Formation en Calcul Scientifique - LIEM2I

Introduction au calcul parallèle

Loïc Gouarin, Violaine Louvet, Laurent Series

Groupe Calcul CNRS

9-13 avril 2012

Qu'est ce que le calcul parallèle

Le calcul parallèle

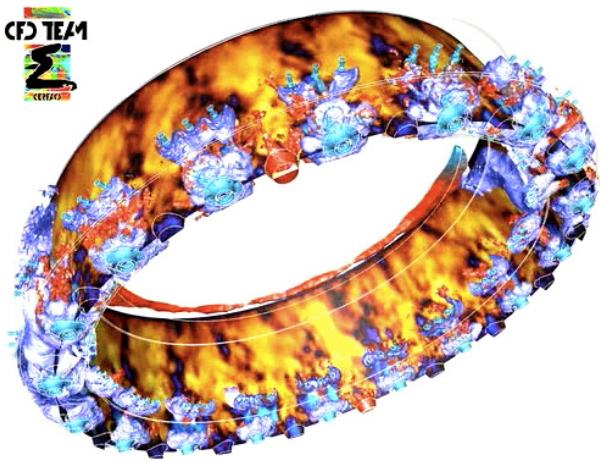
l'ensemble des techniques **logicielles** et **matérielles** permettant l'exécution **simultanée** de séquences d'instructions **indépendantes** sur des processeurs et/ou cœurs différents.

- techniques matérielles : les différentes architectures de calculateur parallèle
- techniques logicielles : les différents modèles de programmation parallèle

Objectifs du calcul parallèle

- Exécution plus rapide d'un programme en distribuant le travail ;
- Exécution de problèmes plus gros en utilisant plus de ressources matérielles, notamment la mémoire.

Exemple :



Allumage d'une chambre de combustion d'hélicoptère réalisé avec le code AVBP

Temps de calcul sur 112 processeurs Intel Xeon hexa-cœurs cadencé à 2,67 Ghz :

78 heures

Temps de calcul sur 1 processeur de même type :

près de 1 an

Exemple de supercalculateur

Tianhe-1A (Chine) : supercalculateur classé 2^{ieme} au Top 500 (11/2011)

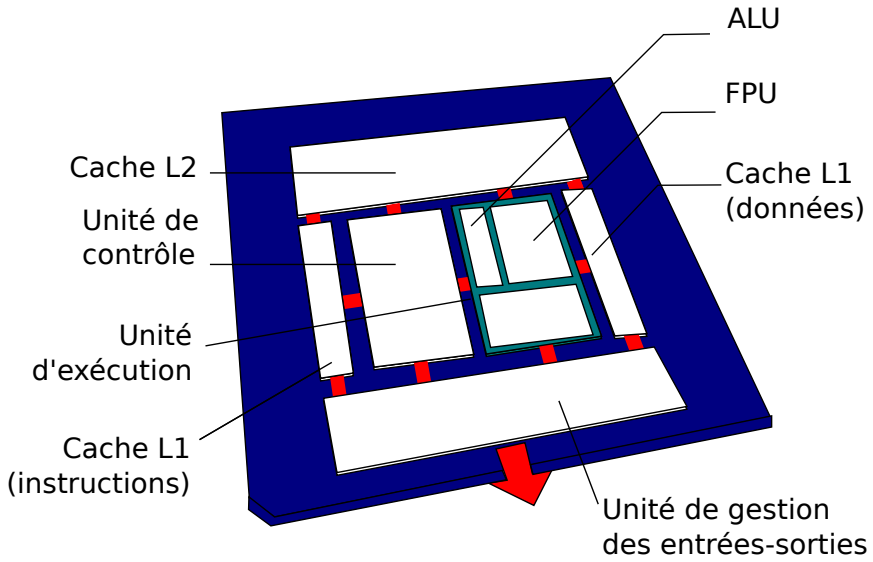
- 7168 nœuds bi-processeurs hexa-cœurs Intel X5670 à 2.93 GHz équipés de carte GPU Nvidia Tesla M2050
- soit 186368 cœurs (86016 venant des CPU et 100352 venant des GPU)
- Consommation : 4040.00 kW
- Performance théorique : 4.7 PetaFlops
- Performance réalisée sur le Linpack : 2.6 PetaFlops

Les composants d'un supercalculateur

- Processeur
  - fournit la puissance de calcul
- Noeud
  - contenant plusieurs processeurs partageant une même mémoire
- Mémoire
  - dont l'accès depuis le processeur n'est pas uniforme
- Réseau
  - relient les noeuds entre eux
  - de plusieurs types (calcul, administration, E/S)
- Accélateur
  - processeur graphique (GPU) utilisé pour le calcul
  - processeur MIC (Many Integrated Core) d'Intel (plusieurs cœurs sur la même puce), pas encore en production
- Stockage

Composants d'un processeur (CPU, *Central Processing Unit*)

- 1 unité de gestion des entrée-sorties.
- 1 unité de commande (*Control Unit*) : lit les instructions arrivant, les décode puis les envoie à l'unité d'exécution.



- 1 unité d'exécution
  - 1 ou plusieurs ALU (*Arithmetical and Logical Unit*) : en charge notamment des fonctions basiques de calcul arithmétique ;
  - 1 ou plusieurs FPU (*Floating Point Unit*) : en charge des calculs sur les nombres flottants.
- Registres et caches : mémoires de petites tailles.

Principales caractéristiques d'un processeur (1/2)

- Le jeu d'instructions (*ISA, Instruction Set Architecture*) constituel'ensemble des opérations élémentaires qu'un processeur peut accomplir.
  - CISC (*Complex Instruction Set Computer*) :
    choix d'instructions aussi proches que possible d'un langage de haut niveau.
  - RISC (*Reduced Instruction Set Computer*) :
    choix d'instructions plus simples et d'une structure permettant une exécution très rapide.
- La finesse de gravure des transistors dont dépend la fréquence de du processeur et sa consommation (de l'ordre de 32 *nm* en 2011).

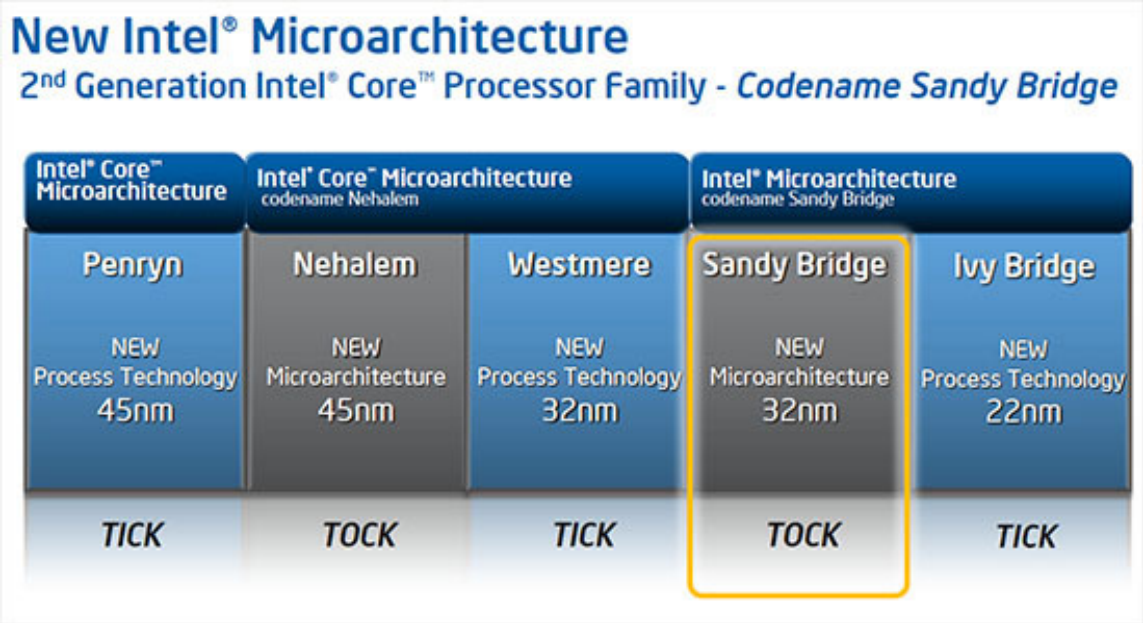
Caractéristiques d'un processeur (2/2)

- La fréquence d'horloge (en *MHz* ou *GHz*)
  - Elle détermine la durée d'un cycle (un processeur exécute chaque instruction en un ou plusieurs cycles).
  - Plus elle augmente, plus le processeur exécute d'instructions par seconde.
  - Actuellement, supérieur à 3 *GHz* (temps de cycle $\approx 0.33\,ns$).
  - Croissance de plus en plus limitée.
  - Principales limites à sa croissance : la consommation électrique et la dissipation thermique du processeur.

