapper

- Par exemple:
  - Avec Intel ® MPI, constituer le wrapper pour permettre le lancement de 2 tâches MPI par noeud (1 tâche par socket) et permettre l'exécution de 4 tâches openmp par tâche MPI.
  - Wrapper "extra" specialisé !

```
#!/bin/bash

RANK=$PMI_RANK
LRANK=$((RANK%2))
NODE=$LRANK

echo "running process $RANK on $(hostname):$NODE"

export OMP_NUM_THREADS=4
export I_MPI_PIN=disable

exec numactl --cpunodebind=$NODE -l $@
```

```
# ls
wrapper.sh
# chmod +x wrapper.sh
```

Technique du wrapper | Exemple

```
# mpirun -hosts kay380,kay381 -ppn 2 -n 4 ./wrappers.sh ./hw-omp

running process 0 on kay380:0
running process 1 on kay380:1
running process 2 on kay381:0
running process 3 on kay381:1
processor per node (check on rank 0): 8
proc   0/  4 says Hello,World! (from node kay380) 0 1 2 3
proc   2/  4 says Hello,World! (from node kay381) 0 1 2 3
proc   1/  4 says Hello,World! (from node kay380) 4 5 6 7
proc   3/  4 says Hello,World! (from node kay381) 4 5 6 7
hello, was launched by process (0,2) and I am running on logical core: 0 1 2 3
hello, was launched by process (0,0) and I am running on logical core: 0 1 2 3
hello, was launched by process (0,3) and I am running on logical core: 0 1 2 3
hello, was launched by process (0,1) and I am running on logical core: 0 1 2 3
hello, was launched by process (2,0) and I am running on logical core: 0 1 2 3
hello, was launched by process (2,3) and I am running on logical core: 0 1 2 3
hello, was launched by process (2,2) and I am running on logical core: 0 1 2 3
hello, was launched by process (3,1) and I am running on logical core: 4 5 6 7
hello, was launched by process (2,1) and I am running on logical core: 0 1 2 3
hello, was launched by process (3,0) and I am running on logical core: 4 5 6 7
hello, was launched by process (3,3) and I am running on logical core: 4 5 6 7
hello, was launched by process (3,2) and I am running on logical core: 4 5 6 7
hello, was launched by process (1,3) and I am running on logical core: 4 5 6 7
hello, was launched by process (1,0) and I am running on logical core: 4 5 6 7
hello, was launched by process (1,2) and I am running on logical core: 4 5 6 7
hello, was launched by process (1,1) and I am running on logical core: 4 5 6 7
```

Technique du wrapper | Exemple

```
# export OMP_PROC_BIND=true
# mpirun -hosts kay380,kay381 -ppn 2 -n 4 ./wrappers.sh ./hw-omp

running process 0 on kay380:0
running process 1 on kay380:1
running process 2 on kay381:0
running process 3 on kay381:1
processor per node (check on rank 0): 8
proc   2/  4 says Hello,World! (from node kay381) 0 1 2 3
proc   0/  4 says Hello,World! (from node kay380) 0 1 2 3
proc   3/  4 says Hello,World! (from node kay381) 4 5 6 7
proc   1/  4 says Hello,World! (from node kay380) 4 5 6 7
hello, was launched by process (0,0) and I am running on logical core: 0
hello, was launched by process (0,1) and I am running on logical core: 1
hello, was launched by process (0,2) and I am running on logical core: 2
hello, was launched by process (0,3) and I am running on logical core: 3
hello, was launched by process (2,0) and I am running on logical core: 0
hello, was launched by process (3,1) and I am running on logical core: 5
hello, was launched by process (3,0) and I am running on logical core: 4
hello, was launched by process (3,2) and I am running on logical core: 6
hello, was launched by process (3,3) and I am running on logical core: 7
hello, was launched by process (1,0) and I am running on logical core: 4
hello, was launched by process (1,1) and I am running on logical core: 5
hello, was launched by process (1,2) and I am running on logical core: 6
hello, was launched by process (1,3) and I am running on logical core: 7
hello, was launched by process (2,1) and I am running on logical core: 1
hello, was launched by process (2,3) and I am running on logical core: 3
hello, was launched by process (2,2) and I am running on logical core: 2
```

Technique du wrapper | autre exemple d'utilisation

```
#!/bin/bash

if [ $SLURM_PROCID -eq 0 ] ; then
        gdb $1
else
        exec $@
fi
```

- Permet d'attacher un débogueur (`gdb`) sur le processus de rang 0 et d'intéragir avec. Les autres processes s'exécutent normalement.

