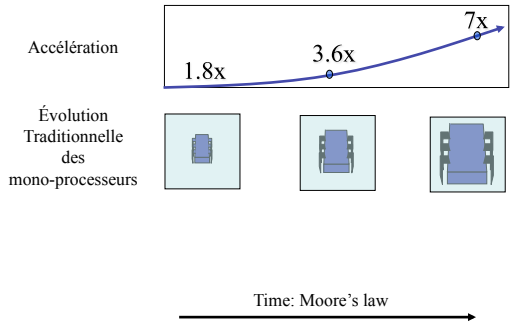
Programmation Système des Multicœurs

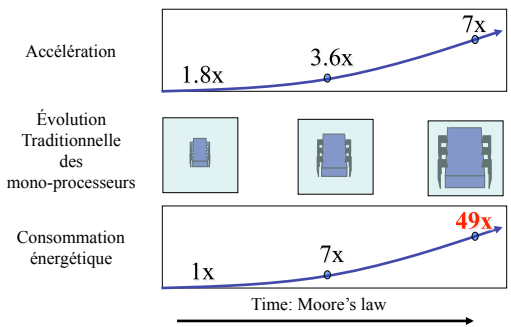
Gaël Thomas
gael.thomas@lip6.fr

Université Pierre et Marie Curie
Master Informatique
M2 – Spécialité SAR

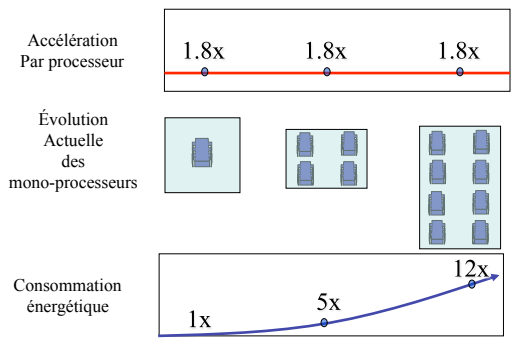
Évolution de la puissance des machines



Évolution de la puissance des machines

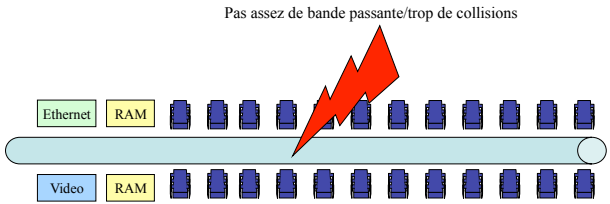


Évolution de la puissance des machines



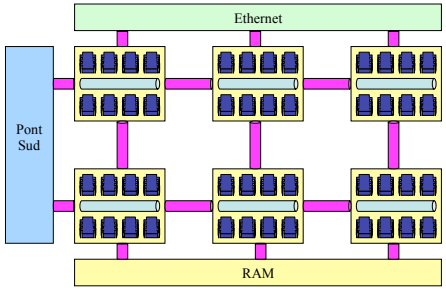
Architecture d'un multicœurs

Impossible d'utiliser une topologie classique à base de BUS!



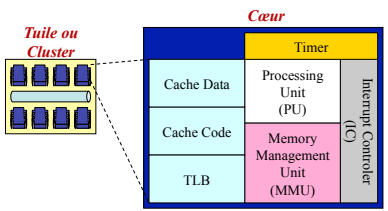
Architecture d'un multicœurs

Pour passer à l'échelle, il faut changer la topologie



Architecture d'un multicœurs

Un peu de terminologie

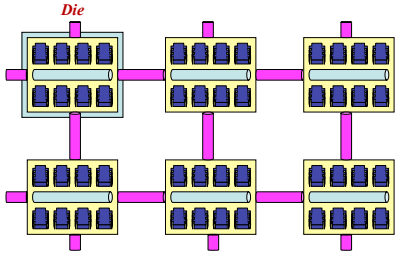


Architecture d'un multicœurs

Un peu de terminologie :

die = ensemble de clusters correspondant à un **circuit intégré unique**

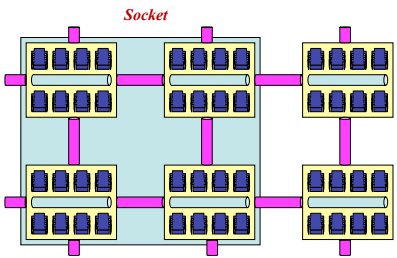
(souvent die = cluster)