    $Puissance\ électrique\ dissipée \propto Fréquence^3$

Augmenter les performances des processeurs

- augmenter la fréquence d'horloge mais limites techniques et solution coûteuse.
- introduire de nouvelles technologies afin d'optimiser l'architecture générale du processeur.



Roadmap d'Intel : Tick-Tock

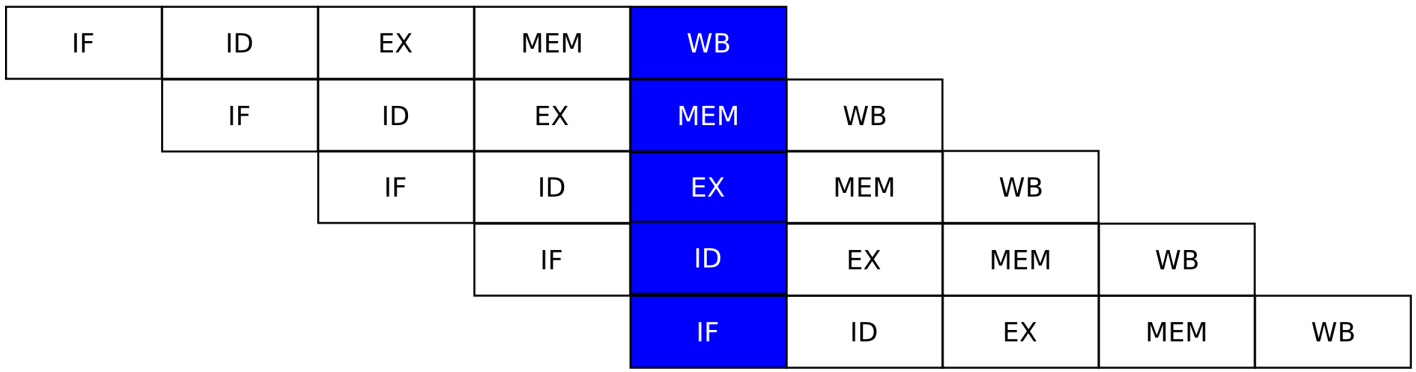
Optimiser l'architecture

Parallélisme au niveau de l'architecture.

Ce parallélisme peut être qualifié d'implicite : il est transparent pour l'utilisateur.

- Parallélisme d'instructions (*ILP : Instruction Level Parallelism*)
  - Pipeling et architectures superscalaires.

Exemple de pipeline sur 5 étages (5 instructions en parallèle sur 9 cycles).



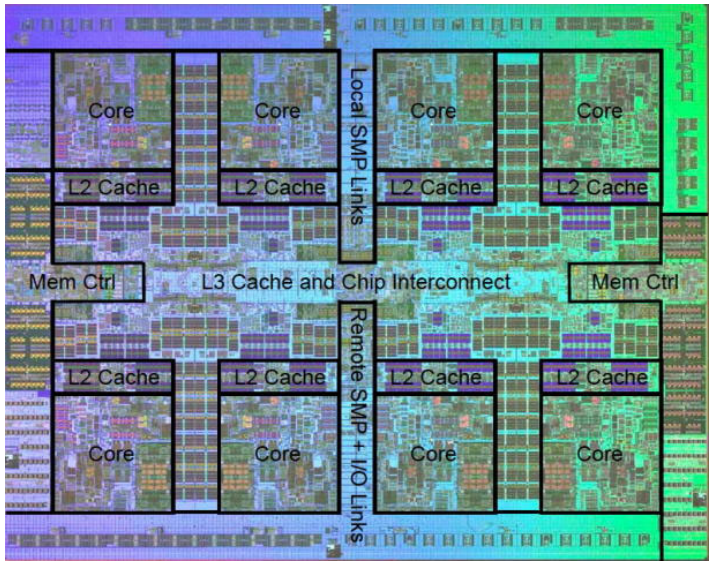
Optimiser l'architecture

- Parallélisme de données (*DLP : Data Level Parallelism*)
  - SIMD (*Single Instruction Multiple Data*) : SSE, AVX...
  - Processeur vectoriel.
- Parallélisme au niveau des threads (*TLP : Thread Level Parallelism*)
  - SMT (*Simultaneous MultiThreading*), HyperThreading chez Intel.

Performances des processeurs : multi-cœurs

Plusieurs instances de calcul sur un même *socket*(1).

- Connectique inchangée par rapport à un mono-cœur ;
- Certains éléments (caches L2/L3...) peuvent être partagés ;
- Exemple : Intel Westmere jusqu'à 10 cœurs par socket, Power 7 jusqu'à 10 cœurs par socket



IBM Power7 (octo-cœurs)

(1) socket : connecteur utilisé pour interfacer un processeur avec une carte mère (les multi-processeurs utilisent plusieurs sockets)

Intérêt du multicœur ?

- **Diminuer la fréquence des cœurs** pour réduire la consommation électrique et la dissipation thermique, **mais en mettre plus** pour gagner en puissance de calcul.

- Rappel : $\textit{Consommation électrique} \propto \textit{Fréquence}^3$
  $\textit{Puissance} \propto \textit{Fréquence}$

	mono-cœur	bi-cœurs
Fréquence	F	0.75F
Consommation par processeur	W	0.84 W
Performance par processeur	P	1.5P

Un bi-cœurs avec une fréquence moindre de 25% par rapport à un mono-cœur offre 50% de performance en plus et 20% environ de consommation en moins qu'un mono-cœur.

Puissance crête d'une machine

- Puissance crête (*Peak performance*) R_{peak} exprimée en *Flops* (*FLoating point Operations Per Second*).

R_{peak} (pour des flottants DP ou SP)

$$R_{peak} = N_{FLOP}/cycle \times Freq \times N_c$$

Freq : Fréquence du processeur.
$N_{FLOP}/cycle$: Nombre d'opérations flottantes (SP ou DP) effectuées par cycle.
N_c : Nombre de cœurs du processeur ou de la machine.

Exemple : un processeur hexa-cœurs à 2.66 *GHz* (Intel Westmere) est capable d'effectuer 4 opérations flottantes par cycle :
$R_{peak}(DP) = 4 \times 2.66 \times 10^9 \times 6 = 63.8$ *GigaFlops*.

Puissance de calcul d'une machine

Quelques ordres de grandeur...

- PC de bureau : de quelques dizaines à une centaine de GigaFlops(1)

  Exemple pour une machine équipée de 2 processeurs Westmere hexa-cœurs à 2.66 GHz : $R_{peak} = 127.6\ GigaFlops$(1)

- Calculateurs de mesocentre : quelques dizaines de TeraFlops(1) ;

- Premiers supercalculateurs du Top 500 : quelques PetaFlops(1) (2018 : ExaFlops(1)) !

(1) : $Giga/Tera/Peta/ExaFlops$=$10^9/10^{12}/10^{15}/10^{18}$ $Flops$.

Puissance maximale d'une machine

- La puissance crête R_{peak} est une valeur théorique maximale !

  $R_{max} < R_{peak}$ pour un benchmark donné.

- Exemple : Jaguar, machine #3 du Top500 en 2011
  (Cray XT5-HE, processeurs Opteron hexa-cœurs à 2.6 *GHz*)
  $R_{peak} = 2.3$ *PetaFlops* et $R_{max} = 1.7$ *PetaFlops* (Linpack)

- Dans la pratique, la puissance maximale R_{max} dépend de nombreux paramètres et en particulier
  - de la charge de la machine ;
  - du système de fichiers pour les IOs importantes ;
  - des **accès mémoire** (codes de calcul "*memory-bound*") ;
  - ...