Suppression et vérification du binding …

… Pour faire mieux ! 

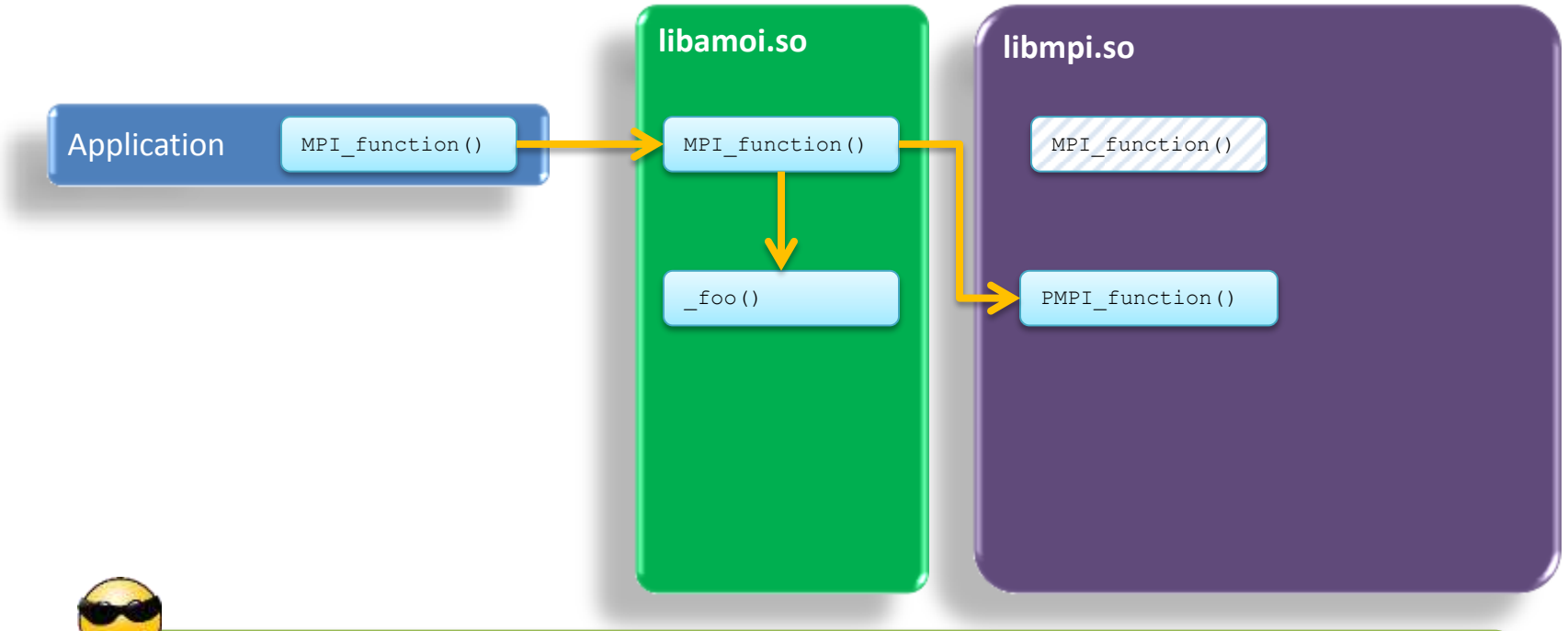
lanceur	Suppression du binding	Affichage du binding
Slurm	`srun` **`--cpu_bind=none`** …	`srun` **`--cpu_bind=verbose,`**… …
OpenMPI/bullxmpi	`mpirun` **`--bind-to-none`** …	`mpirun` **`--report-bindings`** …
Intel ® MPI	`export` **`I_MPI_PIN=disabled`** …	`export` **`I_MPI_DEBUG=4`**

- Pour tous les lanceurs, les options liées aux placements sont très riches et permettent de “tout” faire.

- Assurez-vous du placement.
  - A l’aide des modes “verbeux” des lanceurs
  - Testez vos options sur de simples “HelloWorld” hybrides
  - Instrumentez vos codes avec `sched_getaffinity(…)` ou utilisez des librairies d’interposition.

