Ecole Doctorale MathIf

Architecture des ordinateurs, concepts du parallélisme

Violaine Louvet [1]

Remerciements à Françoise Roch, Guy Moebs, Françoise Berthoud

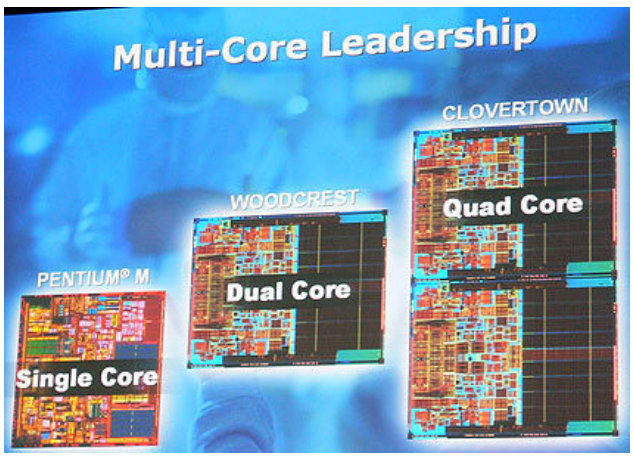
[1] ICJ - CNRS

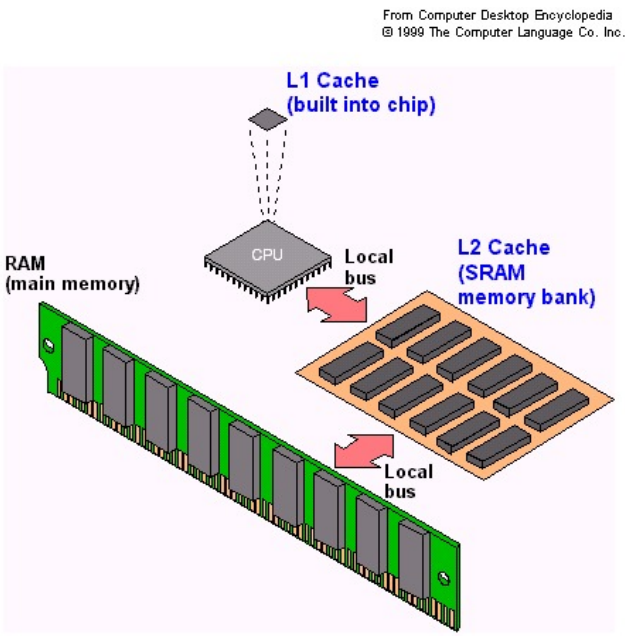
Année 2009-2010

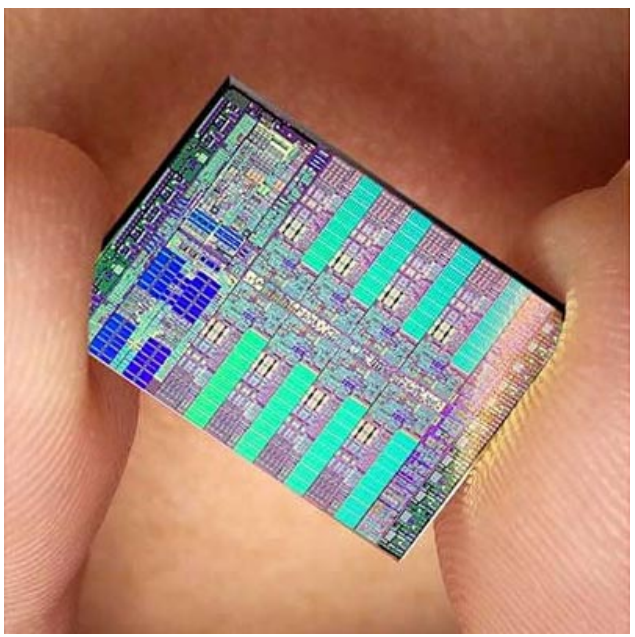
Objectifs de ce cours

Décoder la relation entre l'architecture et les applications :

- Adapter les algorithmes, la programmation,
- Comprendre le comportement d'un programme,
- Choisir son infrastructure de calcul en fonction de ses besoins.



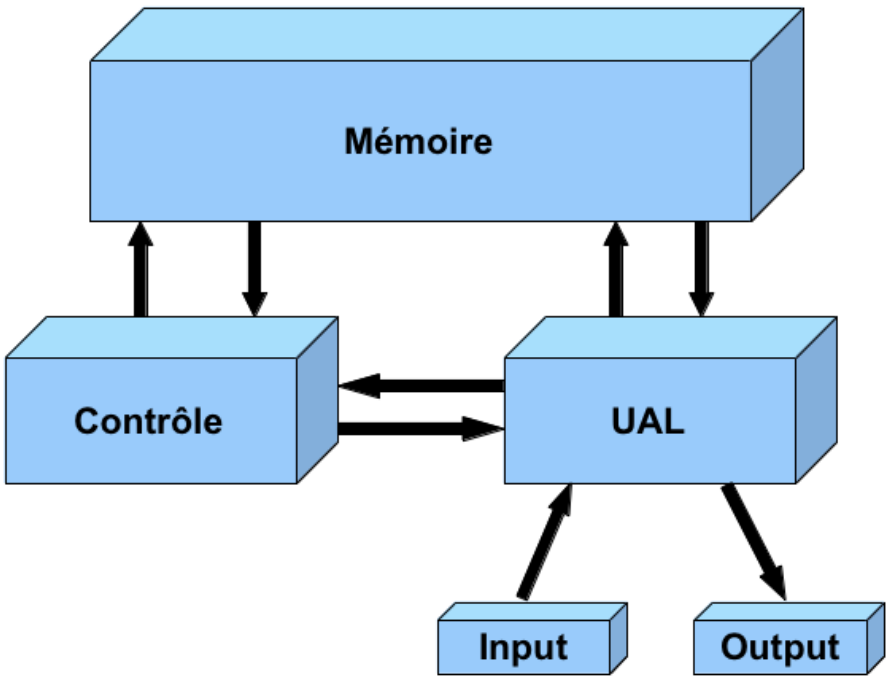




Sommaire

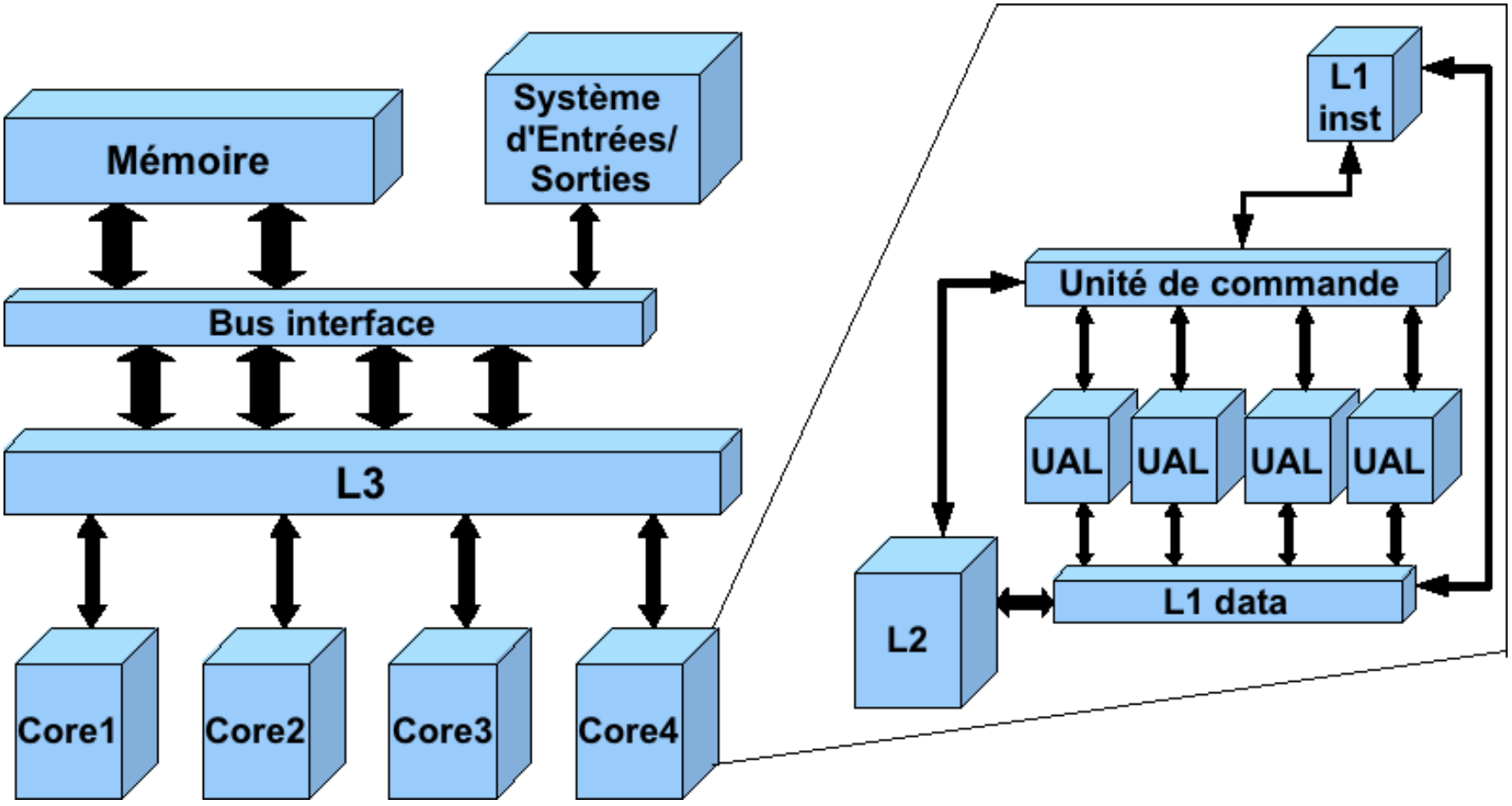
Sommaire

Modèle de Von Neumann (1945)



- La mémoire : contient le programme (instructions) et les données.
- Une Unité Arithmétique et Logique : UAL qui effectue les opérations.
- Une unité de contrôle chargée du séquençage des opérations.
- Une unité d'Entrée/Sortie.

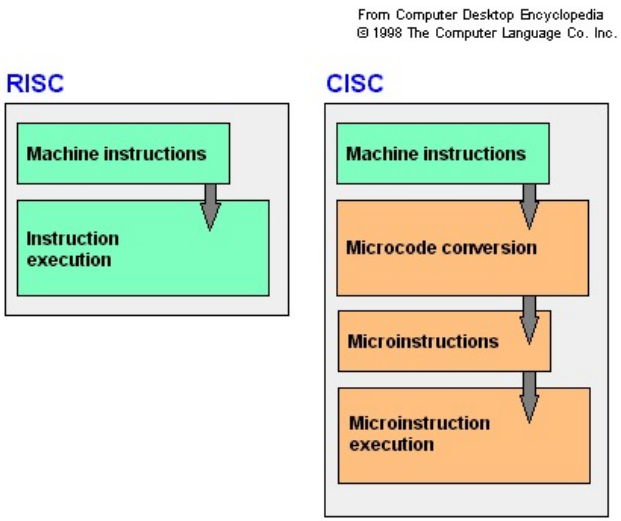
Architecture actuelle



Sommaire

Caractéristiques d'un processeur

- Jeux d'instructions qu'il peut exécuter :
  - complexe : CISC (beaucoup d'instructions complexes mais prenant plusieurs cycles d'horloge). *Exemple : x86 (processeurs compatibles Intel)*.
  - réduit : RISC (moins d'instructions mais n'utilisant que quelques cycles d'horloge). *Exemple : processeurs PowerPC*.



- Complexité de son architecture (nombre de transistors, plus il contient de transistors, plus il peut exécuter d'instructions en une seconde).

Caractéristiques d'un processeur

- Nombre de bits pouvant être traités en une instruction, 32 ou 64 bits.
  - Définit la taille des registres généraux du processeur (emplacement mémoire interne du processeur) pour les nombres entiers.
  - En 64 bits, les entiers et les adresses passent de 32 bits (4 octets) à 64 bits (8 octets).
- Vitesse maximale de l'horloge (plus elle augmente, plus le processeur complète d'instructions en une seconde).
  - La fréquence d'horloge détermine la durée d'un cycle.
  - Chaque opération utilise un certain nombre de cycles.
  - La fréquence d'horloge est fonction de :
    - la technologie des semi-conducteurs,
    - le packaging,
    - les circuits.

Composants d'un processeur

1 CPU = plusieurs unités fonctionnelles :

- une unité de gestion des bus (unité d'entrées-sorties) en interface avec la mémoire vive du système,
- une unité d'instruction (control unit) qui lit les données arrivant, les décode et les envoie à l'unité d'exécution,
- une unité d'exécution qui accomplit les tâches que lui a données l'unité d'instruction, composée notamment de :

  une ou plusieurs unités arithmétiques et logiques (UAL) qui assurent les fonctions basiques de calcul arithmétique et les opérations logiques.

  une ou plusieurs unités de virgule flottante (FPU) qui accomplissent les calculs complexes. L'instruction de calcul de base est la multiplication / addition (FMA pour Floating-point Multiply and Add) en double précision.

Etapes d'exécution d'une opération

Une opération s'exécute en plusieurs étapes indépendantes par des éléments différents du processeur.

- IF : Instruction Fecht, charge l'instruction à exécuter dans le pipeline.
- ID : Instruction Decode/Register Fecht, décode l'instruction et adresse les registres.
- EX : Execution/Effective Address, exécute l'instruction (par la ou les unités arithmétiques et logiques).
- MA : Memory Access/ Cache Access, dénote un transfert depuis un registre vers la mémoire dans le cas d'une instruction du type STORE (accès en écriture) et de la mémoire vers un registre dans le cas d'un LOAD (accès en lecture).
- WB : Write-Back, stocke le résultat dans un registre.

Dans un micro-processeur sans pipeline, les instructions sont exécutées les unes après les autres. En supposant que chaque étape met 1 cycle d'horloge pour s'exécuter, il faut normalement 5 cycles pour exécuter une instruction, 15 pour 3 instructions.

Exécution simultanée d'instructions : ILP (pipelining)

Instruction Level Parallelism

Exécuter simultanément les différentes étapes sur des données distinctes (parallélisme d'instructions).