Mémoires

Principales caractéristiques :

- Type (SRAM, SDRAM, DDR-SDRAM...) ;
- Capacité ;
- Latence ;
- Type d'accès (séquentiel, direct...).

Capacité mémoire

- Les capacités mémoire ou disque sont exprimées en *octets* (1 *octet* = 8 *bits*).

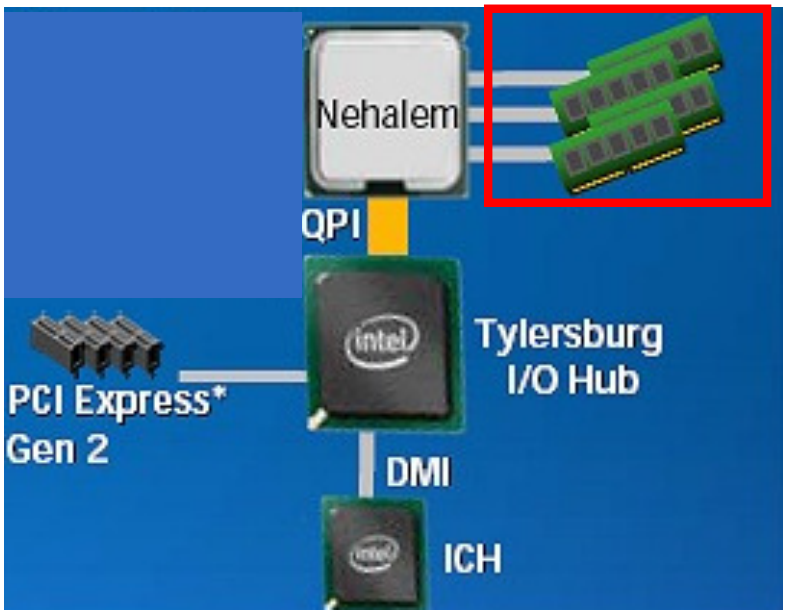
- Unités standardisées par l'IEC depuis 1998 :
  1 Kibioctet (*Kio* ou *KiB*)=2^{10}=1024 *octets*,
  1 Mébioctet (*Mio* ou *MiB*)=2^{20} octets,
  1 Gibioctet (*Gio* ou *GiB*)=2^{30} octets...
  Dans ce système, 1 Kilooctet (*Ko* ou *KB*)=10^3 octets *etc.*

Un PC de bureau a une mémoire centrale de quelques *GiB*, un disque dur d'environ 1 *TiB* (2^{40} octets).

Les plus gros supercalculateurs ont une mémoire vive de quelques centaines de *TiB* et accès à des systèmes de disques de quelques *PiB* (2^{50} octets).

Débit mémoire

- Le débit entre le processeur et la mémoire est exprimé en octets par seconde. Il dépend :
  - des connexions (largeur du bus entre le processeur et la mémoire...) ;
  - du type de mémoire (DDR-SDRAM, QDR..) ;



Exemple :

Processeur Nehalem et RAM DDR3 à 1066 MHz accédée par 3 canaux de largeur 64 $bits$ (8 $octets$) :
$3 \times (8 \times 1066 \times 10^6) = 25.6\ GiB/s$

Latence mémoire

- La latence est le temps passé entre une requête et le début de la réponse correspondante.
- En général, il s'agit du temps qu'il faut attendre pour obtenir des données.
- Elle est exprimée en secondes ou en nombre de cyles (temps de cycle $\approx 0.33\,ns$ pour un processeur à $3\,GHz$).

Hiérarchie mémoire

- Registres : mémoires intégrées au processeur.
  Accès rapide : en général, 1 cycle.

- **Caches** : mémoires intermédiaires (SDRAM) entre le processeur et la mémoire centrale.
  Les caches sont hiérarchisés (L1, L2 et L3).
  Ils peuvent être privés ou partagés entre les cœurs.
  L1 : accédé en quelques cycles, L2 : latence 2 à 10 fois supérieure.

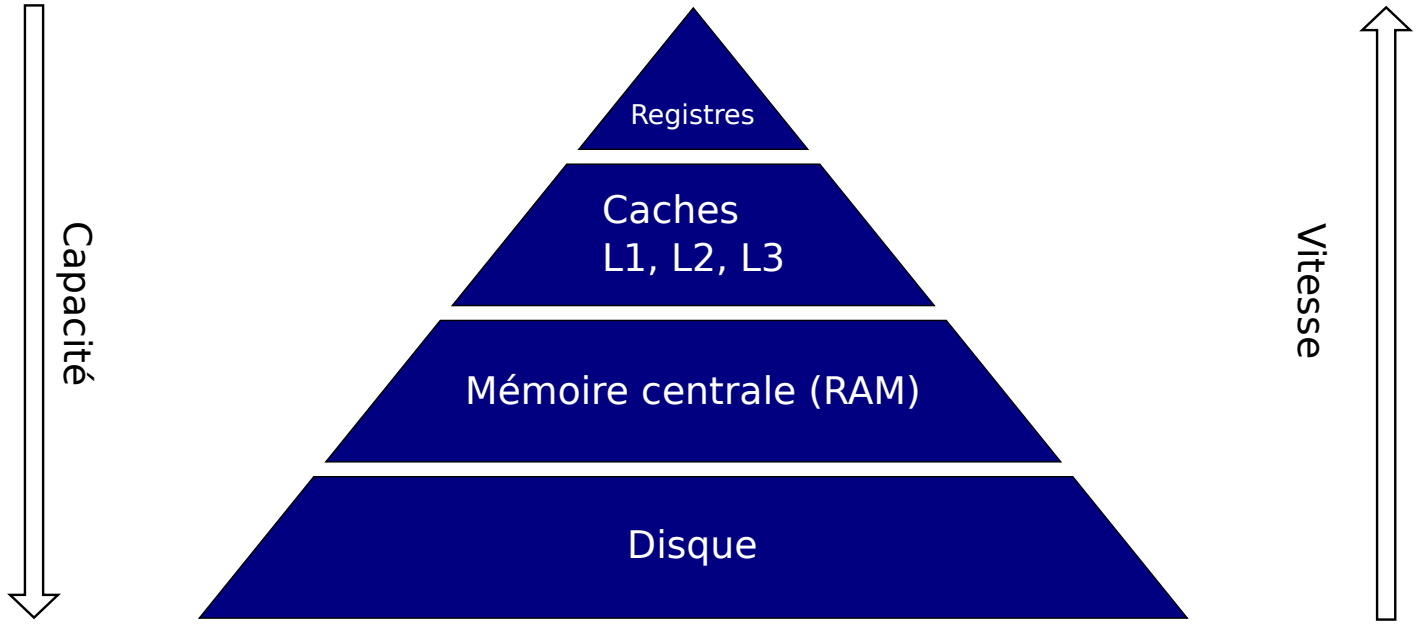
  Exemple, processeur Nehalem : 64 *KiB* de cache L1 par cœurs, 256 *KiB* de cache L2 par cœurs et 8 *MiB* de cache L3 partagé par tous les cœurs.

Hiérarchie mémoire

- **Mémoire centrale** (RAM).
  De la dizaine à quelques centaines de *GiB*.
  Latence de quelques centaines de cyles.

- Disques.
  Généralement plusieurs *TiB*.
  Latence de plusieurs millions de cycles.

Hiérarchie mémoire



Caches

- Le cache est divisé en lignes (ou blocs) de mots.
- Les lignes présentes dans le cache ne sont que des copies temporaires des lignes de la mémoire centrale.
- Le transfert des données de la mémoire centrale vers le cache se fait par lignes.

Caches

- Lorsque le processeur doit accéder à une donnée :
  - soit la donnée se trouve dans le cache ("*cache hit*") : le transfert de la donnée vers les registres se réalise immédiatement.
  - soit la donnée ne se trouve pas dans le cache ("*cache miss*") : une nouvelle ligne est chargée dans le cache depuis la mémoire centrale.
    Pour ce faire, il faut libérer de la place dans le cache et donc renvoyer en mémoire une des lignes, de préférence celle dont la durée d'inactivité est la plus longue.
    Le processeur est en attente pendant toute la durée de chargement de la ligne.

