Calcul Haute Performance architectures et modèles de programmation

Françoise Roch [1]

[1]Observatoire des Sciences de l'Univers de Grenoble
Mesocentre CIMENT

Un peu de connaissance des architectures est nécessaire

Des architectures

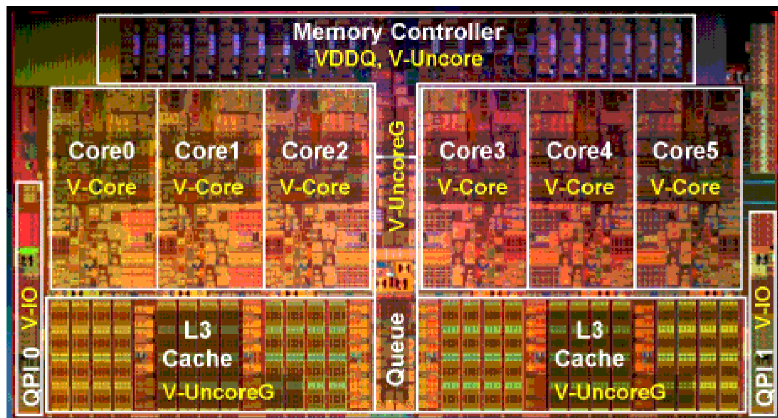
de plus en plus complexes

avec différents niveaux de parallélisme

Première étape de la paralllélisation :

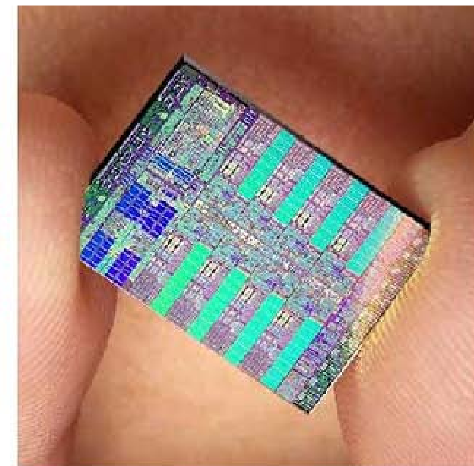
décoder la relation entre l'architecture et l'application

- Adapter les algorithmes, la programmation aux architectures actuelles,
- Comprendre le comportement d'un programme,
- Choisir son architecture et son modèle de programmation en fonction de ses besoins