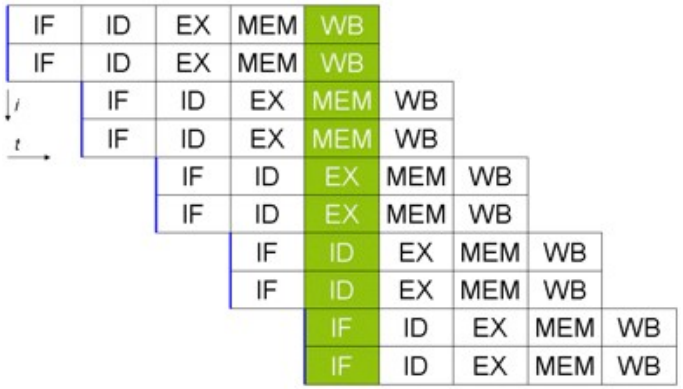
5 instructions en parallèle en 9 cycles (25 cycles en séquentiel) $\longrightarrow$ permet de multiplier le débit avec lequel les instructions sont exécutées par le processeur.

Chaque étape n'a pas la même durée. La plus longue (MA) détermine la latence (délai entre 2 instructions entrant consécutivement dans le pipeline.

Instruction Level Parallelism : architectures superscalaires

Principe

Duplication de composants (FMA, FPU) et exécution simultanée de plusieurs instructions.



10 instructions en 9 cycles

- Gestion des instructions :

  Statique (in-order) : exécutées dans l'ordre du code machine.
  Dynamique (out-of-order) : le hardware modifie l'ordre des instructions pour favoriser le parallélisme.

- Exécution spéculative, prédiction de branchement : faire une hypothèse sur la suite d'un traitement après un branchement conditionnel (lorsque la valeur de la condition n'est pas encore calculée).
- Prefetching : anticiper les données et les instructions dont le processeur aura besoin, et ainsi les précharger depuis la hiérarchie mémoire.

Les problèmes de dépendances entre instructions :

- Partage des ressources : par exemple la mémoire.
- Dépendances des données entre instruction : une instruction produit un opérande utilisé immédiatement par l'instruction suivante.
- Dépendances des contrôles : une instruction est une branche.

Appliquer la même instruction à plusieurs données : SIMD

Le mode SIMD (Single Instruction on Multiple Data) permet d'appliquer la même instruction simultanément à plusieurs données (stockées dans un registre) pour produire plusieurs résultats.

- Soit physiquement installé dans les processeurs.
- Soit simulé par des instructions de type vectoriel de bas niveau.

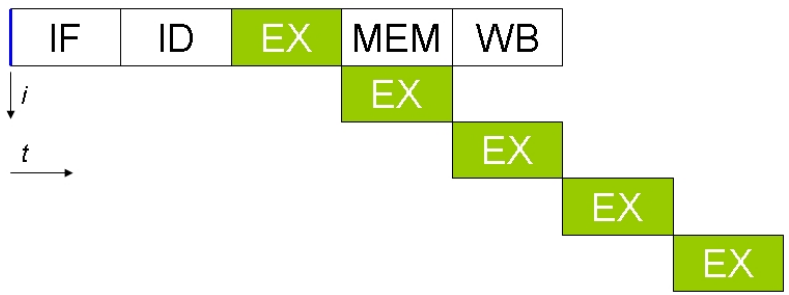
Exemples de jeux d'instructions SIMD

- Sur processeur x86 : MMX (MultiMedia eXtensions), 3DNow, SSE (Streaming SIMD Extensions).
- Sur processeur PowerPC : AltiVec.

- Adaptées aux traitements réguliers, comme le calcul matriciel sur matrices pleines ou le traitement d'images.

Les architectures vectorielles

- De nombreux problèmes sont intrinsèquement vectoriels : ils opèrent sur des vecteurs de données unidimensionnels.
- Certaines architectures disposent d'instructions vectorielles :
  - l'instruction vectorielle est décodée une seule fois, les éléments du vecteur sont ensuite soumis un à un à l'unité de traitement.
  - Les pipelines des unités de traitement sont pleinement alimentés.
- Exemples : Cray, NEC SX, Fujitsu VPP, Altivec (PowerPC G4 et G5)



Les limites du parallélisme d'instructions/données

Obstacles à la performance sur les systèmes superscalaires standard :

- les branchements (IF, CASE, ...) :
  - les mauvaises prédictions : en cas d'erreur de prédiction, il faut revenir en arrière,
  - les petites branches ont peu de code à exécuter ce qui limite le parallélisme.
- La latence de la mémoire :
  - utilise plus d'un cycle d'horloge.
- L'extraction du parallélisme des instructions :
  - le compilateur sérialise le code, dont le parallélisme intrinsèque doit être redécouvert dynamiquement par le processeur.

Dépasser les limites en « aidant »le processeur

Les architectures VLIW (Very Long Instruction Word) et EPIC (Explicitly Parallel Instruction Set Computing) permettent de traiter des instructions longues, agrégations d'instructions courtes indépendantes de façon à les exécuter explicitement en parallèle.

- VLIW : Le compilateur a la charge d'organiser correctement les instructions parmi les bundles (blocs formés de plusieurs instructions successives), tout en respectant les types de dépendances habituelles qui sont normalement gérées au niveau matériel par les architectures classiques.

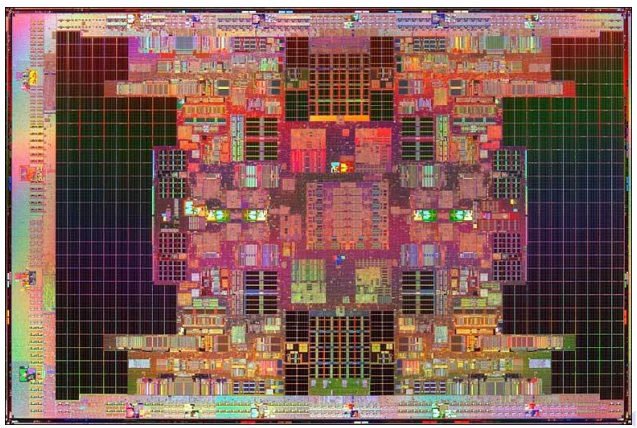
  $\longrightarrow$ On gagne en performance (pas de contrôle), mais la compatibilité binaire entre générations successives est quasi-impossible. Le code généré est spécifique à une implémentation de l'architecture.

Dépasser les limites en « aidant »le processeur

- EPIC : type d'architecture de microprocesseurs utilisée notamment dans les Itaniums. Le parallélisme est exprimé de manière indépendante de la mise en oeuvre du processeur. Disparition du réordonnancement à l'exécution : les instructions sont exécutées dans l'ordre exact dans lequel le compilateur les a mis.

  $\longrightarrow$ l'effort d'optimisation repose sur le compilateur qui a la charge d'organiser statiquement les dépendances inter-instructions.

Le rôle du compilateur est essentiel : il ne faut pas s'étonner de pertes de performances importantes sur ce type d'architecture si le compilateur n'est pas adapté !



TLP, Thread Level Parallelism

Améliorer le remplissage du flot d'instructions du processeur

Idée : mixer deux flux d'instructions arrivant au processeur pour optimiser l'utilisation simultanée de toutes les ressources (remplir les cycles perdus - cache miss, load ...).

Le SMT (Simultaneous Multithreading) partage du pipeline du processeur entre plusieurs threads(d'un même programme ou de deux programmes différents). Les registres et les caches sont aussi partagés.

L'hyperthreading = SMT d'Intel.

Le multithreading dégrade les performances individuelles des threads mais améliore les performances de l'ensemble.

Sommaire

Les différentes technologie de mémoire

- Principalement deux types à base de semi-conducteur :

  DRAM (Dynamic Random Access Memory) chaque bit est représenté par une charge électrique qui doit être rafraîchie à chaque lecture/écriture.

  SRAM (Static Random Access Memory) retient ses données aussi longtemps qu'il y a du courant.
- Temps d'un cycle SRAM de 8 à 16 fois plus rapide qu'un cycle DRAM.
- Coût de la mémoire SRAM de 8 à 16 fois plus élevé que la mémoire DRAM.

Solution la plus couramment choisie :

1. De 1 à 3 niveaux de SRAM en mémoire cache.
2. La mémoire principale en DRAM.
3. La mémoire virtuelle sur des disques

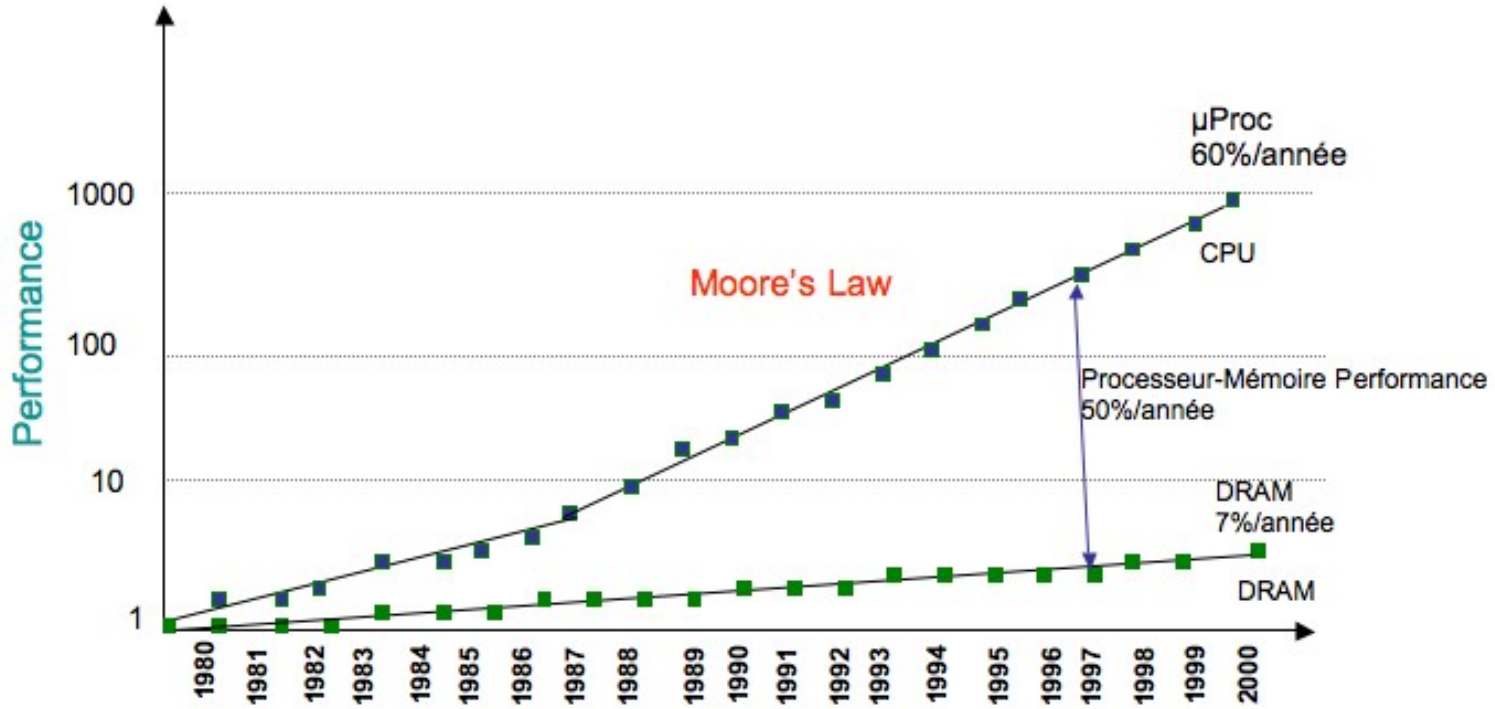
Type	Taille	Vitesse	Coût/bit
Registre	< 1 KB	< 1 ns	$$$$
SRAM On-chip	8 KB - 6 MB	< 10 ns	$$$
SRAM Off-chip	1 MB - 16 MB	< 20 ns	$$
DRAM	64 MB - 1 TB	< 100 ns	$
Flash	64 MB - 32 GB	< 100 μs	c
Disk	40 GB - 1 PB	< 20 ms	$\sim$0

Hiérarchisation de la mémoire

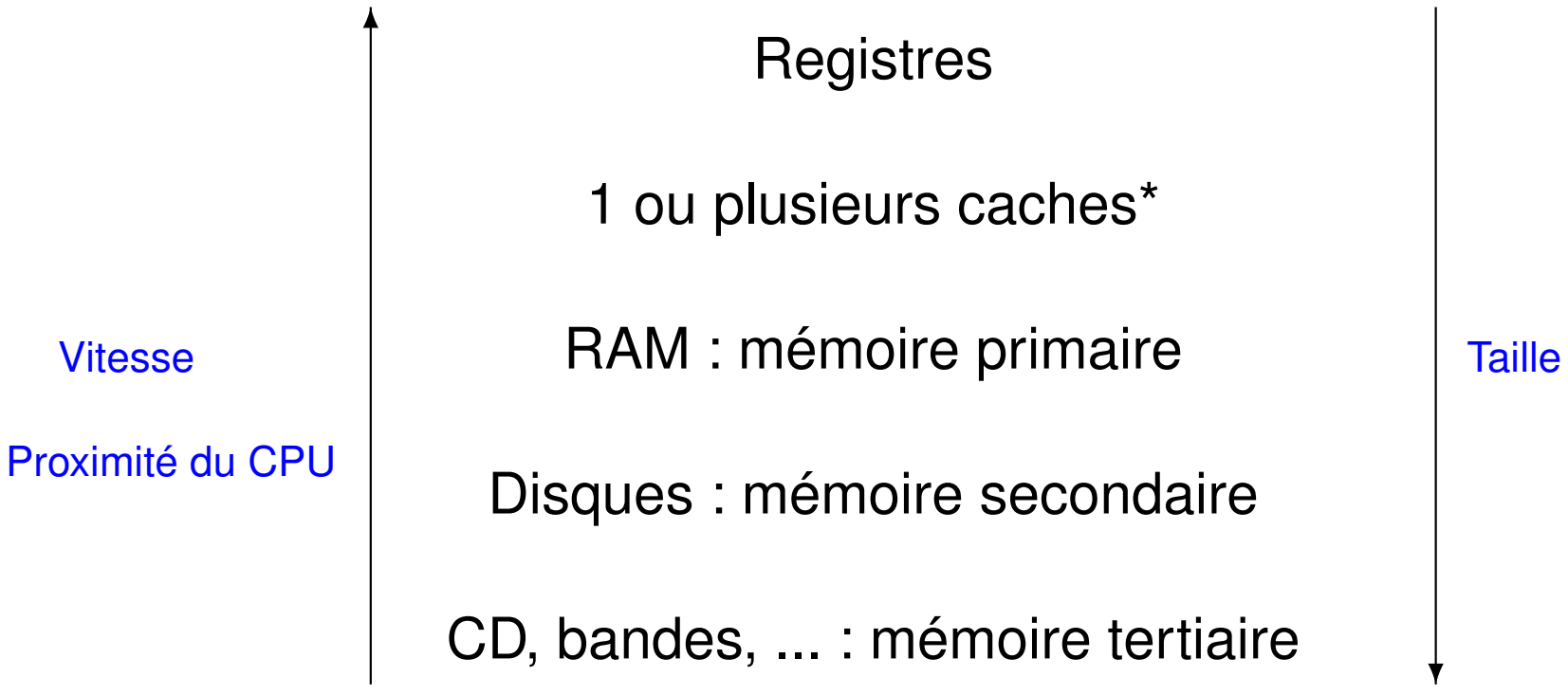
Cas idéal : Mémoire la plus grande et la plus rapide possible

La performance des ordinateurs est limitée par la latence et la bande passante de la mémoire :

- Latence = temps pour un seul accès.
  Temps d'accès mémoire $\gg$ temps cycle processeur.
- Bande passante = nombre d'accès par unité de temps.