Localité

- Améliorer le taux de succès du cache (éviter les "cache miss") en utilisant les principes de localités spatiale et temporelle.
- **Localité temporelle** :
  Lorsqu'un programme accède à une donnée, il est probable qu'il y accédera à nouveau dans un futur proche.
- **Localité spatiale** :
  Lorsqu'un programme accède à une donnée, il est probable qu'il accédera ensuite aux données voisines.
- Ces principes sont utilisés par :
  - les constructeurs via l'architecture du processeur (*Prefetching*...) ;
  - les programmeurs (*cf.* exemples suivants).

Localité spatiale

- Exemple : accès aux éléments d'une matrice.

Fortran : accès non optimal

```
do i = 1, n
  do j = 1, n
    y(i) = a(i,j) * x(j)
  end do
end do
```

C : accès non optimal

```
for (j=0; j<n; ++j)
{
  for (i=0; i<n; ++i)
  {
     y[i] += a[i][j] * x[j];
  }
}
```

Fortran : accès optimal

```
do j = 1, n
  do i = 1, n
    y(i) = a(i,j) * x(j)
  end do
end do
```

C : accès optimal

```
for (i=0; i<n; ++i)
{
  for (j=0; j<n; ++j)
  {
     y[i] += a[i][j] * x[j];
  }
}
```

Temps sans optimisation du compilateur :

- implémentation non optimal en C : 53.26 s
- implémentation optimal en C : 7.66 s

Localité temporelle

Produit matrice matrice :

```
Pour i variant de 1 à n
  Pour j variant de 1 à n
    Pour k variant de 1 à n
       Aij = Aij + Bik * Ckj
    Fin pour
  Fin pour
Fin pour
```

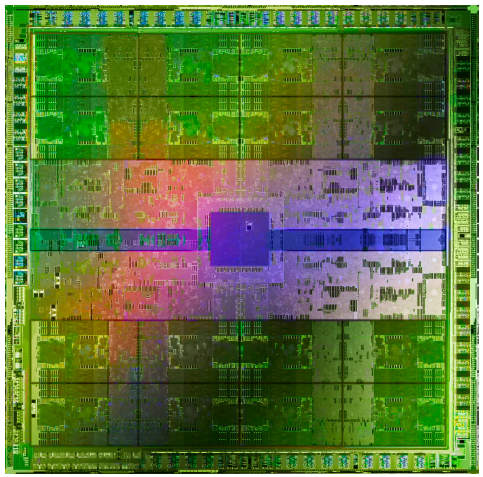
Pour favoriser la localité temporelle, il vaut mieux réaliser les produits par blocs. Par exemple, pour une découpage en $P \times P$ blocs, on écrit :

```
Pour I variant de 1 à P faire
  Pour J variant de 1 à P faire
     AIJ = AIJ + BIK * CKJ
  Fin pour
Fin pour
```

$$\begin{pmatrix} A11 & A12 & \vdots & \vdots & A1J & \vdots & A1P \\ A12 & A22 & \vdots & \vdots & A2J & \vdots & A2P \\ \ldots & \ldots & \ddots & \vdots & \vdots & \vdots & \vdots \\ AI1 & AI2 & \ldots & \ddots & AIJ & \vdots & AIP \\ \ldots & \ldots & \ldots & \vdots & \ddots & \vdots & \vdots \\ \ldots & \ldots & \ldots & \vdots & \ldots & \ddots & \vdots \\ AP1 & AP2 & \ldots & \vdots & APJ & \ldots & APP \end{pmatrix}$$

Accélérateurs : GPU

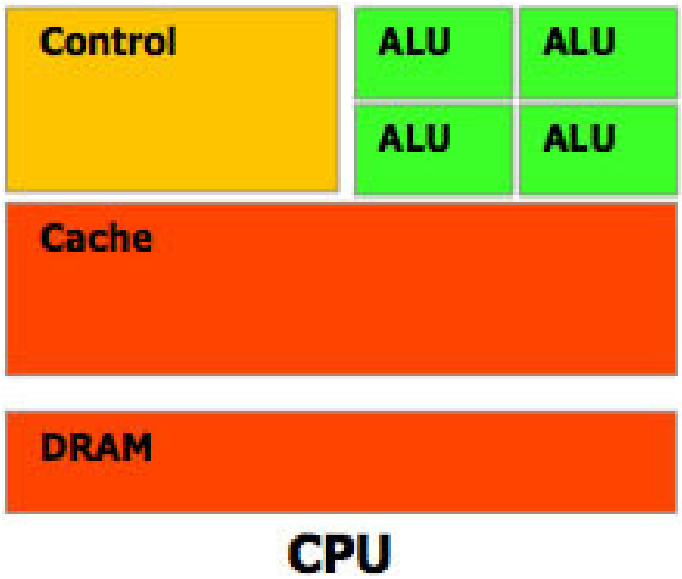
- GPU : *Graphic Processing Unit* : processeur **massivement parallèle, disposant de sa propre mémoire** assurant les fonctions de calcul de l'affichage.