Architecture d'un multicœurs

Un peu de terminologie :

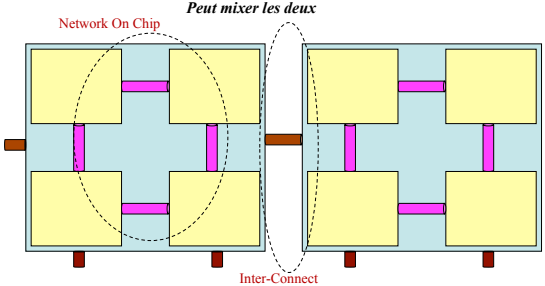
package/socket = ensemble de dies identiques intégrés sur un même support (celui qu'on achète)



Architecture d'un multicœurs

Un peu de terminologie

Network On Chip (NOC) si sur un seul die
Inter-connect si entre die
Peut mixer les deux



Multicœurs et gestion mémoire

Problème : comment invalider une ligne de cache

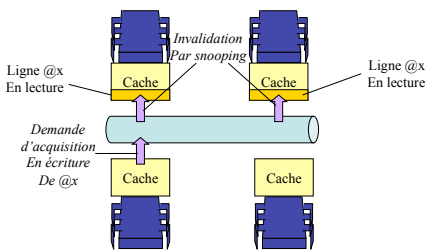
Protocole MOESI : état par ligne de cache

	Modified	Owned	Exclusive	Shared	Invalid
Mémoire centrale	Incohérente	Incohérente	Cohérente	Cohérente	Cohérente
Répliquées	Non	Oui	Non	Oui	Indéfini
Écriture locale	Modified	Modified	Modified	Modified	Modified
Lecture locale	Modified	Owned	Exclusive	Shared	Exclusive
Écriture autre	Invalid	Invalid	Invalid	Invalid	Invalid
Lecture autre	Owned	Owned	Shared	Shared	Invalid

Multicœurs et gestion mémoire

Problème : comment invalider une ligne de cache

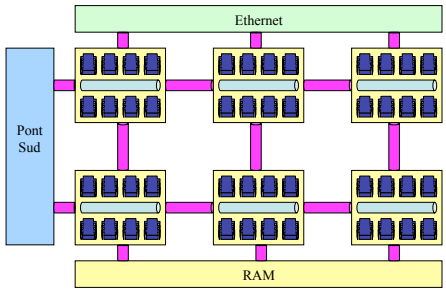
Classiquement avec un bus unique : changement d'état via méthode de snooping (espionnage des demandes de lecture/écriture)



Multicœurs et gestion mémoire

Problème : comment invalider une ligne de cache

Snooping ne passe pas à l'échelle car demande un broadcast



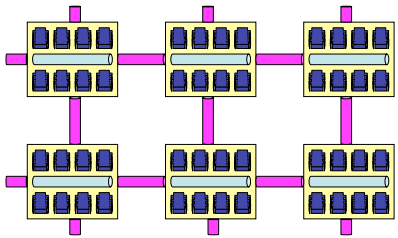
Multicœurs et gestion mémoire

Problème : comment invalider une ligne de cache

Solution : une table associant adresse physique avec cœurs qui cachent la ligne

invalidations adressées explicitement aux cœurs qui cachent la ligne

Problème : où placer cette table!

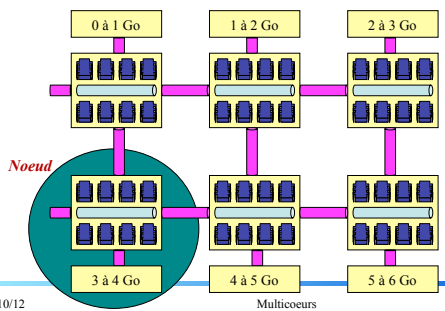


Multicœurs et gestion mémoire

Solution : partitionnement de l'espace d'adressage physique

⇒ notion de nœud = ensemble de clusters gérant une partition mémoire

(en général : un nœud = un die ou un nœud = un cluster)