Hiérarchisation de la mémoire



* les caches peuvent être organisés de façon hiérarchiques

Notion de localité mémoire

- Localité temporelle : si une zone est référencée, elle a des chances d'être référencée à nouveau dans un futur proche.
  - Exemple de proximité temporelle : dans une boucle simple, chaque itération accède aux mêmes instructions.
- Localité spatiale : si une zone est référencée, les zones voisines ont des chances d'être référencées dans un futur proche.
  - Exemple de proximité spatiale : dans un bloc simple, chaque instruction sera accédée l'une après l'autre.

Proximité temporelle

```
DO I = 1,10 0000
        S1 = A(I)
        S2 = A(I+K) # S2 sera réutilisé à l'itération I+K
END DO
```

- L'idée de base est de conserver A(I+K) (lecture de la référence à l'itération I, déclaration S2) en mémoire rapide jusqu'à sa prochaine utilisation à l'itération I+K, déclaration S1.
- Si on veut exploiter la localité temporelle via les registres, cela suppose qu'il faille au moins K registres.
- Cependant, dans le cas général, on aura certainement besoin de stocker d'autres données.

L'évaluation précise de la proximité temporelle est très complexe

Proximité spatiale

- Le voisinage d'une zone localisée par une adresse doit être évaluée dans l'espace d'adressage virtuel.
- Les structures de données sont linéarisées et réduites à une dimension pour être placées dans l'espace d'adressage linéaire.

Exemple en Fortran : rangement des tableaux 2D par colonne

```
DO J = 1,10000
      DO I = 1,1000
            X1 = A(I,J)
            X2 = B(J,I)
      END DO
END DO
```

- Pour A : localité spatiale parfaite, accès à des zones de mémoire consécutives.
- Pour B : 2 références consécutives sont distantes de 1000 emplacements mémoires.

Pour exploiter la localité spatiale, il faut que la distance entre 2 références soit inférieure à la taille de la ligne du cache.

Organisation des caches

- Le cache est divisé en lignes de n mots.
- 2 niveaux de granularités :
  - le CPU travaille sur des « mots » (par exemple, 32 ou 64 bits).
  - Les transferts mémoire se font par blocs (par exemple, lignes de cache de 256 octets).
- Une même donnée peut être présente à différents niveaux de la mémoire : problème de cohérence et de propagation des modifications.
- Les lignes de caches sont organisées en ensembles à l'intérieur du cache, la taille de ces ensembles est constante et est appelée le degré d'associativité.

Mémoire virtuelle

- La mémoire virtuelle est le dernier niveau de la hiérarchie mémoire.
- Espace d'adressage virtuel (ou logique) :
  - Quantité maximale d'espace d'adressage disponible pour une application.
  - Cet espace d'adressage virtuel est en correspondance avec l'espace d'adressage physique pour l'accès aux données via le MMU (Memory Management Unit) au niveau hard, et via le système d'exploitation.
- L'espace d'adressage virtuel de chaque programme et l'espace mémoire sont divisés en pages (physiques ou virtuelles) de même taille.

Page faults

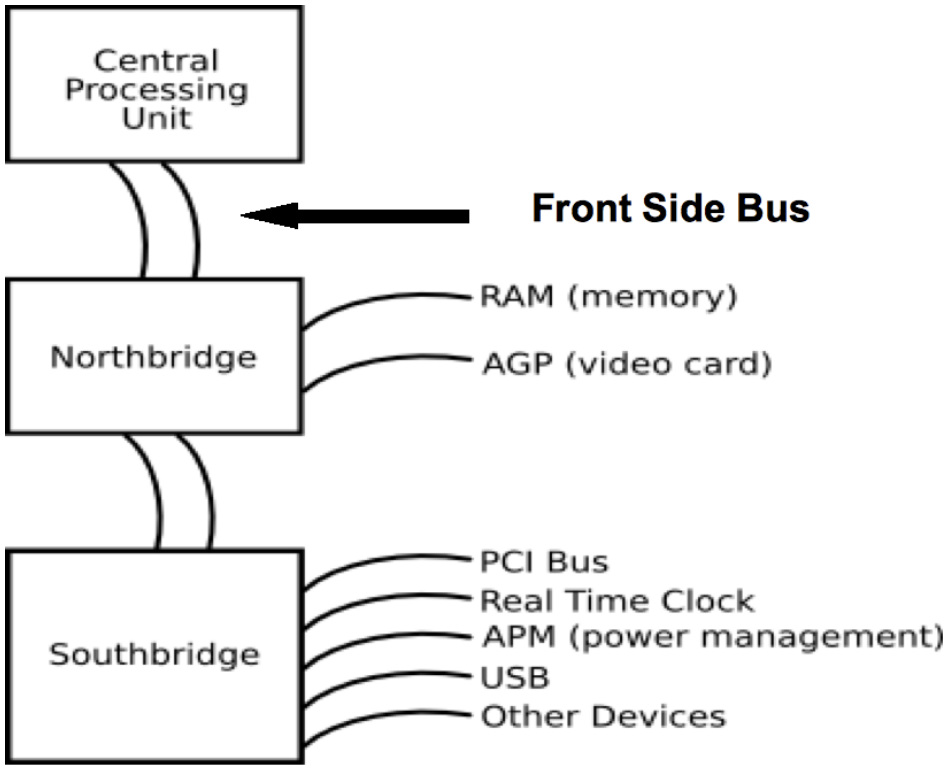
L'adresse virtuelle référence une page qui n'est pas trouvée en mémoire physique :

- Si la mémoire physique est pleine, virer de la mémoire physique une page (remplacement) :
  - choisir une page « victime »,
  - si elle a été modifiée, la réécrire sur disque,
  - modifier les indicateurs de présence dans la table.
- Dans tous les cas :
  - charger la page référencée en mémoire physique (placement),
  - modifier les indicateurs de présence dans la table.

Les défauts de page sont extrêmement coûteux !

Sommaire

Composants



Le débit maximal d'un bus est caractérisé par :

- Le volume d'informations transmises simultanément (exprimé en bits = nombre de lignes physiques sur lesquelles les données sont envoyées de manière simultanée) = largeur du bus.
- La vitesse du bus = fréquence (en Hz)

Evolutions du FSB

- Hypertransport, solution AMD
  - Chaque processeur dispose de 2 canaux de mémoire (contrôleur intégré au processeur).
  - Les processeurs sont reliés entre eux via un lien à 2 GHz en HT 2.0 jusqu'à 3.2 GHz en HT 3.1.
  - L'HyperTransport 3.0 offre une bande passante théorique de 41,6 Go/s.
- QuickPath Interconnect (QPI), solution Intel
  - Remplace le FSB sur l'architecture Nehalem, similaire à l'HT.
  - Le principal intérêt du bus QPI provient de sa topologie point à point : le bus connectant les processeurs au chipset n'est plus partagé.
  - Constitué de 2*20 lignes (20 lignes pour l'émission, 20 lignes pour la réception) par lien QPI. Chaque lien est capable d'effectuer 6.4 milliards de transferts par secondes (GT/s), ce qui correspond à un débit 12.8 Go/s de manière bidirectionnelle, soit un total de 25.6 Go/s pour un lien complet.

Bus PCI

- ✓ Bus série : transmet les bits d'informations un par un.
- ✓ Bus parallèle : transmet les bits d'informations 32 par 32. Les bus parallèles sont limités en cadence par des difficultés techniques et physiques.

- Peripheral Component Interconnect (PCI) : standard de bus local (interne) permettant de connecter des cartes d'extension sur la carte mère d'un ordinateur. Fréquence de 33MHz, largeur de 32 bits, bande passante maximum de 132 MB/s.
- Bus PCI-X : évolution du bus PCI standard. Fréquence d'horloge de 66 à 533 MHz suivant les versions en 32 ou 64 bits (nécessaires pour les cartes réseaux Gigabits et les SCSI Ultra 160 et Ultra 320). Bande passante maximale : 4256 MB/s.
- Bus PCI Express : norme dérivée du PCI, spécifie un bus local série. Contrairement au PCI qui est relié au southbridge de la carte mère, le PCI Express est relié au northbridge.

Bus PCI

- Dans sa version purement PCI la bande passante est partagée entre tous les éléments connectés sur le bus, contrairement à ce qui se passe pour la version PCI Express où elle est dédiée pour chaque périphérique.
- Le PCI utilise un unique bus de largeur 32 bits bidirectionnel alterné (half duplex) pour l'ensemble des périphériques, le PCI Express utilise une interface série (de largeur 1 bit) à base de lignes bidirectionnelles.

PCI Express Lines	Bandwidth per Stream	Bandwidth, duplex
1	256 MB/s	512 MB/s
2	512 MB/s	1 GB/s
4	1 GB/s	2 GB/s
8	2 GB/s	4 GB/s
16	4 GB/s	8 GB/s

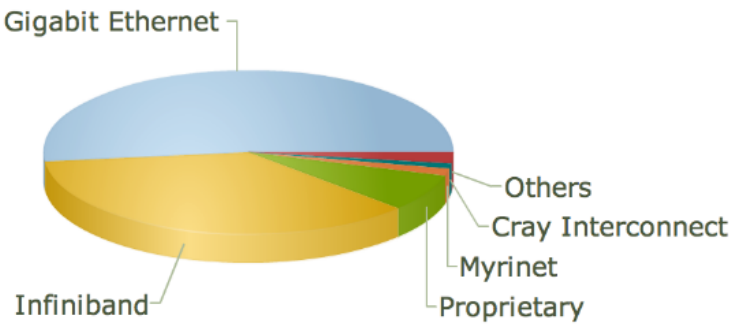
Sommaire

Réseaux de communication

Les caractéristiques importantes :

- La bande passante (bandwidth) est le débit maximal.
- La latence : temps d'initialisation de la communication.
- La distance maximum entre deux noeuds.

Technologie	Latence	Bande passante
1 Gb Ethernet	-	1 Gb/sec
10 Gb Ethernet	~10 μs	10 Gb/sec
Infiniband	< 2 μs	10, 20 & 40 Gb/sec
Myrinet	2.5 - 5.5 μs	10 Gb/sec



Répartition des familles d'interconnexion, top 500, nov 2009

Sommaire

Comment faire des calculateurs plus rapides ?

1. Améliorer la vitesse du processeur : augmenter la fréquence d'horloge (limites techniques, solution coûteuse).
   - La chaleur dégagée par le processeur est fonction de sa fréquence d'horloge.
   - Le nb de cycles d'horloge pour interruptions, « cache miss » ou mauvaise prédiction de branchement augmentent.
2. Complexifier le processeur : permettre l'exécution simultanée de plusieurs instructions :
   - Instruction Level Parallelism : pipelining, superscalabilité, architecture VLIW et EPIC.
   - Thread Level Parallelism : multithreading et SMT.
3. Augmenter le nombre de transistors.
4. Utiliser des processeurs spécialisés.

Tendances actuelles et évolution

- Loi de Moore : nb de transistors par circuit intégré $\times$ 2 tous les 2 ans.
- En 10 ans, la finesse de gravure est passée de $0,25\ \mu m$ (Pentium III) à $0,045\ \mu m$ (Nehalem), et même $0,032\ \mu m$ (Westmere d'Intel).
- Evolution de l'architecture des processeurs (pipeline, duplication des unités fonctionnelles, exécution spéculative, exécution désordonnée ...).
- Evolution des interactions architecture/applications : dans le cas de l'Itanium, le compilateur fait partie intégrante de l'architecture.
- Evolution des infrastructures vers un haut degré de parallélisme en intégrant un grand nombre de cœurs.
- Utilisation de cœurs hétérogènes (CPU + GPU) et API compatibles (CUDA, Brook+, OpenCL)

Kilo, Méga, Giga, Téra, Péta, Exa ...

Top 500 Bench Linpack	**Puissance soutenue en Gflops**	**Puissance crête en Gflops**	**Nombre de processeurs ou de cœurs**
Juin 1993	59.7	131	1024
Nov. 2009	1 759 000	2 331 000	224 162

1997 : année du Teraflops (10^{12} Flops)

2008 : année du Petaflops (10^{15} Flops)

2017 : année de l'Exaflops (10^{18} Flops) ? ? ?