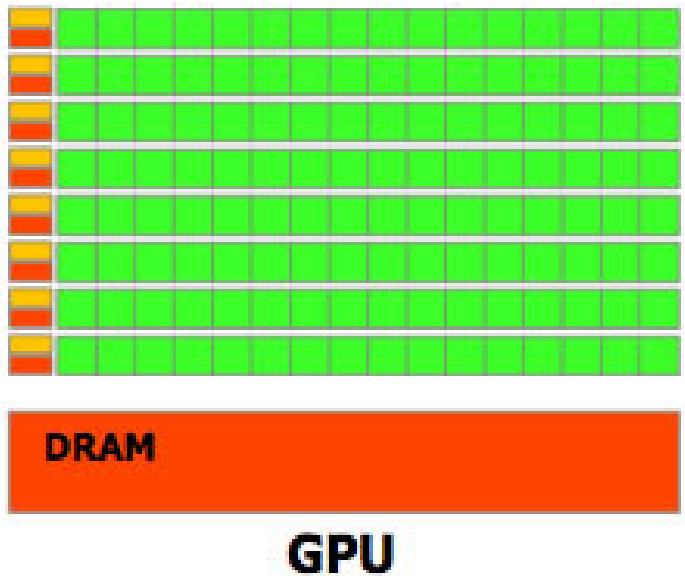


Puce NVidia Fermi et carte Tesla

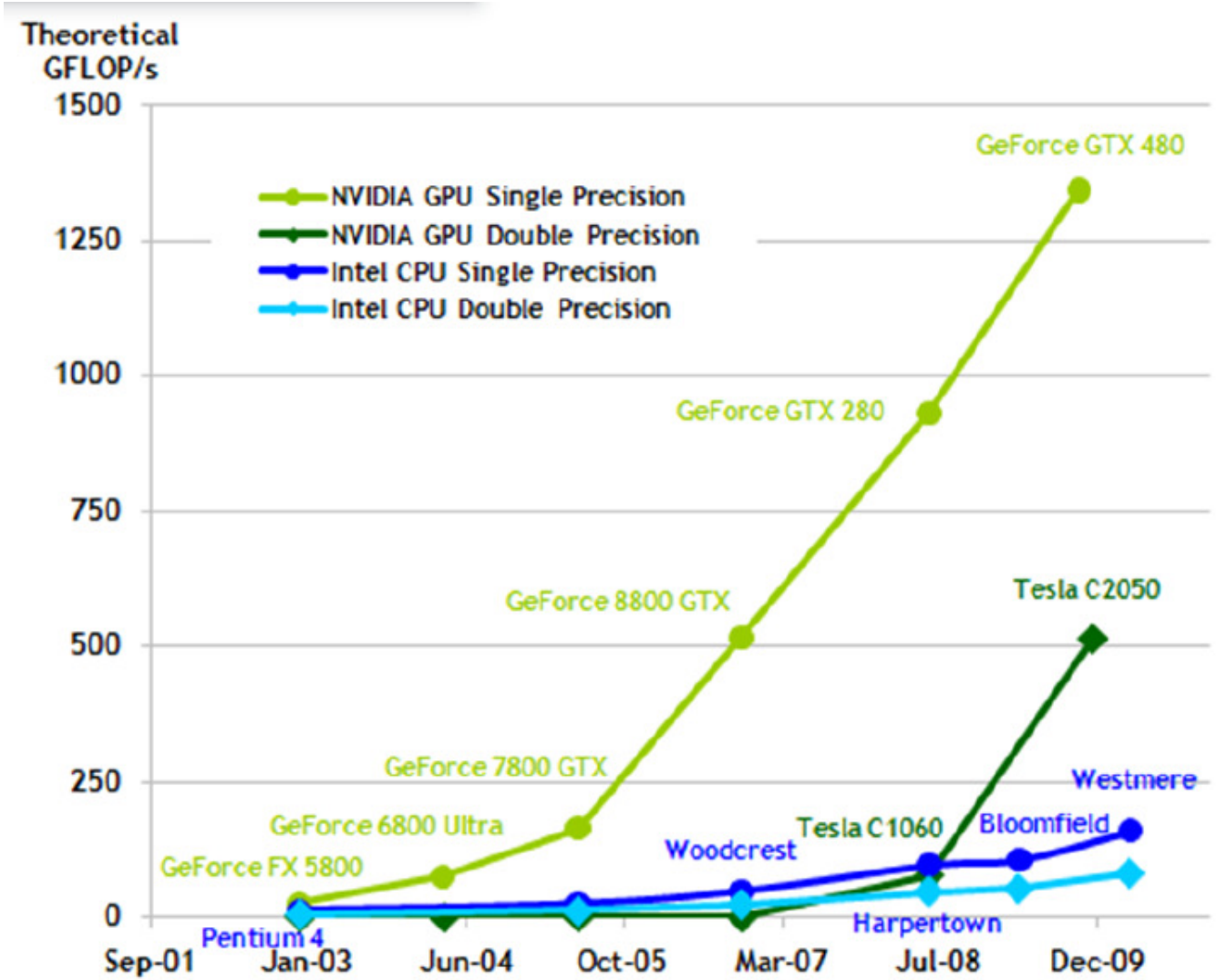
CPU *vs* GPU

- CPU : optimisé pour exécuter rapidement une série de tâches de tout type.
- GPU : optimisé pour exécuter des opérations simples sur de très gros volumes de données.
  $\Rightarrow$ **beaucoup plus de transistors dédiés au calcul sur les GPUs et moins aux unités de contrôle**.





Pourquoi utiliser des GPUs pour le HPC ?



Pourquoi utiliser des GPUs pour le HPC ?

A performance théorique égale, les GPUs sont*, par rapport aux CPUs :

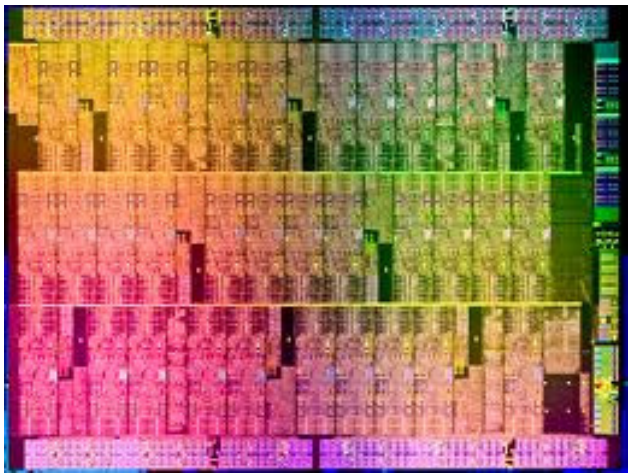
- des solutions plus denses (9× moins de place) ;
- moins consommateurs d'électricité (7× moins) ;
- moins chers (6× moins).

∗ Chiffres issus de "*High Performance Computing : are GPU going to take over ?*", A. Nedelcoux, Université du SI, 2011.

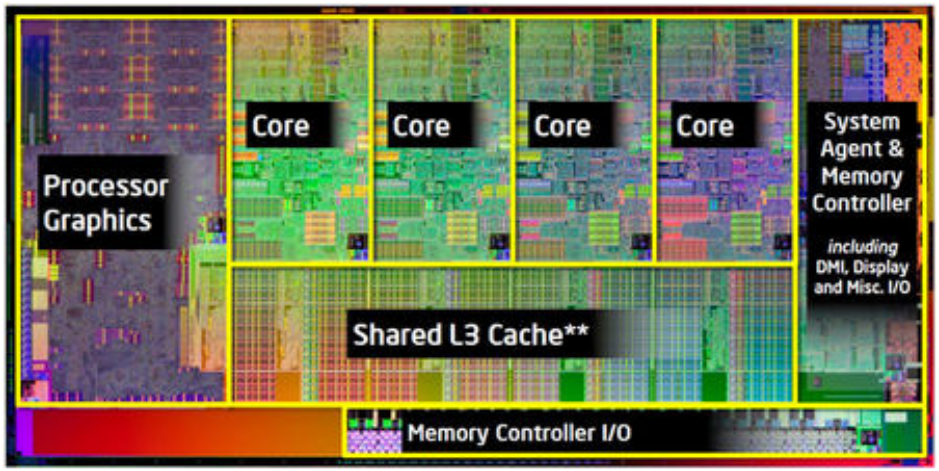
Accélérateurs : nouvelles tendances

En réponse aux GPUs :

- Processeurs *manycores* : Knights Corner (plus de 50 cœurs) d'Intel utilisant l'architecture MIC (*Many Integrated Core*) ;
- Processeurs hybrides (GPU + CPU) : Sandy-Brigde chez Intel et Fusion chez AMD.



Knights Corner



Sandy-Bridge

Architectures de calculateurs parallèles

Les différentes architectures de distinguent selon l'organisation de la mémoire :

- architectures à mémoire partagée
- architectures à mémoire distribuée
- architectures mixtes
- architectures hybrides (GPU)

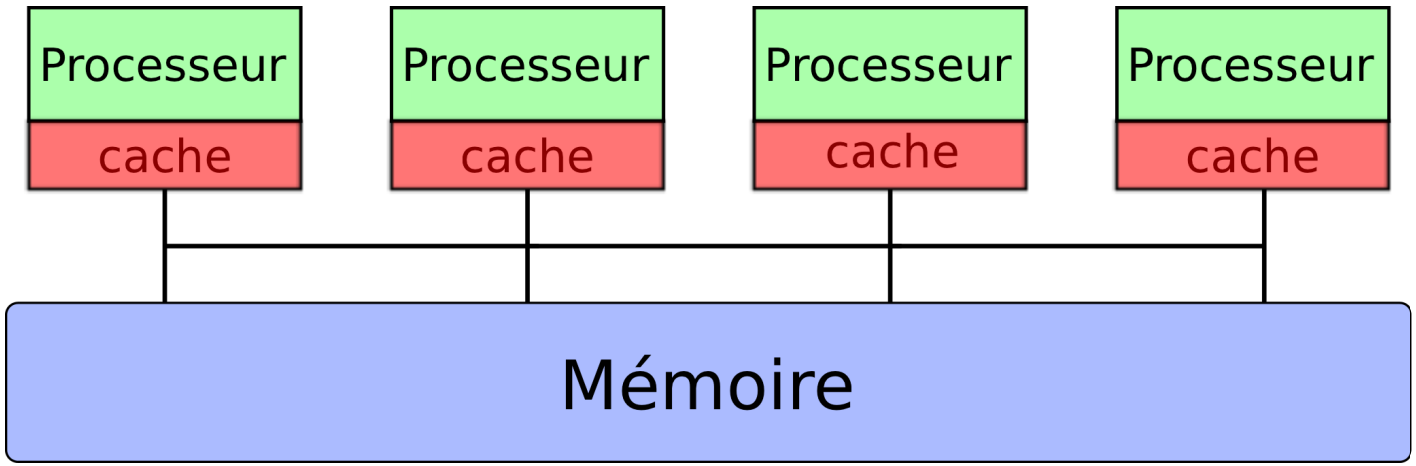
Architectures à mémoire partagée

- Un espace mémoire global visible par tous les processeurs.
- Les processeurs ont leur propre mémoire cache dans laquelle est copiée une partie de la mémoire globale.
- On distingue deux classes d'architecture à mémoire partagée :
  - UMA : Uniform Memory Access
  - NUMA : Non Uniform Memory Access