Westmere-EX

GeForce 8800



Cell

Les ingrédients d’un supercalculateur

- Processeur

  fournit la puissance de calcul

  CPU multi-core, éventuellement GPU, ou proc plus spécialisés
- Mémoire/Stockage

  Aujourd’hui très hierarchisé (caches, DRAM, SSD, HD, bandes)
- Réseaux

  Relient les noeuds entre eux

  Souvent plusieurs types (MPI, Administration, I/O)
- Logiciels

  middleware pour l’accès au ressources distribuées

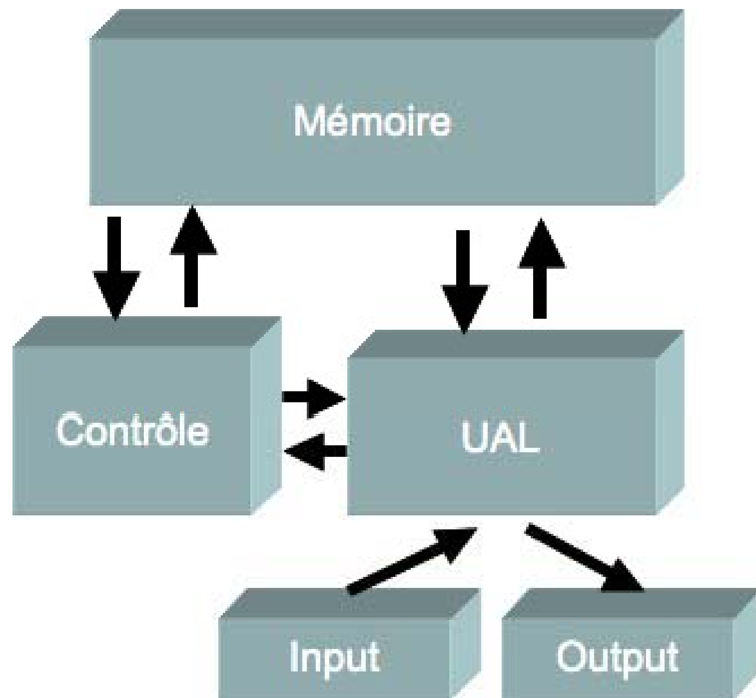
  MPI, gestion des processus, …

L’équilibre entre les différents composants est primordial

Sommaire

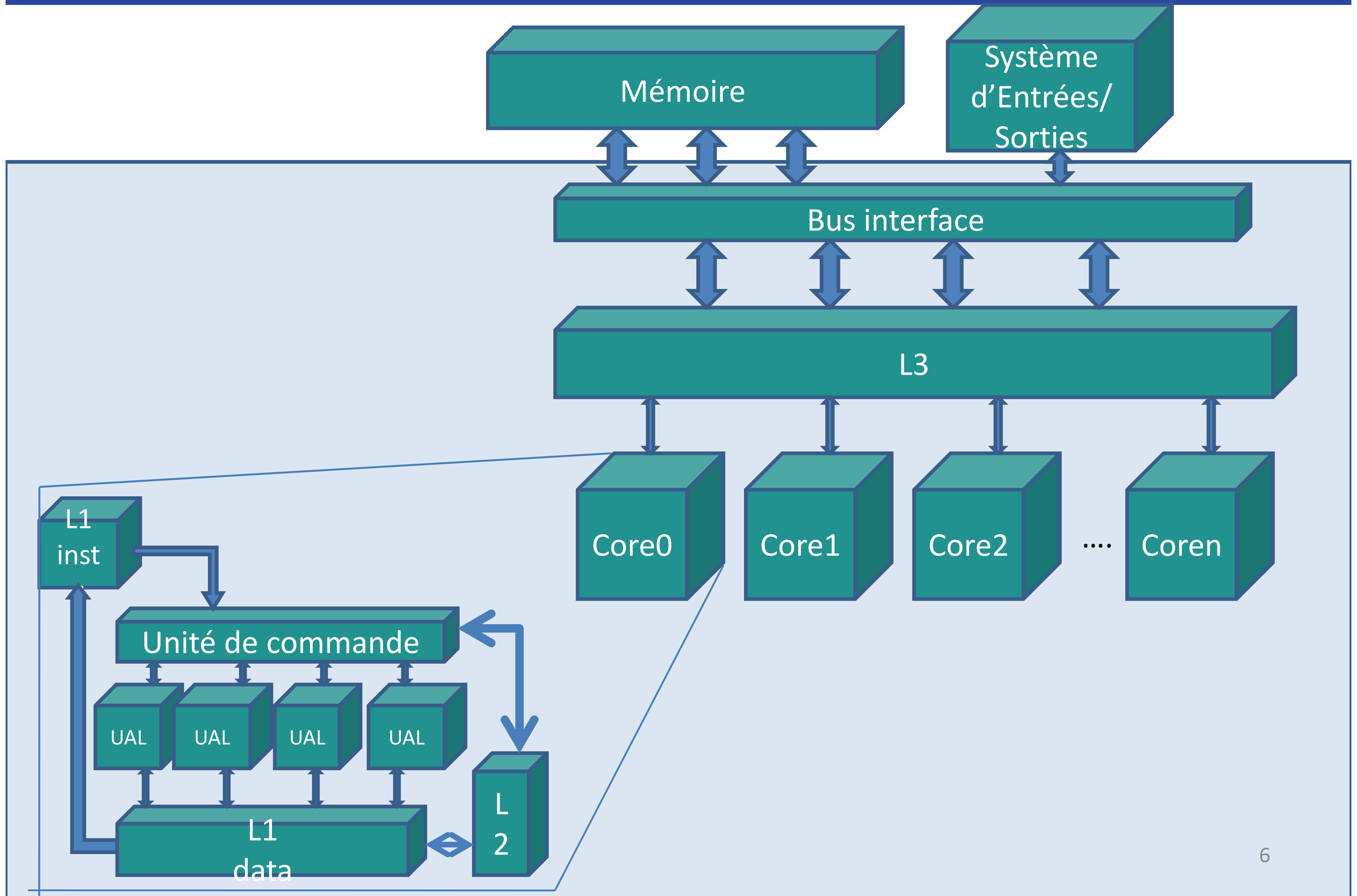
1. Architecture des ordinateurs
   - Les processeurs
   - La mémoire
   - Les réseaux
   - Les tendances d'évolution
2. Concepts du parallélisme
   - Introduction
   - Terminologie et classification
   - Les différents modèles de programmation
3. Conclusion

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.