FLOPS (Floating Point Operations Per Second), 1 GFlops $= 10^9$ Flops

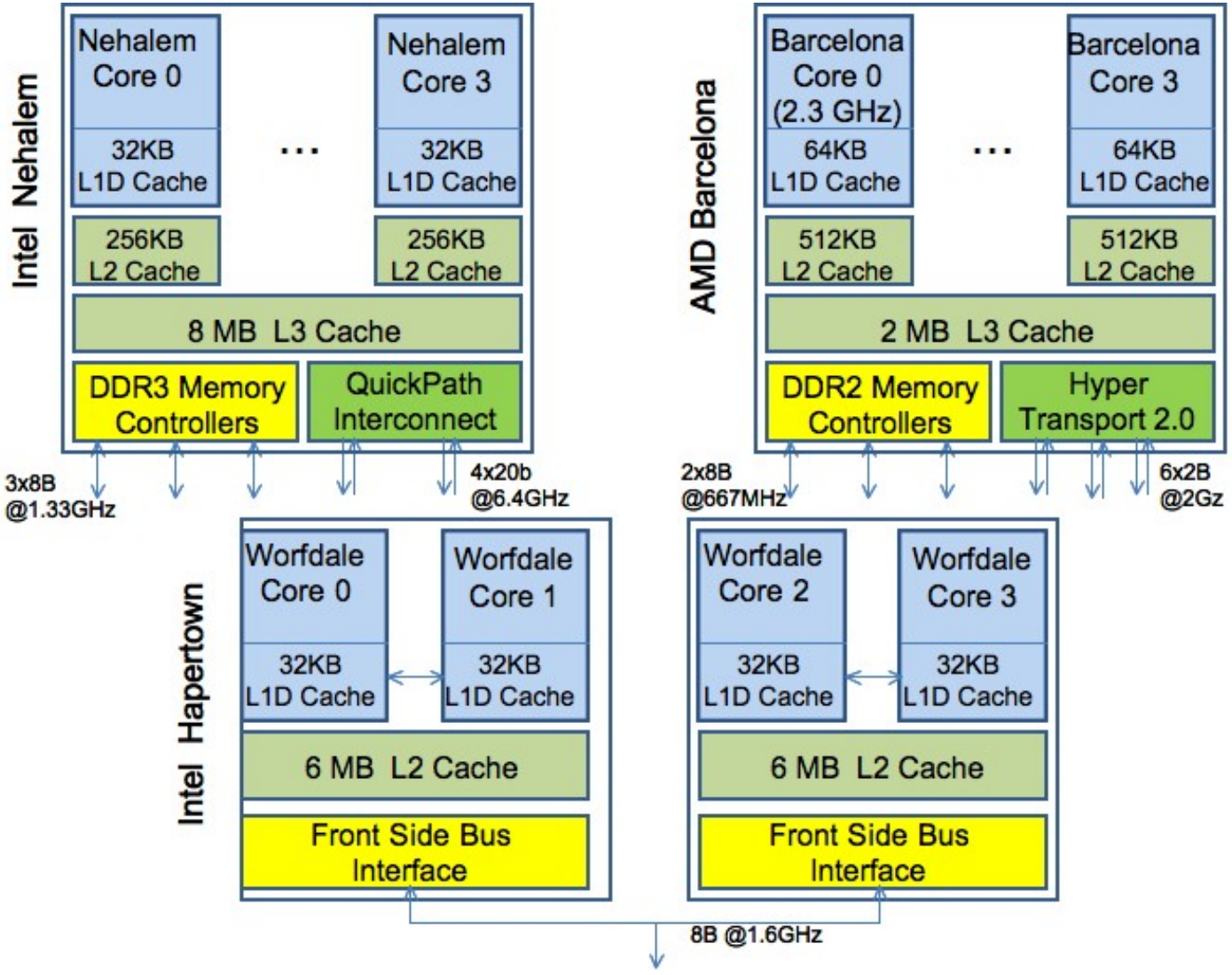
Les architectures multicœurs

- Un processeur composé d'au moins 2 unités centrales de calcul sur une même puce.
- Permet d'augmenter la puissance de calcul sans augmenter la fréquence d'horloge.
- Et donc réduire la dissipation thermique.
- Et augmenter la densité : les coeurs sont sur le même support, la connectique qui relie le processeur à la carte mère ne change pas par rapport à un mono-cœur.

La version originale de la loi de Moore dit que le nombre de transistors le nombre de transistors des circuits intégrés double tous les deux ans. Que faire de tous ces transistors ?

- une complexité sur l'« out of order »accrue,
- des caches de plus en plus grands (mais qui limitent les progrès au niveau de la vitesse),
- plus de CPUs.

Exemples de processeurs multicœurs

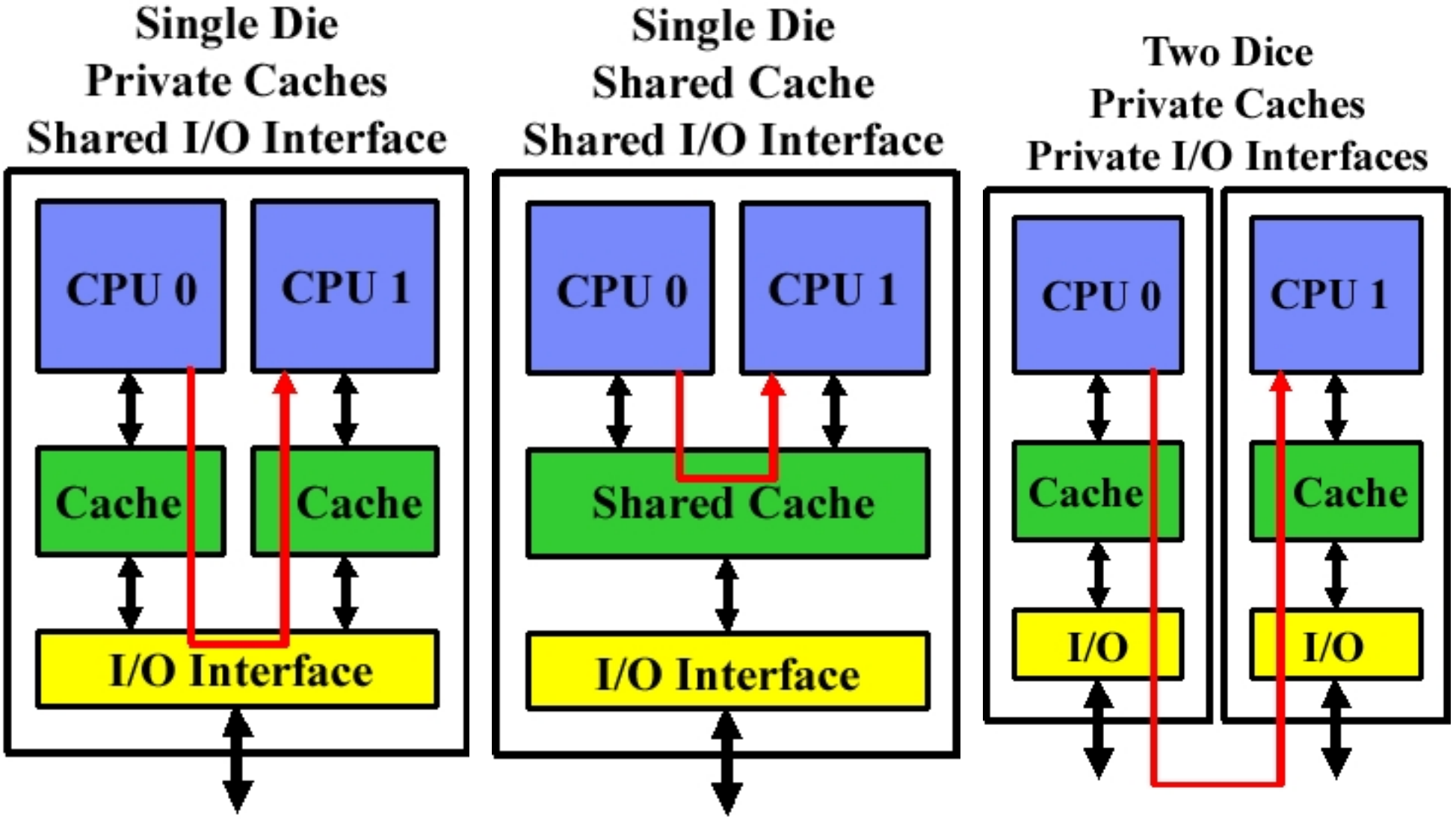


Pourquoi les multicœurs ?

Quelques ordres de grandeur

	Single Core	Dual Core	Quad Core
Core area	A	$\sim$ A/2	$\sim$ A/4
Core power	W	$\sim$ W/2	$\sim$ W/4
Chip power	W + O	W + O’	W + O”
Core performance	P	0.9P	0.8P
Chip performance	P	1.8P	3.2P

Organisation des caches



Partage des caches L2 et L3

- Partage du cache L2 (ou L3) :
  - ☺ communications plus rapides entre cœurs,
  - ☺ meilleure utilisation de l'espace,
  - ☺ migration des threads plus facile entre les cœurs,
  - ☹ contention au niveau de la bande passante et de la mémoire,
  - ☹ problème de cohérence.
- Pas de partage entre les caches :
  - ☺ pas de contention,
  - ☹ communication/migration plus coûteuse, passage systématique par le cache local.
- La tendance :
  - cache L2 non partagé, cache L3 partagé (IBM Power5+ / Power6, Interl Hapertown / Nehalem).

Architectures massivement parallèles

Problématique :

- ✓ Consommation électrique : doubler la fréquence $\Longleftrightarrow \times 8$ de la consommation électrique.
- ✓ Memory wall : la vitesse d'accès à la mémoire augmente beaucoup plus lentement que la puissance crête du processeur.

Solution Blue Gene/P $\Longrightarrow$ augmenter le nombre de coeurs avec des fréquences réduites.

Blue Gene/P

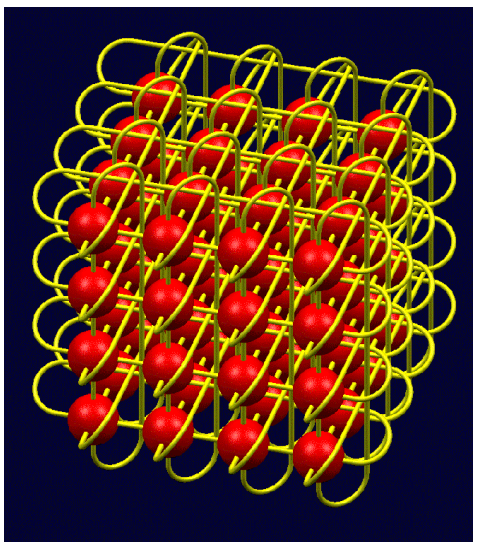
- Avantages :
  - Diminution de l'écart entre les débits et les latences mémoires.
  - Diminution de l'écart entre les débits et les latences réseaux

  $\Longrightarrow$ Machine plus équilibrée.
  $\Longrightarrow$ Puissance de calcul élevée pour une consommation électrique donnée.
- Mais :
  - Chaque coeur est lent.
  - Un très grand nombre de coeurs.
  - La puissance est obtenue que quand on utilise beaucoup de coeurs.
  - Peu de mémoire par coeur.

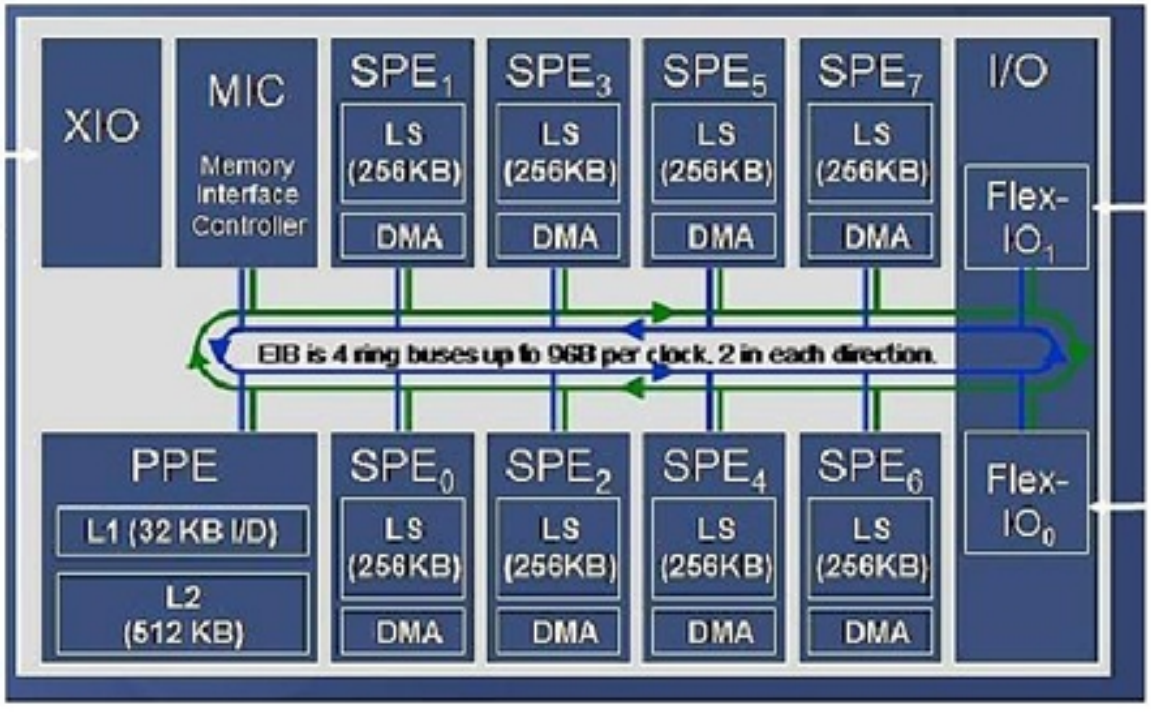
▶ parallélisme massif.

Blue Gene/P, caractéristiques

- ✓ 1 rack Blue Gene/P :
  - 1024 nœuds.
  - 4096 cœurs, Power PC 450, 850 MHz.
  - Cache L1 (32 + 32 ko), L2 privés.
  - 2 Go de mémoire/nœud.
  - 1 nœuds d'IO tous les 64 nœuds.
  - 30 kW/rack, 30 W par nœud (PC standard $\sim$ 150 W).
- ✓ Topologie réseau : 6 réseaux dont 3 pour MPI :
  - tore 3D,
  - réseau collectif,
  - barrières.