Architecture à mémoire partagée : UMA

Architecture UMA (*Uniform Memory Access*) :

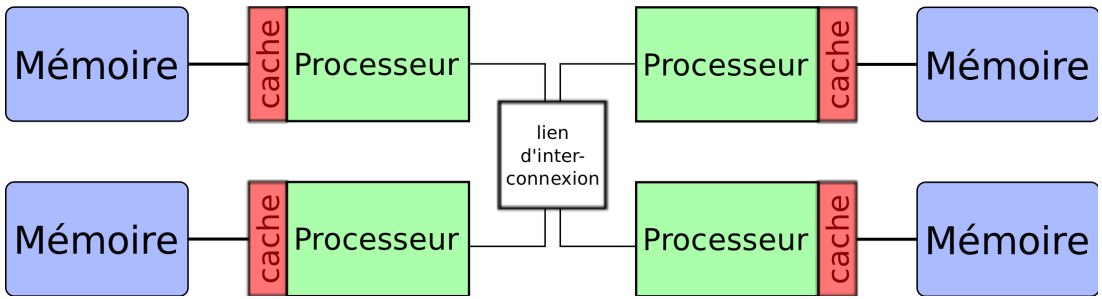
- Une seule mémoire centralisée.
- Le temps d'accès à un emplacement quelconque en mémoire est le même pour tous les processeurs.



Machine à mémoire partagée : NUMA

Architecture NUMA (*Non-Uniform Memory Access*) :

- Une mémoire centrale par processeur (ou par bloc UMA de processeurs).
- Les processeurs sont capables d'accéder à la totalité de la mémoire du système mais les temps d'accès mémoire sont non uniformes.
- Le temps d'accès via le lien d'interconnexion est plus lent que le temps d'accès direct vers la mémoire centrale.

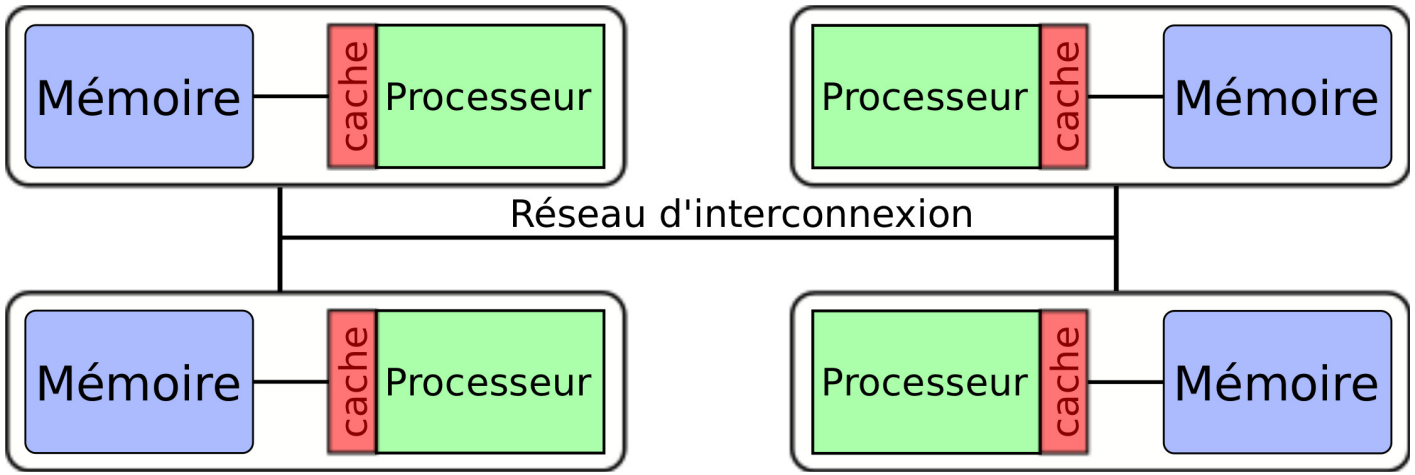


Architecture à mémoire partagée : remarques

- La gestion de la cohérence de cache est le plus souvent transparente pour l'utilisateur (CC-UMA, CC-NUMA).
- L'espace d'adresse global, visible de tous les processeurs, facilite la programmation parallèle.
- L'échange des données entre les tâches attribuées aux processeurs est transparent et rapide.
- Le nombre de processeurs par machine est limité.
- Les performances décroissent vite avec le nombre de processeurs et/ou cœurs utilisés pour une application à cause du traffic sur le chemin d'accès des données en mémoire.
- Machine coûteuse lorsque le nombre de processeurs devient important.

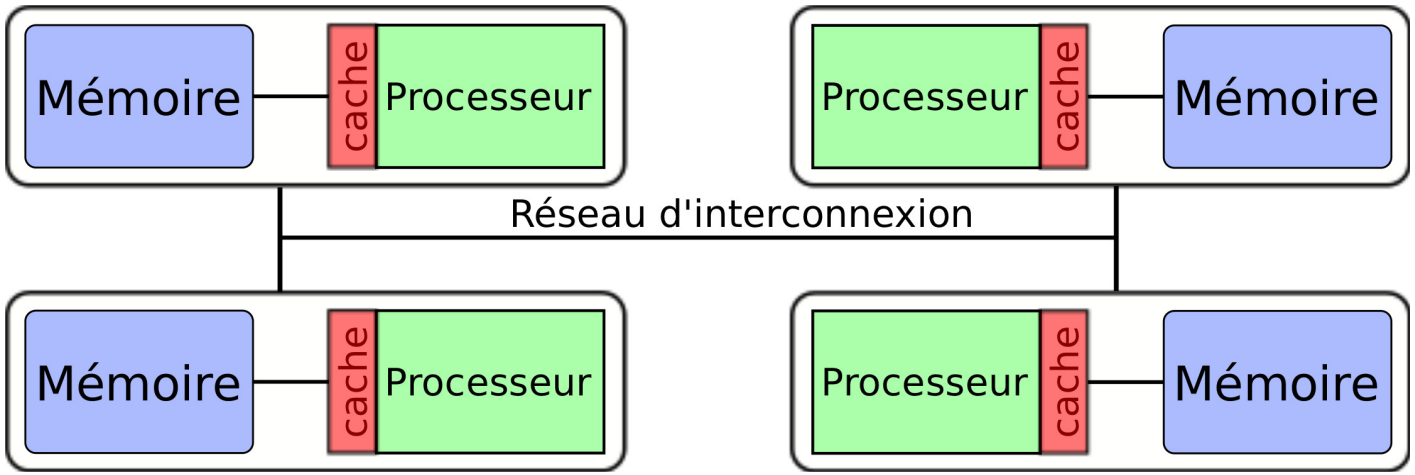
Architectures à mémoire distribuée

- Un espace mémoire est associé à chaque processeur.
- Les processeurs sont connectés entre eux à travers un réseau.



Architecture à mémoire distribuée

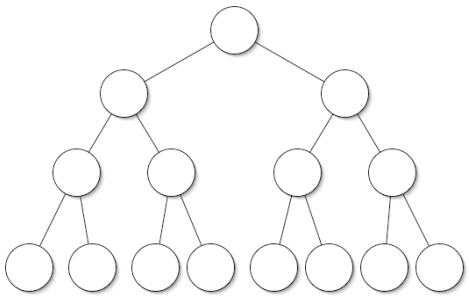
- L'accès à la mémoire du processeur voisin doit se faire explicitement par échange de messages à travers le réseau d'interconnexion entre les processeurs.