Un pocesseur en 2011



Caractéristiques d’un processeur

- Jeux d’instructions qu’il peut exécuter :

  Complexe : CISC (beaucoup d’instr. Complexes prenant plusieurs cycles d’horloge). Ex: x86

  Réduit : RISC (Moins d’instr. Mais n’utilisant que quelques cycles). Ex: PowerPC

- Nombre de transistors: plus il contient de transistors, plus il est capable d’exécuter d’instructions /seconde

Caractéristiques d’un processeur

Nombre de bits pouvant être traités en une instruction

Vitesse maximale de l’horloge augmente, plus le processeur exécute d’instructions par 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.

Les limites technologiques :

- La consommation électrique et la dissipation thermique d’un processeur sont fonction de sa fréquence d’horloge.
- Le temps d’interconnexion ne suit pas la finesse de gravure.
- Le nb de cycles d’horloge pour interruptions, « cache miss » ou mauvaise prédiction de branchement augmente.

Comment augmenterla performance des processeurs

- Augmenter la fréquence d’horloge (limites techniques, solution coûteuse).
- Permettre l’exécution simultanée de plusieurs instructions :
  - ✓ Instruction Level Parallelism : pipelining, instruction superscalaire, architecture VLIW et EPIC.
  - ✓ Thread Level Parallelism : multithreading et SMT.
  - ✓ Jeu d’instructions
- Améliorer les accès mémoire : différents niveaux de cache
- Recourir à un nombre élevé de cœurs de calcul

Les composants d'un processeur

1 CPU = plusieurs unités fonctionnelles qui peuvent travailler en parallèle :

- une unité de gestion des bus (unité d'entrées-sorties) en interface avec la mémoire vive du système,
- une unité de commande (unité de contrôle)) 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 (ALU) qui assurent les fonctions basiques de calcul arithmétique et les opérations logiques.

  une ou plusieurs unités de virgule flottante (FPU) : calculs sur les flottants. Instruction de calcul de base : multiplication / addition.

La performance crête peut dépendre de la taille des registres et est fonction de la précision voulue (SP,DP).

Exemple : perf crête du processeur intel westmere-EP à 2.26GHz

Processeur à 6 cores ,

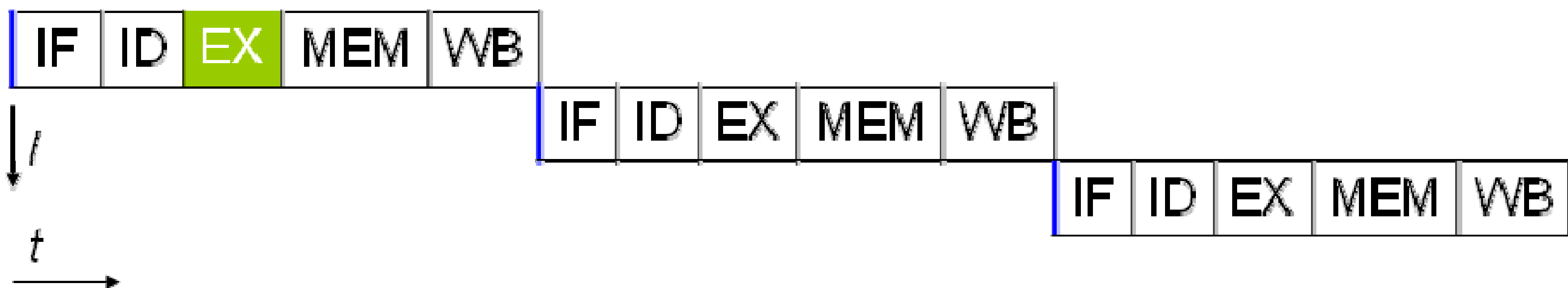
chaque core a 2 unités FPU capables d 'exécuter 2 opérations 64 bits

6 * 2 * 2 * 2.26 = 54.24 Gflops crête en Double Précision

Les différentes étapes d'exécution d'une opération

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

- IF : Instruction Fecht
- ID : Instruction Decode/Register Fecht
- EX : Execution/Effective Address
- MA : Memory Access/ Cache Access
- WB : Write-Back