Les processeurs spécialisés : le CELL



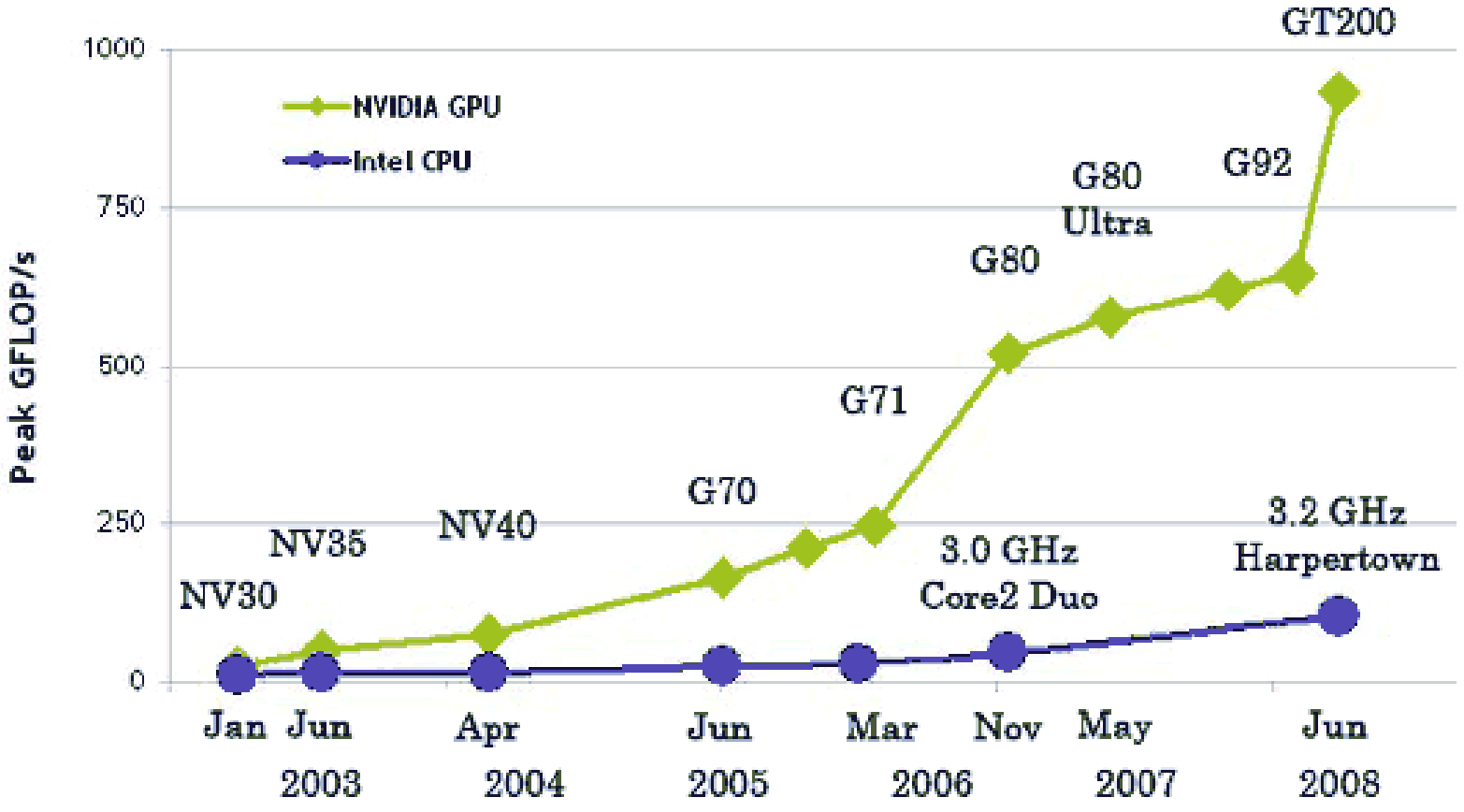
- Développé par Sony, Toshiba et IBM, processeur de la PlayStation 3.
- Un cœur principal (PPE) et 8 cœurs spécifiques (SPE).
- le PPE : processeur PowerPC classique, sans optimisation, « in order », il affecte les tâches aux SPEs.
- les SPEs : constitués d'une mémoire locale (LS) et d'une unité de calcul vectoriel (SPU). Ils ont un accès très rapides à leur LS mais pour accéder à la mémoire principale ils doivent effectuer une requête de transfert asynchrone à un bus d'interconnexion. Les SPEs exécutent les tâches calculatoires.
- Le travail d'optimisation est à la charge du programmeur.

Parallélisme sur le CELL

- les SPU permettent de traiter 4 op 32 bits/cycle (registre de 128 b).
- Programmation explicite des threads indépendante pour chaque cœur.
- Partage explicite de la mémoire : l'utilisateur doit gèrer la copie des données entre cœurs.
  $\Longrightarrow$ Plus difficile à programmer que les GPUs (car pour les GPUs, les threads ne communiquent pas, excepté au début et à la fin).

Processeur CELL : performance crête (registres 128b, SP)
4 (SP SIMD) $\times$ 2 (FMA) $\times$ 8 SPUs $\times$ 3.2 GHz = 204.8 GFlops/socket (en SP)

Evolution des processeurs : GPU vs CPU

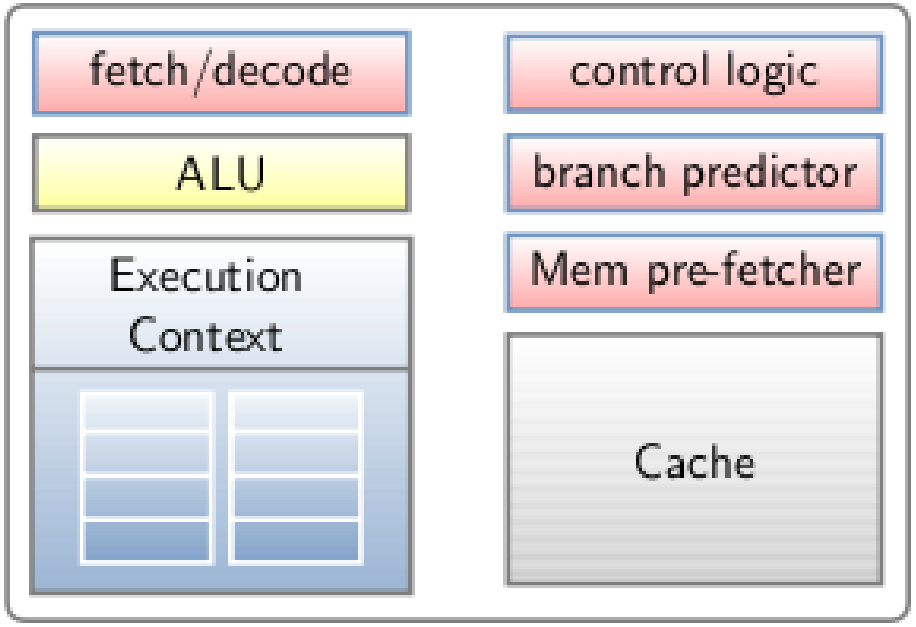


- Vitesse des processeurs classiques * 2 tous les 16 mois
- Vitesse des processeurs graphiques * 2 tous les 8 mois

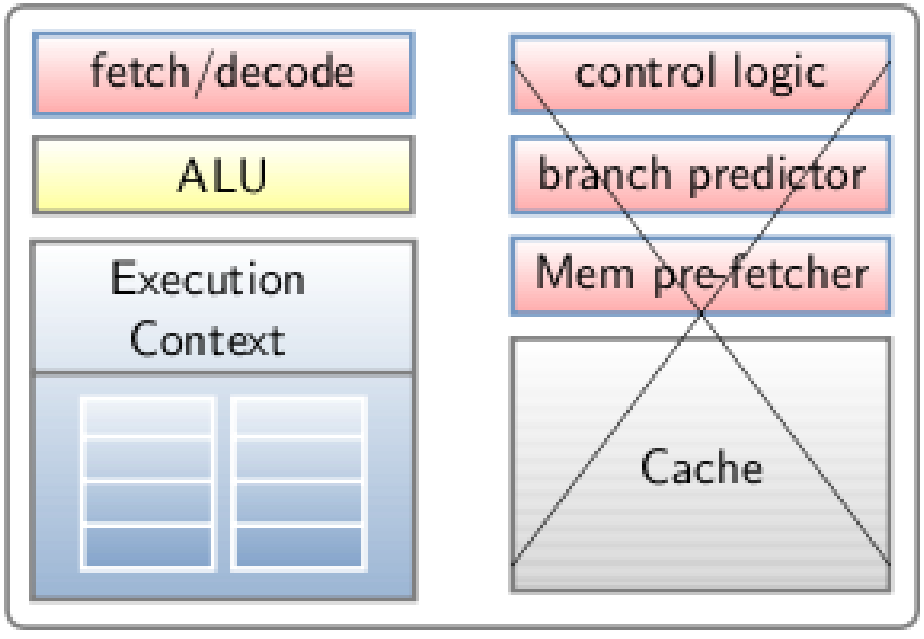
GPU : Graphic Processor Unit

- ✓ Performance théorique GeForce 8800 vs Intel Core 2 Duo 3.0 GHz : 367 GFlops / 32 GFlops.
- ✓ Bande passante mémoire : 86.4 GB/s / 8.4 GB/s.
- ✓ Présents dans tous les PC : un marché de masse.
- ✓ Adapté au massivement parallèle (milliers de threads par application).
- ✓ Jusqu'à récemment, uniquement programmable via des APIs graphiques. Aujourd'hui, des modèles de programmation disponibles : CUDA (Compute Unified Device Architecture).

Du CPU au GPU

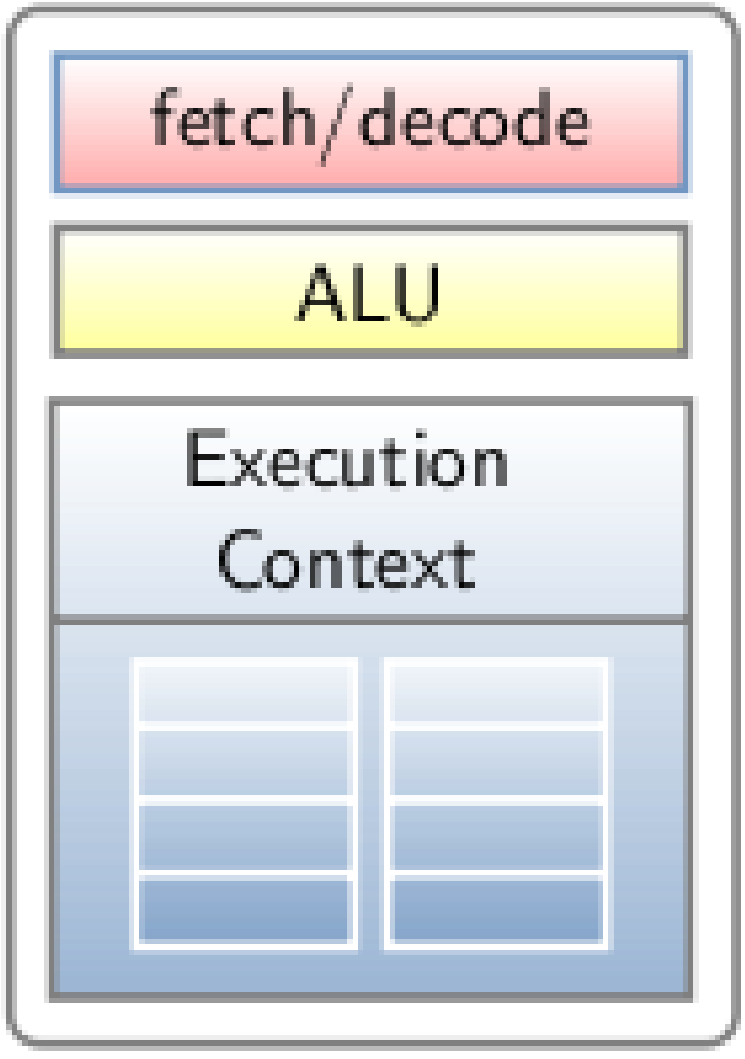

fetch/decode
ALU
Execution Context
control logic
branch predictor
Mem pre-fetcher
Cache

Du CPU au GPU

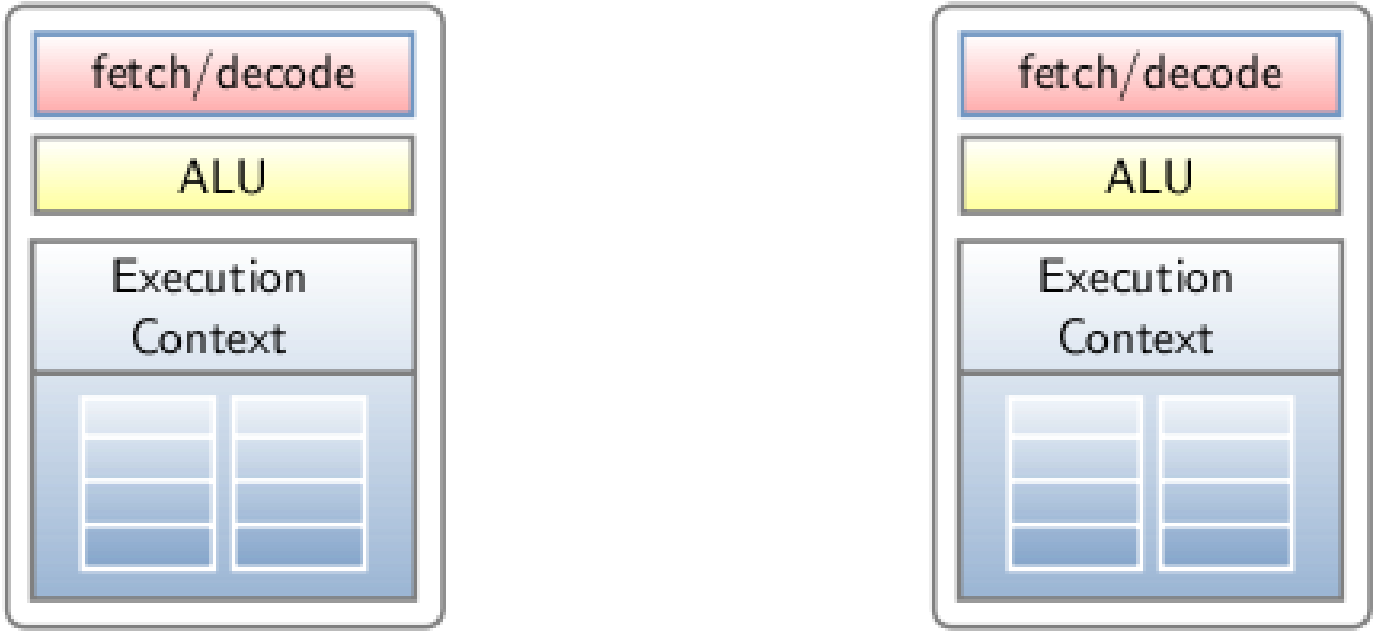


- On enlève les composants qui permettent de gérer le flux d'instructions de façon optimale.