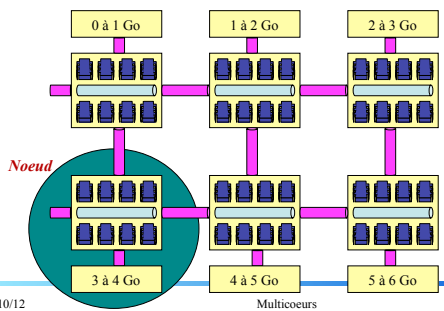


Multicœurs et gestion mémoire

Table d'association ligne de cache/cœur partitionnée sur les nœuds

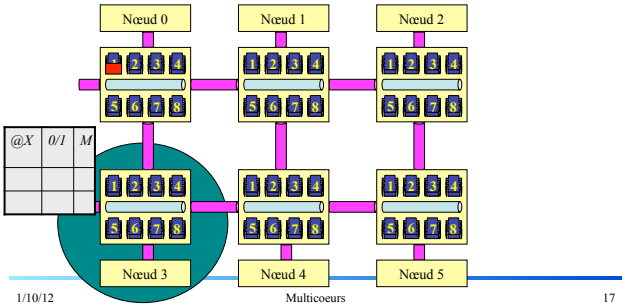
Une écriture vers la mémoire est effectuée via ce nœud

(assure la cohérence)