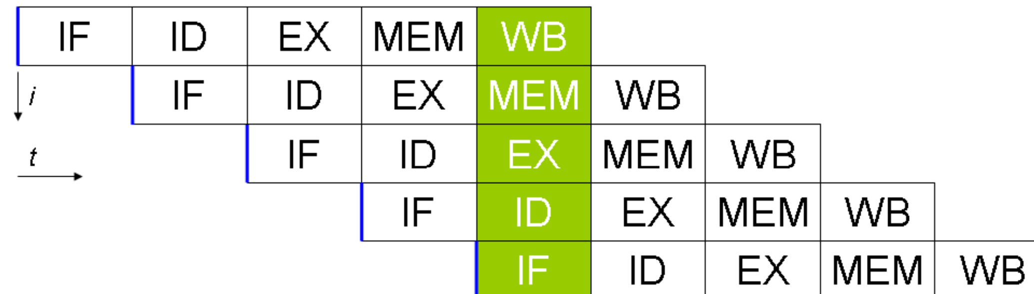


Sur un processeur sans pipeline les instructions sont exécutées les unes après les autres.

Exécution simultanée de plusieurs instructions : ILP (pipeling)

Instruction Level Parallelism

Les différentes étapes d'une opération s'exécutent simultanément sur des données distinctes (parallélisme d'instructions).

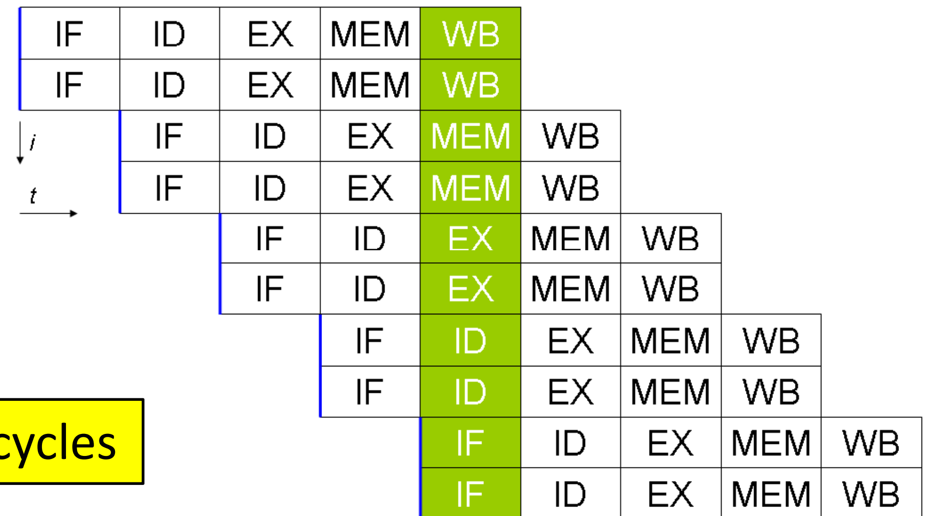


5 instructions en parallèle en 9 cycles (25 cycles en séquentiel)
➔ permet de multiplier le débit avec lequel les instructions sont exécutées par le processeur.

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 (faible nombre d'instructions)
- Execution spéculative : faire une hypothèse sur la suite d'un traitement après un branchement conditionnel (la valeur de la condition n'étant pas encore calculée).

Les problèmes de dépendances entre instructions :

- Partage des ressources (par exemple la mémoire)
- Dépendances des données entre instructions
- Dépendances des contrôles : une instruction est une branche.

Appliquer la même instruction à plusieurs données :

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

Exemples de jeux d'instructions SIMD

MMX : permettent d'accélérer certaines opérations répétitives dans certains domaines :traitement de l'image 2D, son et communications.
SSE (SSE2,SSE3, SSE4, SSE5 annonçé par AMD) : par ex instructions pour additionner et multiplier plusieurs valeurs stockées dans un seul registre
3DNow : jeu d'instructions multimédia développé par AMD
AVX (Advanced Vector Extensions) : nouvel ensemble d'instructions sur les proc SandyBridge intel, unité SIMD de largeur 256 ou 512 bit

- 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

Ex : NEC SX, Fujitsu VPP, Altivec)

Dépasser les limites en « aidant » le processeur

Une des limites du parallélisme d'instruction :

Pour extraire le parallélisme, le compilateur « sérialise »le code, dont le parallélisme intrinsèque doit être redécouvert dynamiquement par le processeur

- Les architectures VLIW (Very Long Instruction Word) et EPIC (Explicitly Parallel Instruction Set Computing) permettent de traiter des instructions longues, aggrégations d'instructions courtes indépendantes, pour les exécuter explicitement en //

  VLIW : le compilateur a la charge d'organiser correctement les instructions, tout en respectant les types de dépendances habituelles qui sont normalement gérées au niveau matériel sur les architectures classiques

  → On gagne en performance (pas de contrôle), mais la compatibilité binaire en générations successives est quasi-impossible. (Ex: Itanium)

Améliorer le remplissage du flot d'instructions du processeur : TLP (Thread Level Parallelism)

- 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.