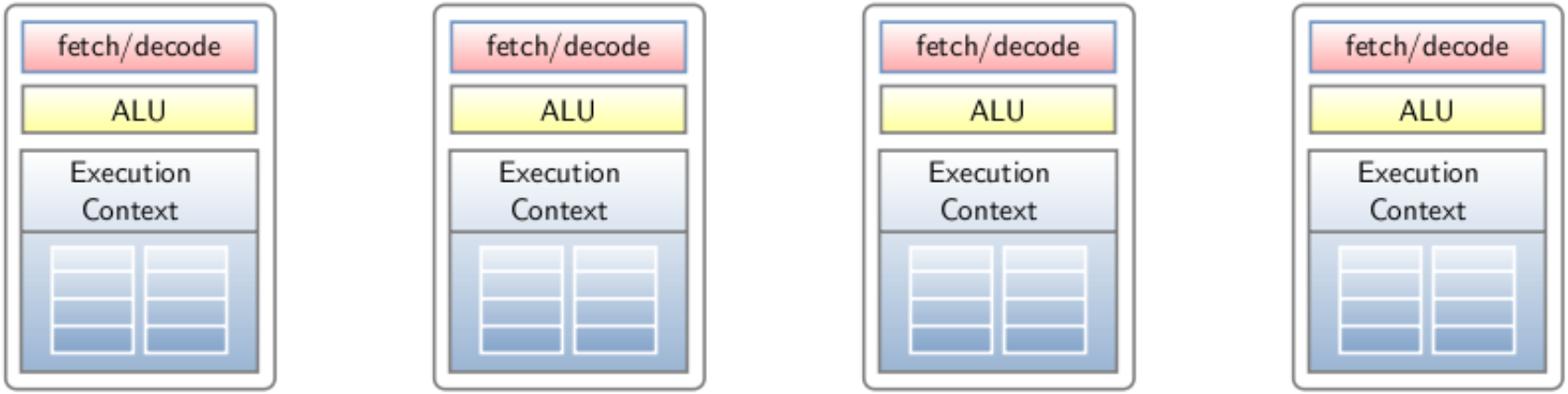
Du CPU au GPU



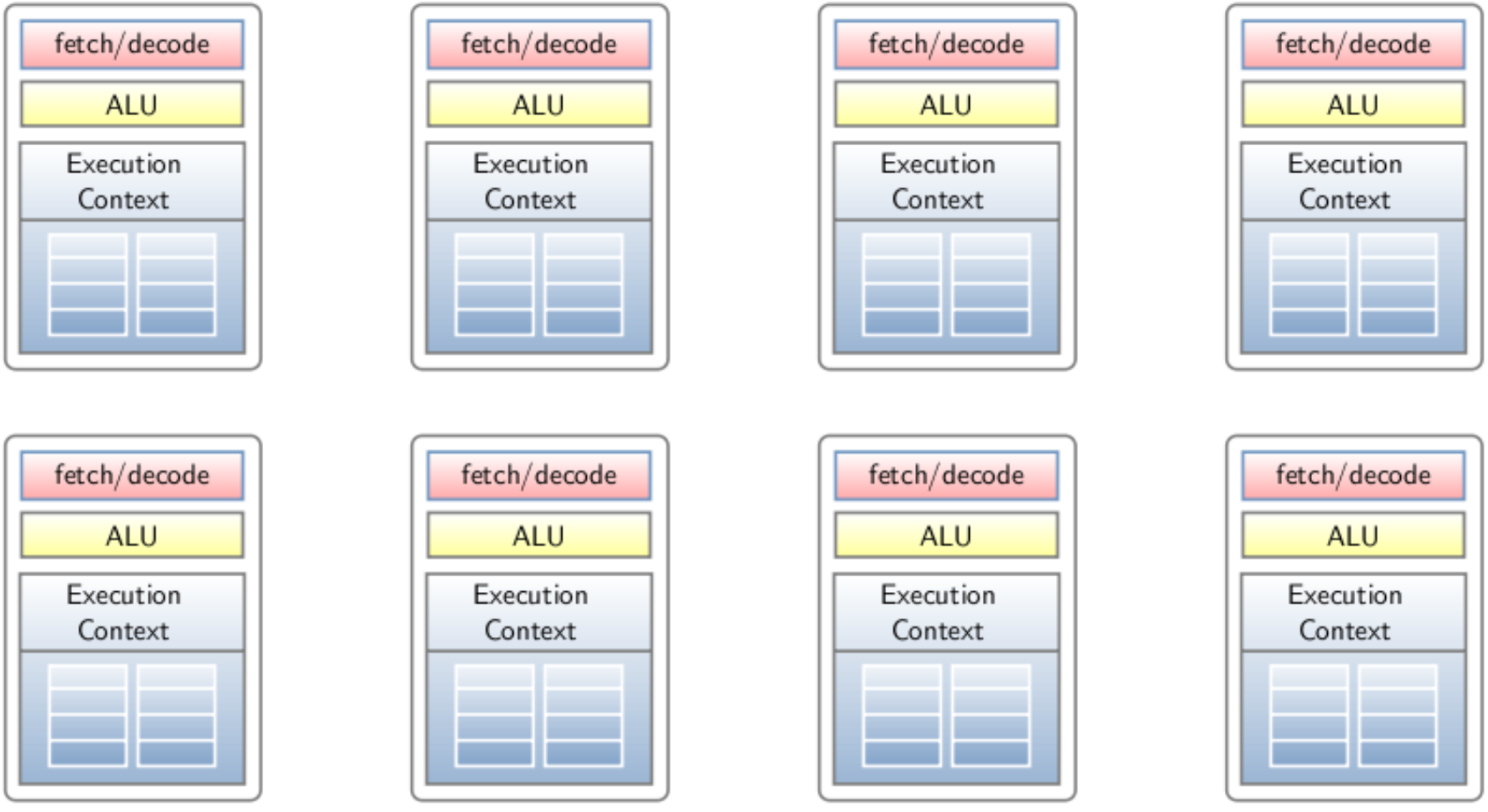
Du CPU au GPU


fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context

Du CPU au GPU


fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context

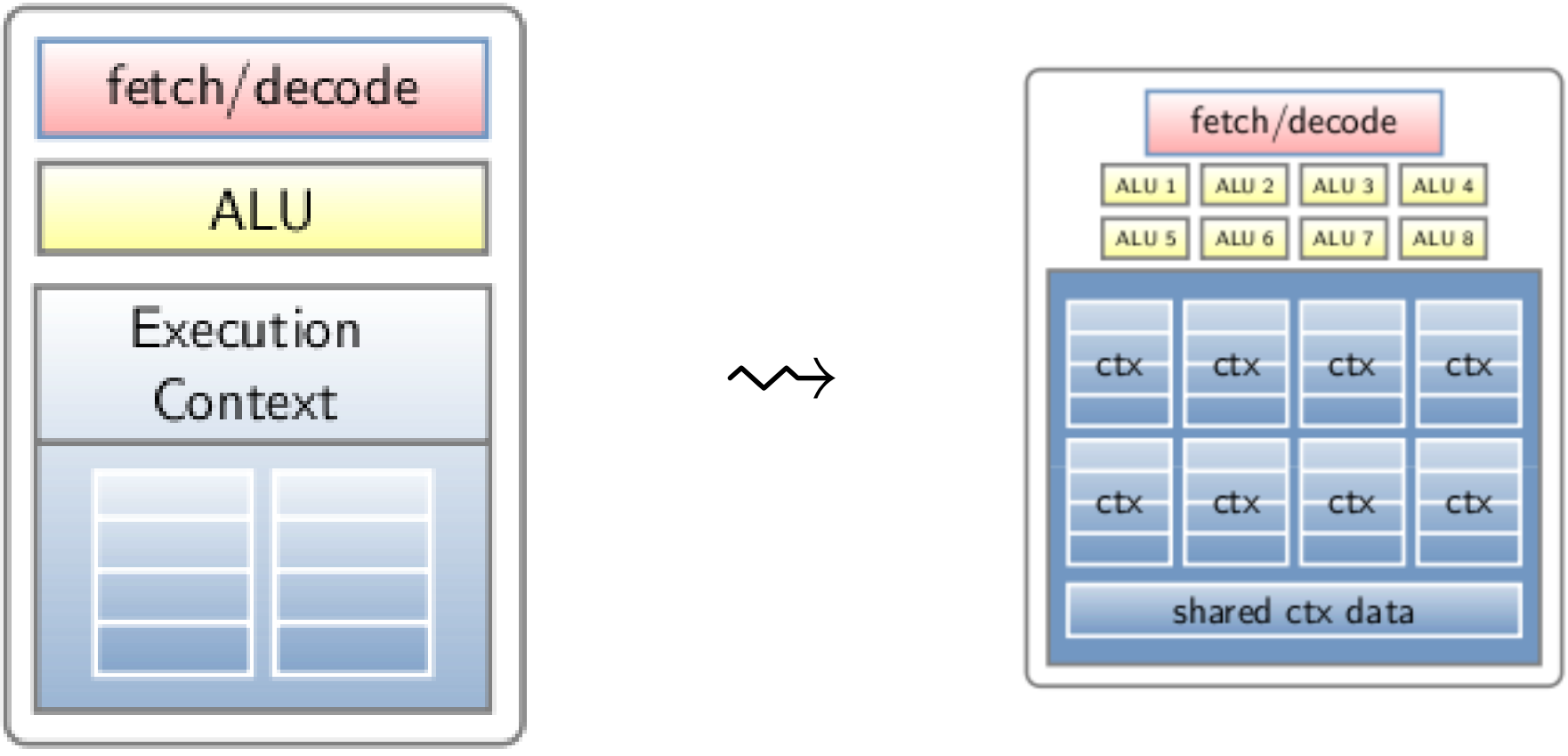
Du CPU au GPU


fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context
fetch/decode
ALU
Execution Context

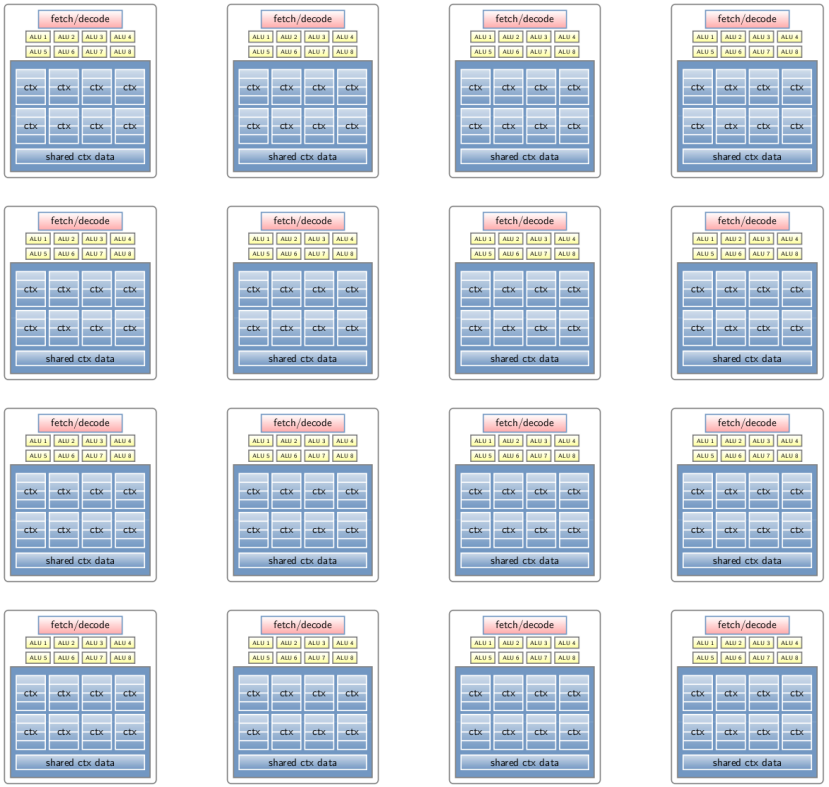
Du CPU au GPU

Du CPU au GPU

Partage du flux d'instuctions :



Du CPU au GPU



- Un GPU est un ensemble de N petites machines SIMD indépendantes et partageant une mémoire globale : N « multiprocesseurs » (ici N = 16)
- Un multiprocesseur est 1 petite machine SIMD avec :
  - ✓ k « ALU » (Stream processor) synchronisés (ici k = 8),
  - ✓ 1 décodeur d'instructions,
  - ✓ 3 mémoires partagées entre toutes les ALUs (dont 2 caches),
  - ✓ 8192 registres distribués entre les ALUs.
- 128 « cœurs »

Sommaire

Conclusions sur les architectures

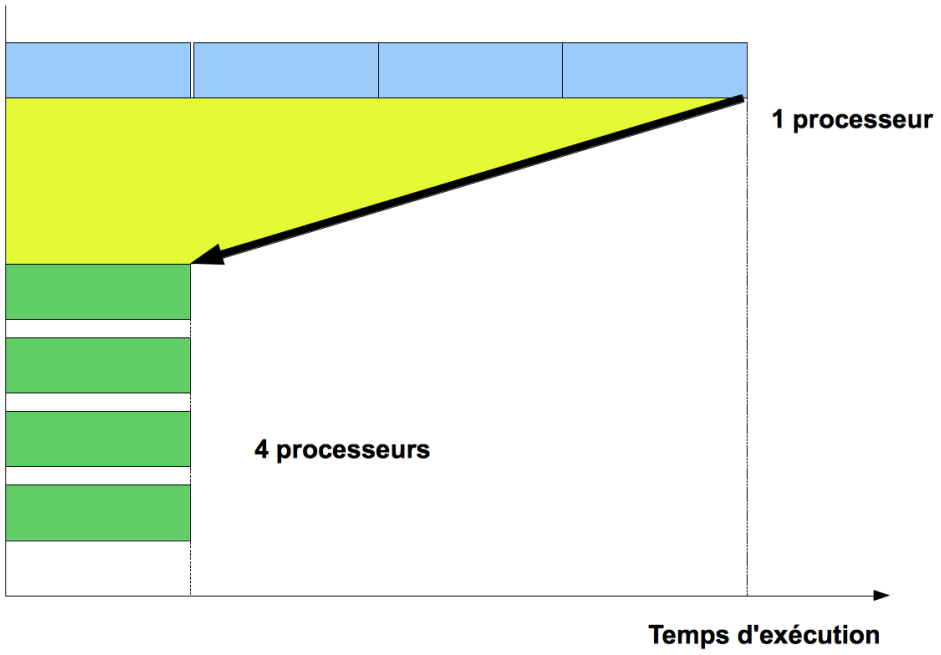
- Vers une augmentation du nombre de cœurs plutôt que de la fréquence d'horloge : problème de finesse de gravure, d'échauffement ...
- Programmation de plus en plus complexe :
  - Nécessité d'une bonne connaissance des architectures, notamment au niveau de la mémoire.
  - Plus de croissance des performances au niveau du cœur $\Longrightarrow$ parallélisation obligatoire pour un gain de performance.
  - Nécessité d'élaborer des algorithmes et des programmes capables d'exploiter un grand nombre de processeurs.
  - Nécessité d'exploiter le parallélisme aux différents niveaux du hardware.
  - Programmation sur des architectures hybrides avec différents types de processeurs.
- Exploitation des performances des processeurs graphiques : produire un code extrêmement parallélisé, capable d'utiliser TOUS les « threads », sinon on n'atteint que quelques % de l'utilisation de la puissance disponible.

Sommaire

Sommaire

Introduction

- Exécutions séquentielles des programmes : le processeur n'exécute les instructions que l'une après l'autre, en passant éventuellement d'un processus à un autre via des interruptions en simulant le multitâche.
- Parallélisation : Ensemble de techniques logicielles et matérielles permettant l'exécution simultanée de séquences d'instructions indépendantes, sur des processeurs différents.