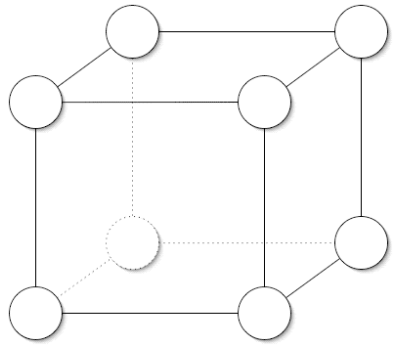
Architecture à mémoire distribuée : Réseau d'interconnexion

Le réseau d'interconnexion est important sur ces architectures car il détermine la vitesse d'accès aux données d'un processeur voisin. Les caractéristiques du réseau sont :

- sa latence : temps pour initier une communication ;
- sa bande passante : vitesse de transfert des données à travers le réseau ;
- sa topologie : architecture physique du réseau.



Topologie en arbre



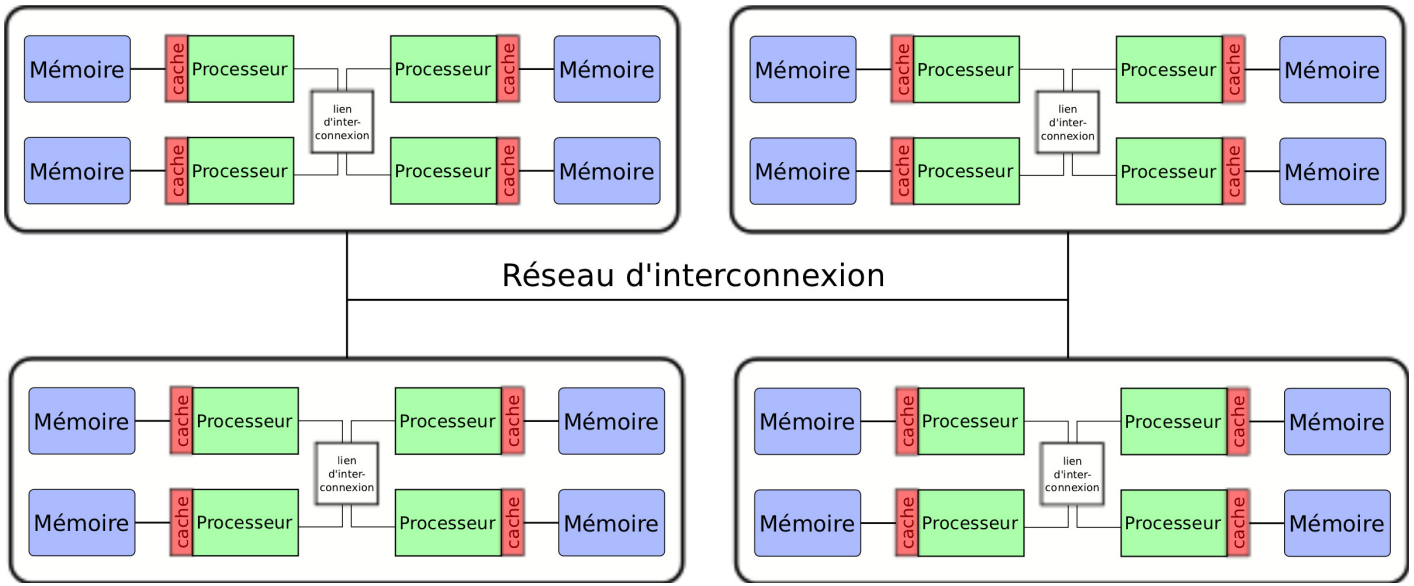
Topologie en hypercube

Architecture à mémoire distribuée : remarques

- Accès rapide à la mémoire locale
- L'architecture permet facilement d'obtenir des machines avec un grand nombre de processeurs et ce, pour un coût réduit par rapport à l'architecture à mémoire partagée
- L'échange de données entre processeurs doit être géré par le programmeur. Un effort de développement est nécessaire pour utiliser ce type d'architecture
- Les performances sont dépendantes de la qualité du réseau d'interconnexion (Infiniband, Gigabit Ethernet, Myrinet...)

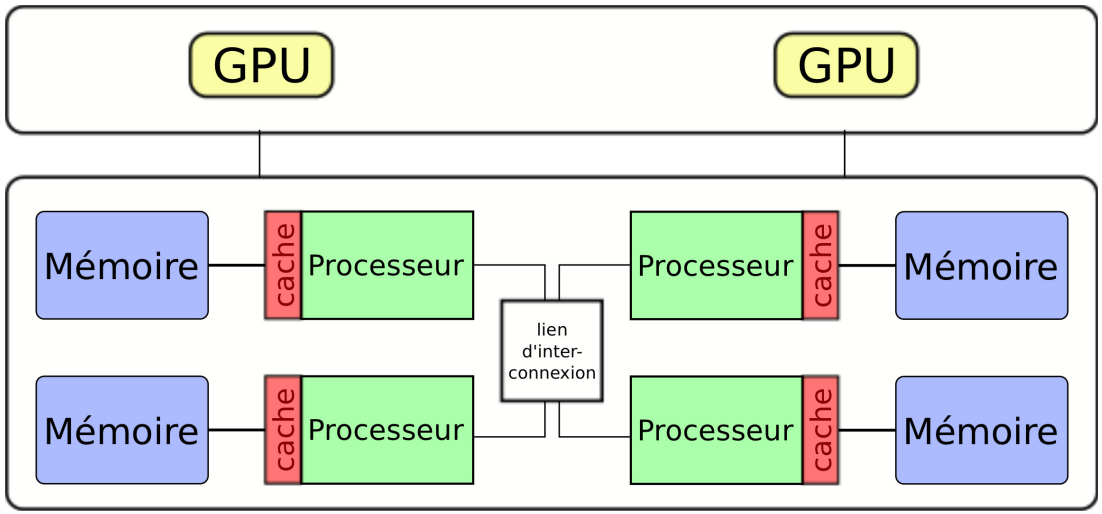
Architecture mixte

- Aujourd'hui, la plupart des calculateurs combinent les architectures à mémoire partagée et à mémoire distribuée.
- Ces machines sont alors constituées de machines à mémoire partagée (nœud de calcul) reliées entre elles par un réseau d'interconnexion :



Architecture hybride

- Avec l'essor des accélérateurs graphiques, de plus en plus de calculateurs possèdent des nœuds de calcul équipés de GPU.



Modèle de programmation parallèle

A chaque architecture de machine parallèle correspond un modèle de programmation :

- modèle de programmation à mémoire partagée
- modèle de programmation à mémoire distribuée
- modèle de programmation mixte : utilisation combinée des deux modèles précédents
- modèle de programmation hybride : utilisation des GPUs

Programmation à mémoire partagée : multithreading

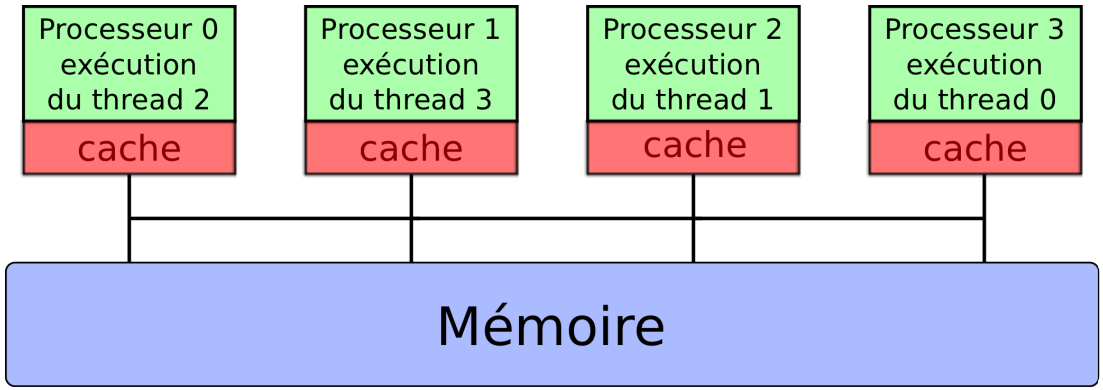
Pour ce type de modèle, on utilise le plus souvent le modèle **multithreading** :

- Un programme multithreading s'exécute dans un processus unique.
- Ce processus active plusieurs processus légers (appelés également *threads*) capables de s'exécuter de manière concurrente.
- L'exécution de ces processus légers se réalise dans l'espace mémoire du processus d'origine.
- Chaque thread possède un espace mémoire local invisible des autres threads.