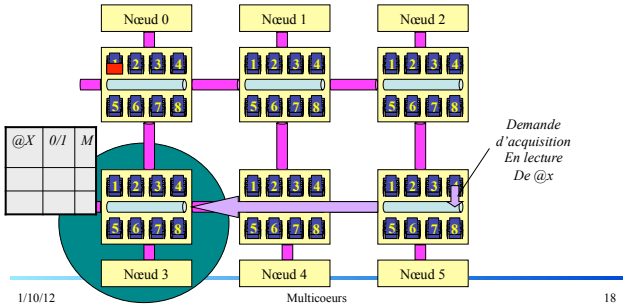
Multicœurs et gestion mémoire

Exemple : lecture de @X (nœud 3) à partir du cœur 5/4



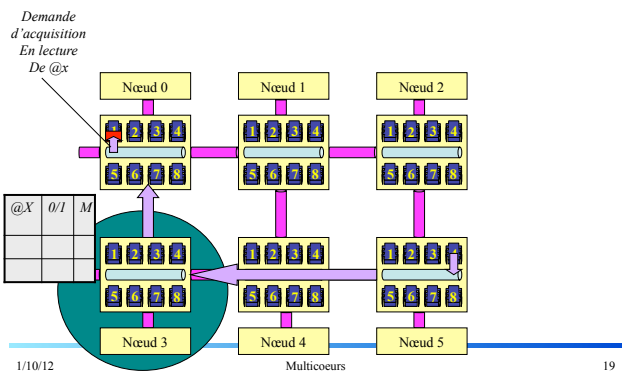
Multicœurs et gestion mémoire

Exemple : lecture de @X (nœud 3) à partir du cœur 5/4



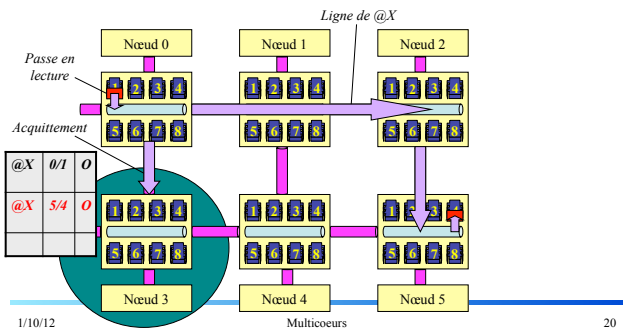
Multicœurs et gestion mémoire

Exemple : lecture de @X (nœud 3) à partir du cœur 5/4



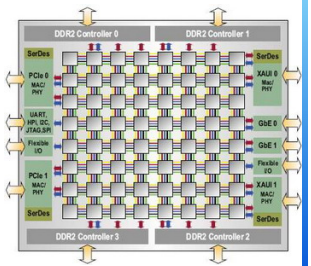
Multicœurs et gestion mémoire

Exemple : lecture de @X (nœud 3) à partir du cœur 5/4

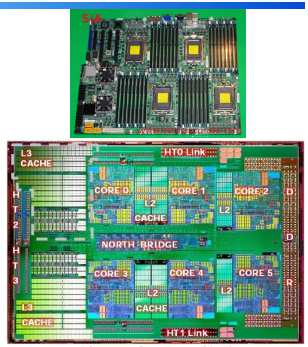


Multicoeurs et gestion mémoire

Deux exemples typiques de multicœurs



64 cœurs de Tilera (unique die)



12 core Opteron 6174

(6 cœurs * 2 dies * 4 sockets = 48)

Multicœurs et gestion mémoire

Conséquences pour le programmeur du multicœurs

- ✓ Accès à la mémoire physique non uniforme : plus un nœud est proche du contrôleur RAM, plus il a un accès rapide
- ✓ Partitionnement de la mémoire sur les nœuds change les latences

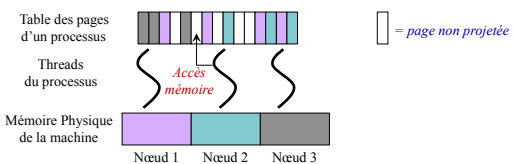
  Deux types d'accès : accès local et accès distant

Gestion mémoire sous Linux

Augmenter la localité d'accès (i.e. accès plutôt à la mémoire locale au nœud)

Trois principes :

- ✓ Eviter de migrer un processus d'un nœud sur un autre
- ✓ Essayer de maintenir les threads d'un processus sur le même noeud
- ✓ First-touch : lors du premier accès à une page non projetée en mémoire, la page physique est alouée sur le nœud local

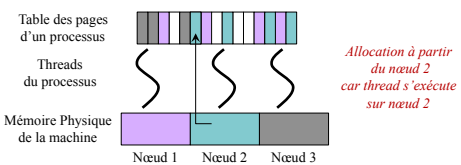


Gestion mémoire sous Linux

Augmenter la localité d'accès (i.e. accès plutôt à la mémoire locale au nœud)

Trois principes :

- ✓ Eviter de migrer un processus d'un nœud sur un autre
- ✓ Essayer de maintenir les threads d'un processus sur le même noeud
- ✓ First-touch : lors du premier accès à une page non mappée en mémoire, la page physique est alouée sur le nœud local



Gestion mémoire sous Linux

Quelques fonctions utiles

Placement des processus et des threads

- ✓ setaffinity(ensemble E de cœurs) : oblige le scheduler à placer les threads du processus sur un des cœurs de E
- ✓ pthread_setaffinity(ensemble E de cœurs) : idem pour un thread

Placement de pages :

- ✓ mbind(plage d'adresses virtuelle, ensemble de nœud) : demande à n'allouer la mémoire physique des pages qu'à partir de l'ensemble de nœuds
- ✓ madvise(plage d'adresses virtuelle, flag) : demande à libérer les pages physique de la plage d'adresse (permet de migrer)

Allocation mémoire :

- ✓ numa_alloc_onnode(taille, nœud N) : malloc sur le nœud N

Dernières avancées en OS Multicœurs

Linux (et autres OS à mémoire partagée) :

- Tous les cœurs partagent leur mémoire
- Communication par mémoire partagée/lock + Interruption entre les cœurs
- Nombreuses optimisations (compteurs répliqués par cœur, RCU etc…)
- Performances bonnes jusqu'à 48 cœurs! [Boyd-Wickizer, OSDI 2010]

Multikernel (Barelfish [Baumann, SOSP 2009], Helios [Nightingale, SOSP 2009]) :

- Pas de mémoire physique partagée entre les cœurs dans l'OS
- Communication par envoi de message (évite la contention sur lignes de cache)

OS Clustering [Song, Eurosys 2011] :

- Chaque nœud exécute un Linux dans une machine virtuelle
- Les machines virtuelles utilisent des pages physiques différentes
- Communication entre les cœurs d'un nœud via mémoire partagée/lock
- Communication entre les nœuds via envois de message