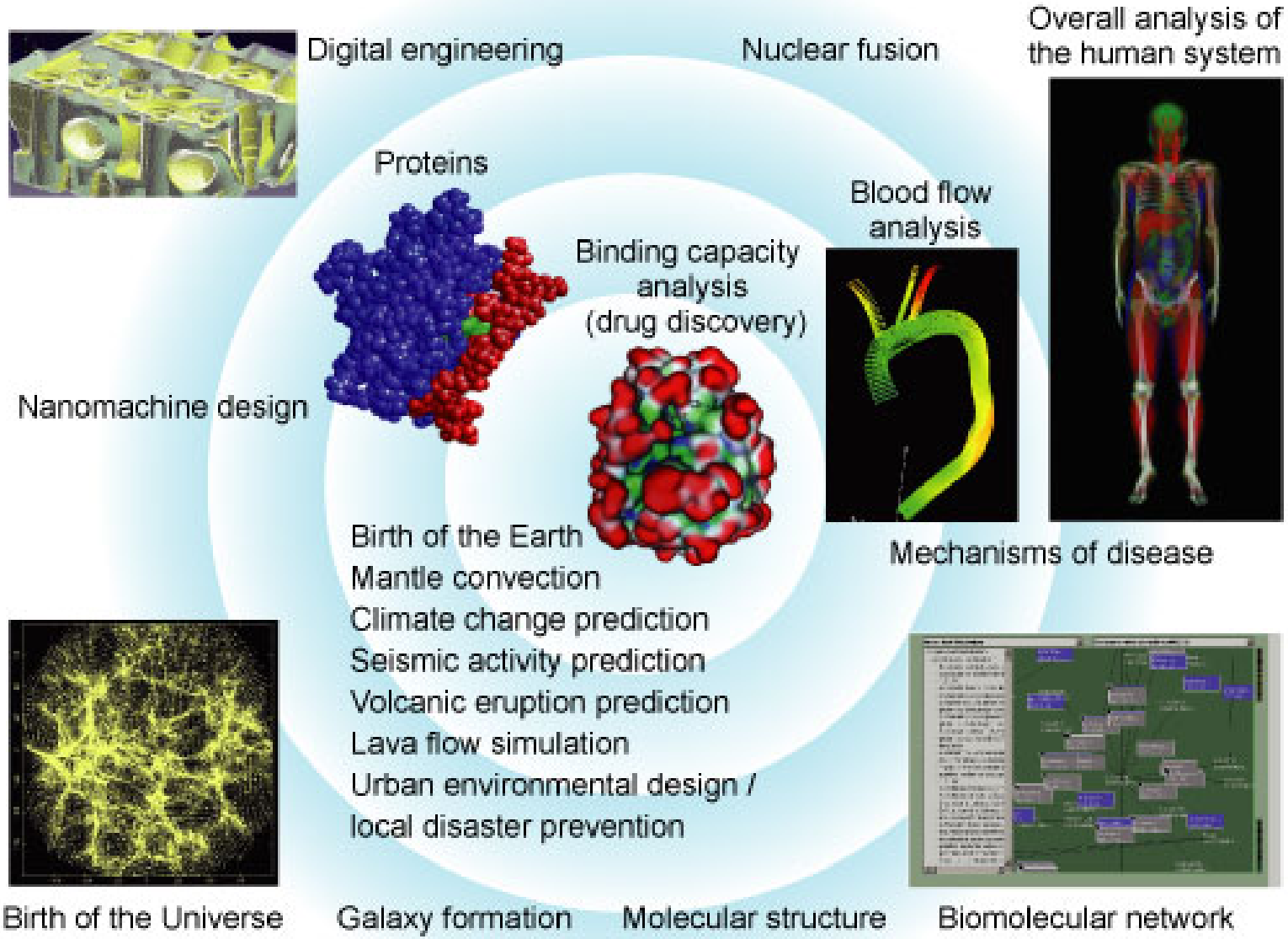
Pourquoi paralléliser ?

- Traitement de problèmes scientifiques constituant de grands challenges, tels que : météo et climat, biologie (génome), géophysique (activité sismique), réactions chimique et réactions nucléaires, ...
- Mais aujourd'hui également pour des applications commerciales : Bases de données parallèles, exploration pétrolière, moteurs de recherche web, réalité virtuelle, ...

✓ Traiter des problèmes plus grands et/ou plus complexes.
✓ Mais aussi exploiter les architectures actuelles !

La multiplication du nombre d'unités de calcul ne divise pas spontanément le temps d'exécution des programmes !

Pourquoi paralléliser ?



Sommaire

Classification de Flynn

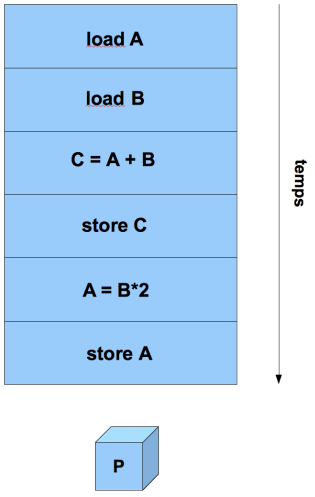
Les programmes et les architectures sont classés selon le type d'organisation du flux de données (« data ») et du flux d'instructions.

- 2 états possibles pour chacune : « single » ou « multiple »

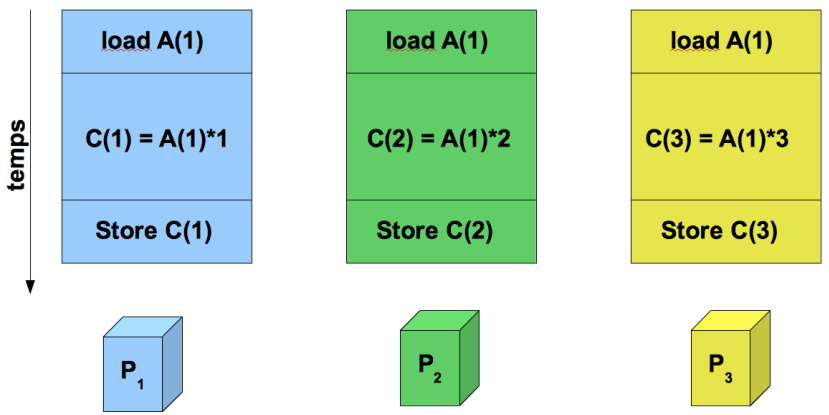
SISD	SIMD
Single Instruction, Single Data	Single Instruction, Multiple Data
$\Longrightarrow$ machines séquentielles	$\Longrightarrow$ processeurs vectoriels, GPU
MISD	**MIMD**
Multiple Instruction, Single Data	Multiple Instruction, Multiple Data
$\Longrightarrow$ rare	$\Longrightarrow$ multiproc, multicores

SISD, MISD

SISD architecture séquentielle avec un seul flot d'instructions, un seul flot de données, exécution déterministe.

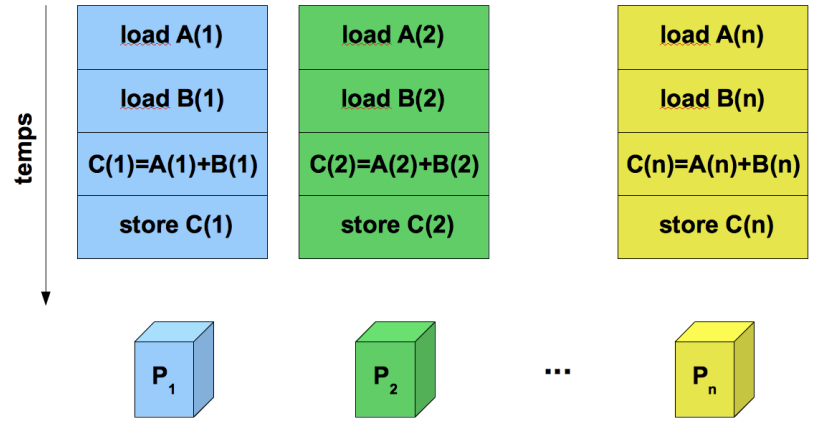


MISD un seul flot de données alimente plusieurs unités de traitement, chaque unité de traitement opère indépendamment des autres, sur des flots d'instructions indépendants.
Peu implémenté.
Ex d'utilisation : exécution en // de plusieurs algos de cryptographie pour le décodage d'1 même message.

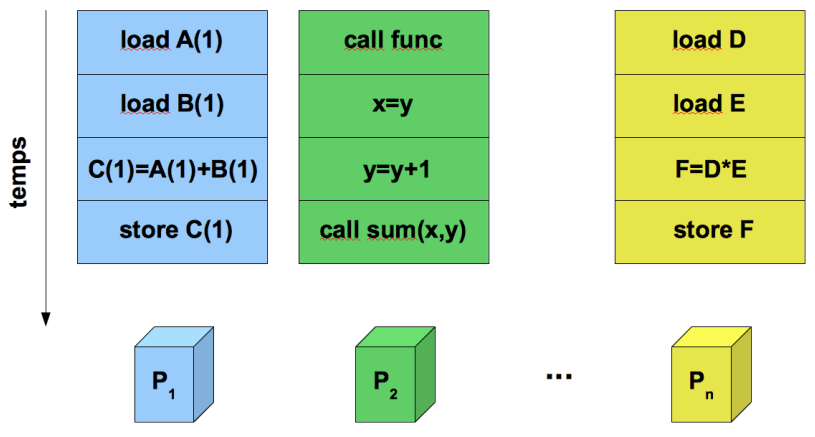


SIMD, MIMD

SIMD architecture // , toutes les unités de traitement exécutent la même instruction à un cycle d'horloge donnée, chaque unité peut opérer sur des données différentes, exécution déterministe.



MIMD architecture la plus courante hui. Chaque unité de traitement peut gérer un flot d'instructions différent. Chaque unité peut opérer sur un flot de données différent. L'exécution peut être synchrone ou asynchrone, déterministe ou non-déterministe.



Terminologie

- Tâche : une portion de travail à exécuter sur un ordinateur, du type un ensemble d'instructions d'un programme qui est exécuté sur un proc.
- Tâche parallèle : une tâche qui peut s'exécuter sur plusieurs processeurs, sans risque sur la validité des résultats.
- Exécution séquentielle : exécution d'un programme séquentiel, une étape à la fois.
- Exécution parallèle : exécution d'un programme par plusieurs tâches, chaque tâche pouvant exécuter la même instruction ou une instruction différente, à un instant donné.
- Mémoire partagée : D'un point de vue hard, réfère à une machine dont tous les proc. ont un accès direct à une mémoire commune (généralement via un bus).
  D'un point de vue modèle de programmation : toutes les tâches ont la même image mémoire et peuvent directement adresser et accéder au même emplacement mémoire logique, peu importe où il se trouve en mémoire physique.

Terminologie

- Mémoire distribuée : d'un point de vue physique, basée sur un accès mémoire réseau pour une mémoire physique non commune.
  D'un point de vue modèle de programmation, les tâches ne peuvent voir que la mémoire de la machine locale et doivent effectuer des communications pour accéder à la mémoire dune machine distante, sur laquelle d'autres tâches s'exécutent.
- Communications : les tâches parallèles échangent des données, par différents moyens physiques : via un bus à mémoire partagée, via un réseau. Quelque soit la méthode employée, on parle de « communication ».
- Synchronisation : la coordination des tâches en temps réel est souvent associée aux communications, elle est souvent implémentée en introduisant un point de synchronisation au-delà duquel la tâche ne peut continuer tant que une ou plusieurs autres tâches ne l'ont pas atteint.

Terminologie

- Granularité : mesure qualitative du rapport calcul / communications
  - Grain grossier (coarse) : relativement bcp de calculs entre différentes communications.
  - Grain fin (fine) : relativement peu de calcul entre différentes communications.
- speedup : mesure de l'amélioration des performances due au parallélisme.
  (wall clock time of serial execution) / (wall clock time of parallel execution)
- Scalabilité : réfère à la capacité d'un système parallèle à fournir une augmentation de l'accélération proportionelle à l'augmentation du nombre de processeurs.

Sommaire

Parallélisation efficace

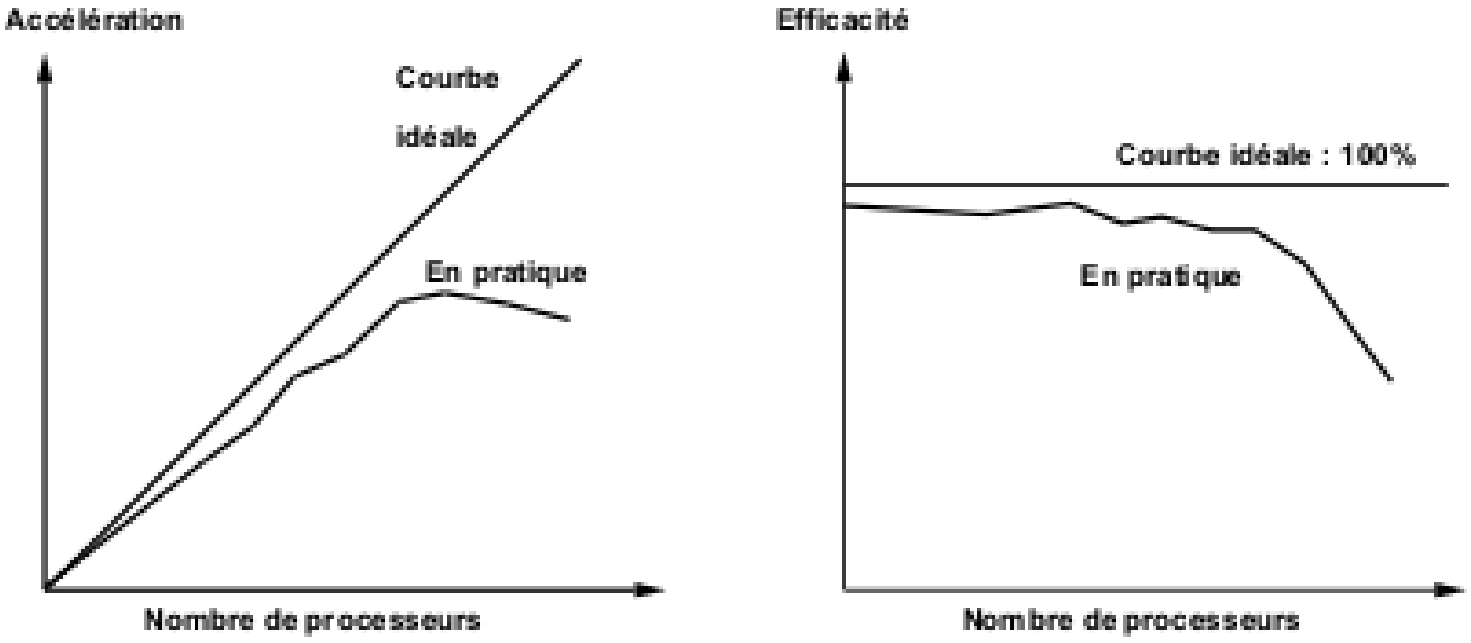
- Accélération et efficacité sont une mesure de la qualité de la parallélisation.
- Soit $T(p)$ le temps d'exécution sur p processeurs.
- L'accélération $A(p)$ et l'efficacité $E(p)$ sont définies comme étant :

$$\begin{aligned} A(p) &= T(1)/T(p) \ \ (p = 1, 2, 3...) \\ E(p) &= A(p)/p \end{aligned}$$

- Pour une accélération parallèle parfaite, on obtient :

$$\begin{aligned} T(p) &= T(1)/p \\ A(p) &= T(1)/T(p) = T(1)/(T(1)/p) = p \\ E(p) &= A(p)/p = p/p = 100\% \end{aligned}$$

Accélération et efficacité



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

Scalabilité, loi d'Amdahl

$\Longrightarrow$ Propriété d'une application à être exécutée efficacement sur un très grand nombre de processeurs.