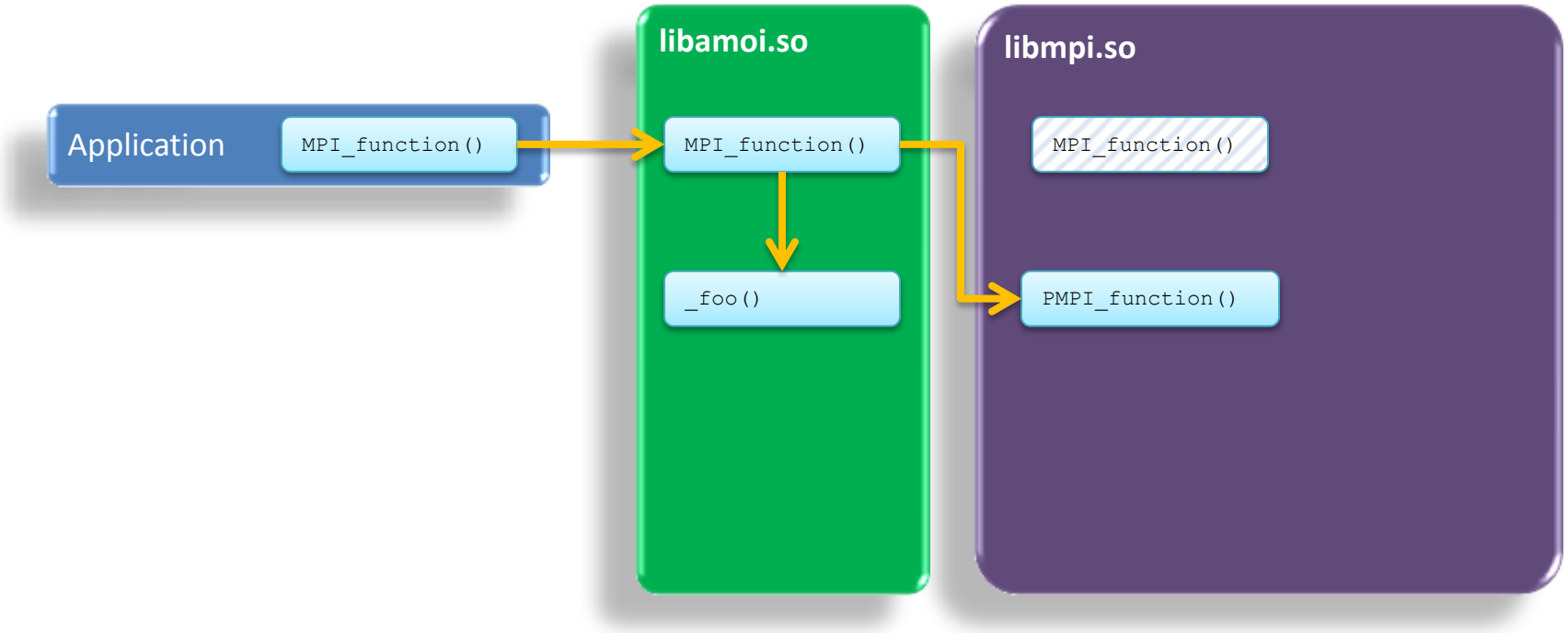
- Lisez le manuel !

Bilbiothèque d'interposition




```
int MPI_Init(int* argc, char*** argv)
{
                int res = PMPI_Init(argc,argv);
                MPI_Comm_rank(MPI_COMM_WORLD,&RANK);
                (void) getaffinity();
                if (RANK==0) {
                        fprintf(stdout,"== MPI started at %s\n",lo_asctime());
                        fflush(stdout);
                }
                fflush(stdout);
                MPI_Barrier(MPI_COMM_WORLD);
                T_START = MPI_Wtime();
                return res;
}
```

Bilbiothèque d'interposition



- Il faut créer une librairie dynamique

```
cc -shared -Wl,-soname=libamoi.so -o libamoi.so main.o
```