Programmation à mémoire partagée : multithreading

- C'est l'exécution concurrente des threads sur plusieurs processeurs ou cœurs qui permet l'exécution parallèle du programmme.
- Le système d'exploitation distribue les threads sur les différents processeurs ou cœurs de la machine à mémoire partagée.

Exemple : distribution de 4 threads sur une machine à mémoire partagée de 4 processeurs de type UMA.



Programmation à mémoire partagée : OpenMP

OpenMP (*Open Multi-Processing*) est une **interface de programmation** (API) pour générer un **programme multithreads**

- L'interface de programmation fournit :
  - des directives de compilation ;
  - des sous-programmes ;
  - des variables d'environnement.
- Cette API est :
  - disponible pour le Fortran, le C et le C++ ;
  - supportée par de nombreux systèmes et compilateurs (gnu, intel...).

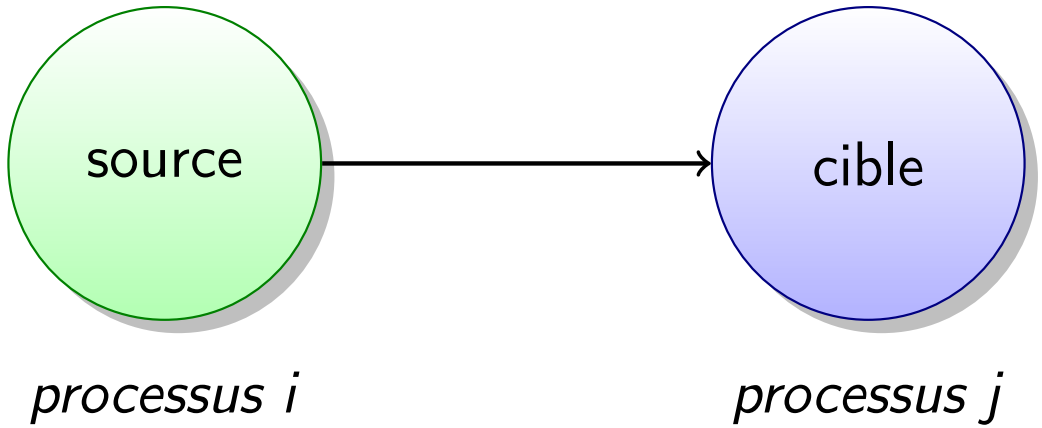
Programmation à mémoire distribuée : échange de messages

Pour ce type de modèle, on utilise le plus souvent le modèle **échanges de messages** :

- Chaque processus exécute un programme sur des données différentes.
- Chaque processus dispose de ses propres données, sans accès direct à celles des autres.
- Les processus peuvent s'échanger des messages entre eux pour :
  - transférer des données
  - se synchroniser

Concepts d'échange de messages

- Un processus *i* envoie des données *source* au processus *j*.
- Le processus *j* reçoit des données du processus *i* et les met dans *cible*.



Programmation à mémoire partagée : MPI

MPI (*Message Passing Interface*) est une **interface de programmation** (API) pour générer un **programme par échange de messages**.

- L'interface de programmation fournit des fonctions permettant de gérer :
  - un environnment de d'exécution ;
  - les communication point à point ;
  - les communication collectives ;
  - des groupes de processus ...
- Il existe plusieurs implémentations de cette API :
  - développées par les constructeurs pour leur plate-formes ;
  - disponible en open source (OpenMPI, Mpich2, ...) ;
- La bibliothèque est disponible en Fortran, le C et le C++ ;

Modèle de programmation mixte

- Ce modèle consiste à utiliser deux niveaux de parallélisme :
  - le modèle de parallélisation pour architecture à mémoire distribuée pour réaliser des tâches peu dépendantes en parallèle sur les différents nœuds du calculateur ;
  - le modèle de parallélisation pour architecture à mémoire partagée sur chaque nœud du calculateur.
- Le principe consiste alors à faire du multithreading avec OpenMP à l'intérieur de chaque processus MPI.

Modèle de programmation hybride

- Ce modèle consiste à confier aux GPUs une partie des calculs.
- Outils pour utiliser les GPUs
  - CUDA
  - OpenCL
  - HMPP ...

Accélération et efficacité

L'**accélération** (*speed-up*) $A(N)$ et l'**efficacité** $E(N)$ constituent deux mesures de la qualité de la parallélisation.

Soient t_1 et t_N les temps d'exécution respectifs sur 1 et N processeurs et/ou de cœurs

- Accélération $A(N)$ est défini par :

$$A(N) = t_1/t_N \quad (N = 1, 2, 3, ...)$$

- Efficacité $E(N)$ est défini par :

$$E(N) = A(N)/N \quad (N = 1, 2, 3, ...)$$

- L'accélération et l'efficacité parfaites sont obtenues pour $t_N = t_1/N$:

$$A(N) = N \quad et \quad E(N) = 100\%$$

Extensibilité

L'**Extensibilité** (*scalability*) est la qualité des programmes à rester efficace pour un grand nombre de processeurs et/ou de cœurs.
L'extensibilité d'un programme est pénalisée par :

- L'overhead dû aux communications
- La répartition plus ou moins equilibrée des tâches entre les processeurs et/ou de cœurs (Load-Balancing)
- La fraction de code parallélisé (loi d'Amdhal)

Loi d'Amdahl

Enoncé

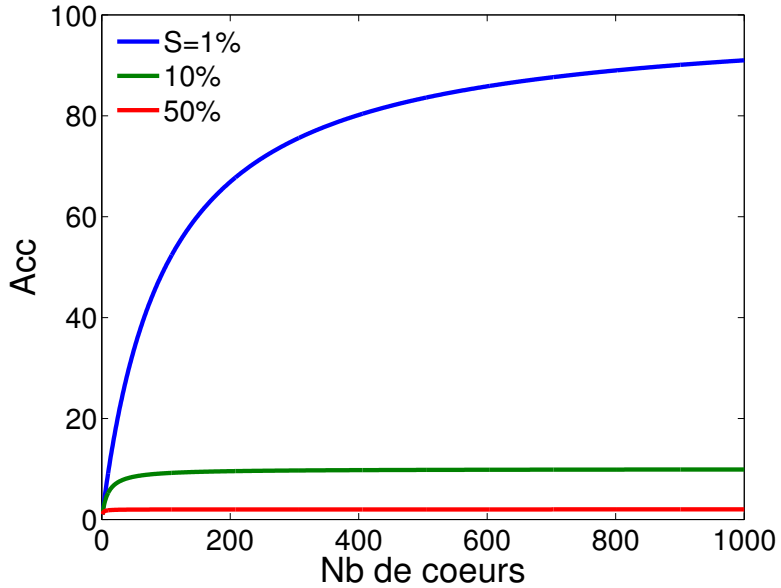
Accélération théorique maximale obtenue en parallélisant idéalement un code, pour un problème donné et une taille de problème fixée :

$$A(N) = \frac{t_1}{t_N} = \frac{(S+P)t_1}{(S+P/N)t_1} = \frac{1}{S+P/N}$$

- t_1 (resp. t_N) est le temps d'éxecution sur 1 (resp. N) cœurs
- P est la fraction parallèle du programme
- $S = 1 - P$ est la fraction séquentielle du programme

Loi d'Amdahl

- $A(N) < 1/S$: **l'accélération est limitée par la partie non-parallélisée du code**.
- Exemple, pour $S = 50\%$, l'accélération maximale vaut 2, pour $S = 10\%$, elle vaut 10 et 100 pour $S = 1\%$.



Références

- *Calcul Haute Performance, architectures et modèles de programmation*, F. Roch, ANGD "Calcul parallèle et application aux plasmas froids", 2011.
- *Architectures massivement parallèles. Introduction à la Blue Gene/P d'IBM*, P. Wautelet, Cours Idris donné à l'Ecole Centrale Paris, 2011.
- *Architecture des calculateurs*, V. Louvet, Formation en Calcul Scientifique, LEM2I, 2011.
- *Modèles de programmation, Introduction au calcul parallèle*, G. Moebs, ANGD "Choix, installation et exploitation d'un calculateur", 2011