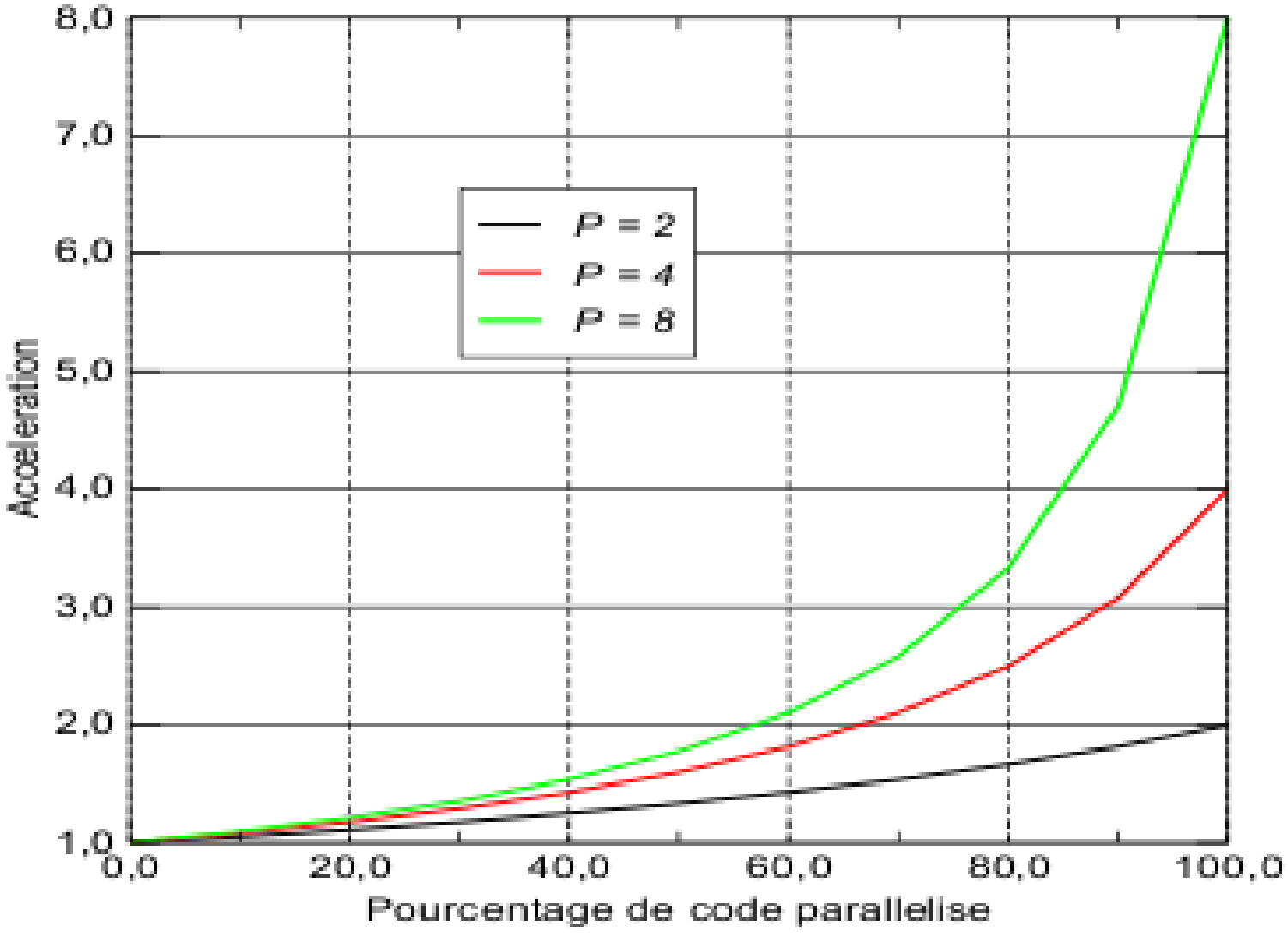
$\Longrightarrow$ Raison possibles d'une faible scalabilité :

- La machine parallèle employée : architecture inadaptée, charge ...
- Le programme parallélisé : analyser le comportement, améliorer les performances.
- un peu des deux ...

Loi d'Amdahl

- Plus une règle qu'un loi.
- Principe : **la partie non parallèle d'un programme limite les performances et fixe une borne supérieure à la scalabilité.**

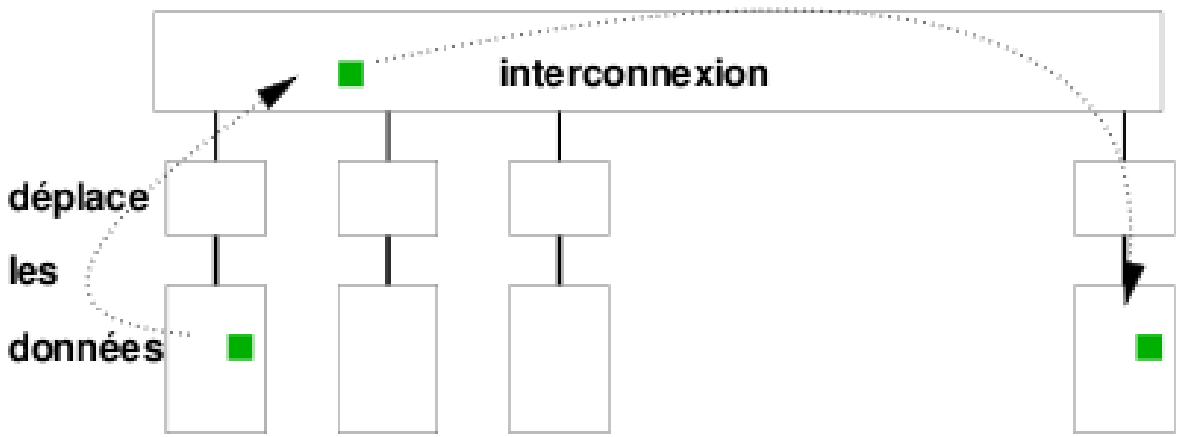
Loi d'Amdahl


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

Sommaire

Mémoire Partagée

- Tous les processeurs accède à toute la mémoire globale avec un même espace d'adressage global.
- Chaque processeur travaille indépendamment des autres, mais les modifications effectuées par un processeur à un emplacement mémoire (en mémoire globale) sont visibles par tous les autres.
- Les processeurs ont leur propre mémoire locale (cache).



Mémoire Partagée

- UMA, Uniform Memory Access :
  - Des processeurs identiques, ayant tous le même temps d'accès à la mémoire. Plus couramment appelées SMP (Symmetric MultiProcessor).
  - Gestion de la cohérence des caches. Certains systèmes sont CC-NUMA : si un processeur modifie un emplacement mémoire partagé, la modification est visible par tous les autres processeurs.
- NUMA (Non Uniform Memory Access) :
  - Conçue pour pallier aux problèmes d'accès mémoire concurrents via un unique bus.
  - En général fabriquées à base de plusieurs blocs SMP interconnectés.
  - Des temps d'accès à la mémoire diffèrent suivant la zone accédée. Le temps d'accès via le lien d'interconnexion des blocs est plus lent.
  - Gestion de la cohérence des caches. CC-NUMA : Cache Coherent NUMA.

Mémoire Partagée

- Espace d'adresse globale $\longrightarrow$ facilite le travail du programmeur.
- Mémoire proche des CPUs $\longrightarrow$ le partage des données entre tâches est rapide.

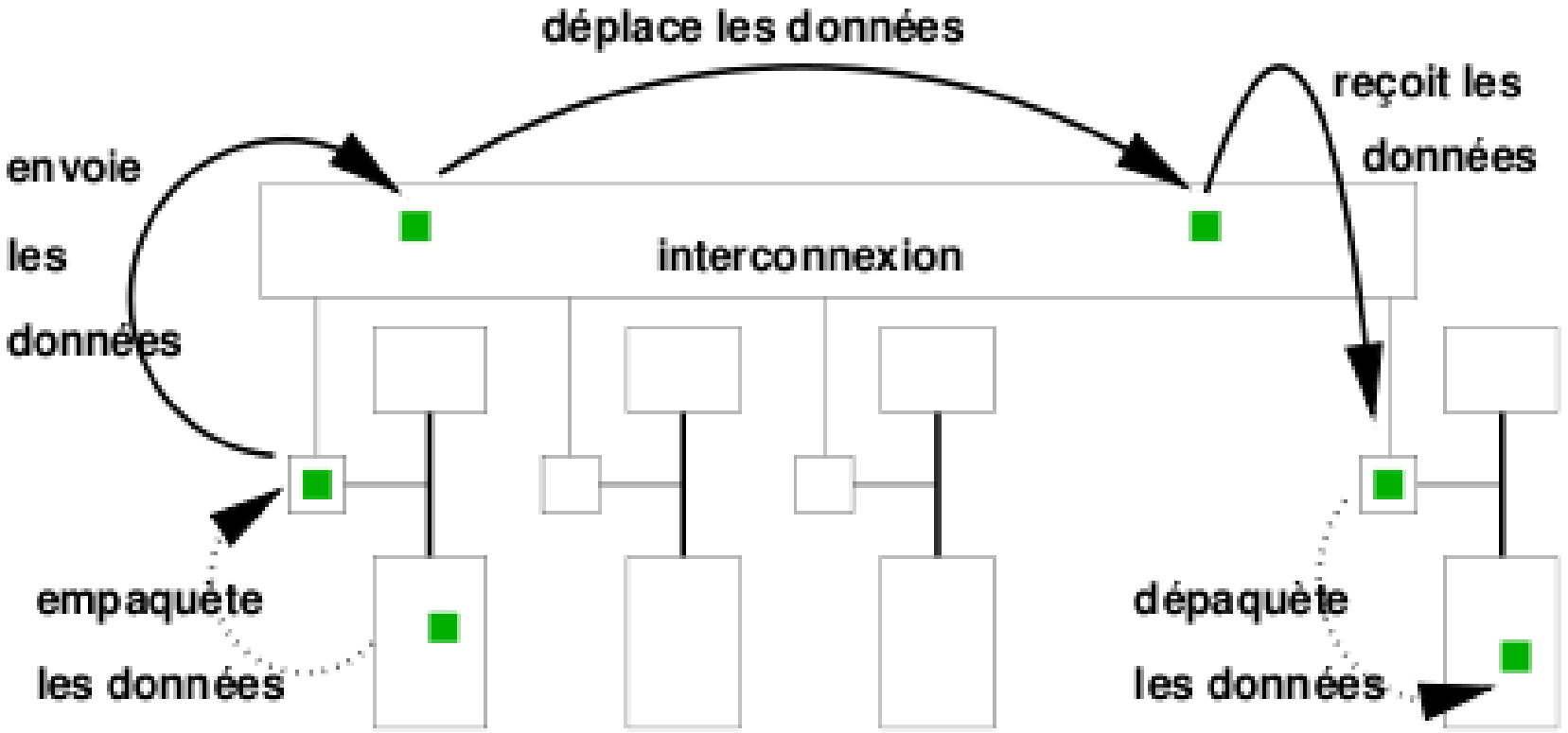
Mais :

- Manque de scalabilité : augmenter le nombre de CPUs accroît le trafic sur le chemin d'accès à la mémoire partagée.
- Le programmeur doit gérer la synchronisation pour un accès correct à la mémoire globale.
- Très coûteux de construire des architectures à mémoire partagée avec un grand nombre de CPUs.

Mémoire Distribuée

- Le point commun de ces architectures : elles possèdent un réseau d'interconnexion pour communiquer entre les mémoires des différents processeurs.
- Chaque processeur à sa propre mémoire locale, il n'y a pas de notion d'espace d'adressage global entre tous les procs.
- Les procs opèrent indépendamment les uns des autres, une modification en mémoire locale n'a pas d'effet sur la mémoire des autres procs.
- Si un processeur a besoin d'une donnée dans la mémoire d'un autre processeur, le programmeur doit définir explicitement la communication.
- Les réseaux d'interconnexion sont divers, avec des niveaux de performance très variables.

Mémoire Distribuée



Mémoire Distribuée

- On peut augmenter le nombre de processeur et la mémoire proportionnellement.
- Accès rapide à la mémoire locale sur chaque proc, sans surcoût de gestion de cohérence de cache.
- Coût raisonnable, ex : PCs en réseau.

Mais :

- Le programmeur doit gérer toutes les communications entre processeurs.
- Peut être difficile de faire coïncider une structure de données basée sur une mémoire globale, à cette organisation physique de la mémoire.
- Temps d'accès mémoire non locaux élevés.

Architecture mémoire hybride

- Les ordinateurs les plus puissants au monde sont aujourd'hui un mixte de mémoire partagée et mémoire distribuée.
- La brique de base (nœud) est un multiprocesseur à mémoire partagée.
- Ces briques sont interconnectées par un réseau (type ethernet, myrinet, Infiniband, ...).

Mémoire partagée vs mémoire distribuée

- Parallélisation plus simple sur architecture à mémoire partagée.
- Mais mémoire partagée + chère que mém. distribuée.

- Des systèmes à mémoire virtuellement partagée pour bénéficier des avantages du partagé sur architecture distribuée.
- Des langages à espace d'adressage global, PGAS (Unified Parallel C, Co-array Fortran, ...).
- Grilles de calcul : constituée de nombreuses ressources informatiques hétérogènes (ordinateurs séquentiels, clusters, supercalculateurs...), géographiquement éloignées, mises en réseau, nécessitant un middleware, ensemble des couches réseaux et services logiciels qui permettent le dialogue entre les différents composants d'une application répartie.

Sommaire

Références

- http://www.realworldtech.com
- https://computing.llnl.gov/tutorials/parallel_comp/
- http://calcul.math.cnrs.fr/
- http://www.idris.fr
- www.top500.org
- http://www.prace-project.eu/hpc-training
- http://www.inria.fr/actualites/colloques/cea-edf-inria/index.fr.html