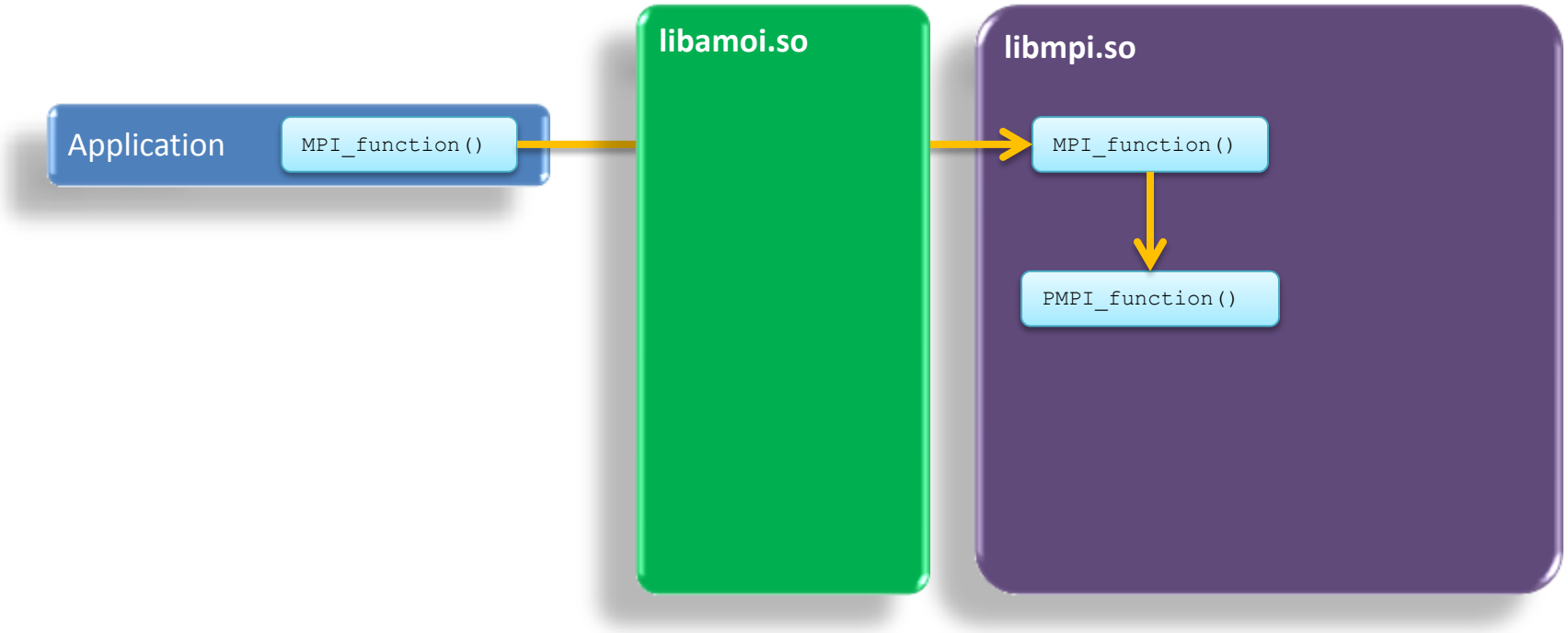
- Utilisation:
  - On rejoue l'édition de lien de l'application en prenant soin à l'ordre des bibliothèques sur la ligne de commande …

```
mpicc … -L /opt/mpi/openmpi/lib -L ./lib -lamoi -lmpi
export LD_LIBRARY_PATH=$HOME/local/lib:$LD_LIBRARY_PATH
```

  - On utilise la variable d'environnement `LD_PRELOAD`

```
export LD_PRELOAD=$HOME/local/lib/libamoi.so
```

Bilbiothèque d’interposition



Conclusion (?)

- L’architecture des machines est complexe, parfois plus complexe qu’on peut le penser
  - Hiérarchique
- La notion de “shared memory” peut parfois donner la fausse illusion d’un espace totalement uniforme d’un point de vue des accès
- Complique la “portabilité” des accès aux ressources (mémoires et périphériques)
- Mais “*un Homme averti en vaut deux*” ;-)

- Ce qui n’est pas abordé ici (faute de temps)
  - L’optimisation de l’empreinte mémoire MPI
    - Allocation OnDemand
    - InfiniBand: SRQ, XRC …
  - Le placement des IRQs
    - Documentation kernel : IRQ-affinity.txt
    - `/proc/interrupts`
    - `/proc/irq/`<#irq>`/smp_affinity`
    - Service irqbalance
  - Autres modèles programmatifs et implémentations
    - MPI-2 (one sided)
    - SHMEM/PGAS
  - Aspects Hybrides GPUs
    - DirectGPU …

Architect of an Open World™