Performance : mémoire versus µprocesseur

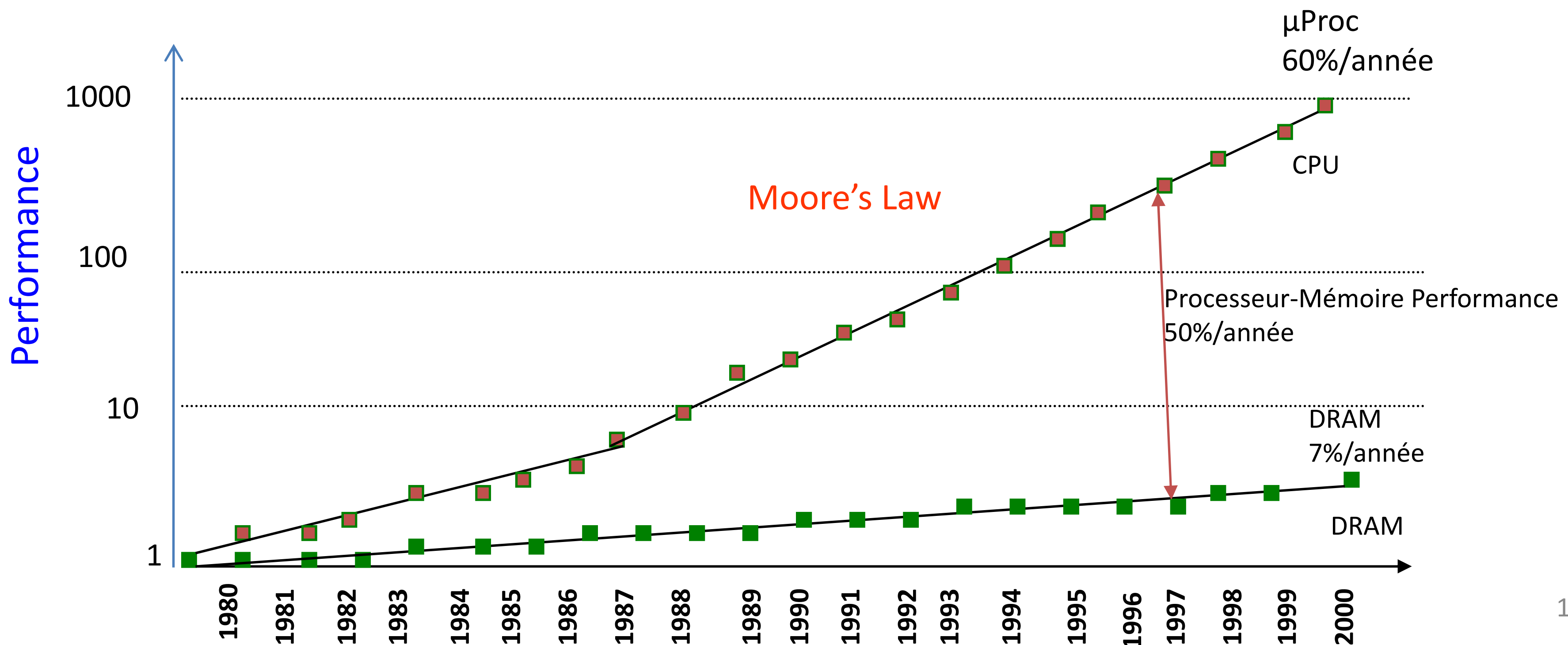
Cas idéal : Mémoire la plus grande et la plus rapide possible

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 >> temps cycle processeur.

- Bande passante = nombre d'accès par unité de temps.



Hiérarchisation de la mémoire

* les caches peuvent être organisés de façon hiérarchique

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, à chaque itération accès 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 = 1,1000
    DO I = 1,10000
        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 granularité :
  - le CPU travaille sur des « mots » (par ex 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 de k lignes. La taille de ces ensembles est constante et est appelée le degré d'associativité. Chaque adresse a une alternative de k lignes.

Effets des paramètres des caches sur les performances

- Cache plus grand

☺ réduit les « cache miss »,

☹ augmente le temps d'accès.

- Plus grand degré d'associativité

☺ réduit les conflits de cache,

☹ peut augmenter le temps d'accès.

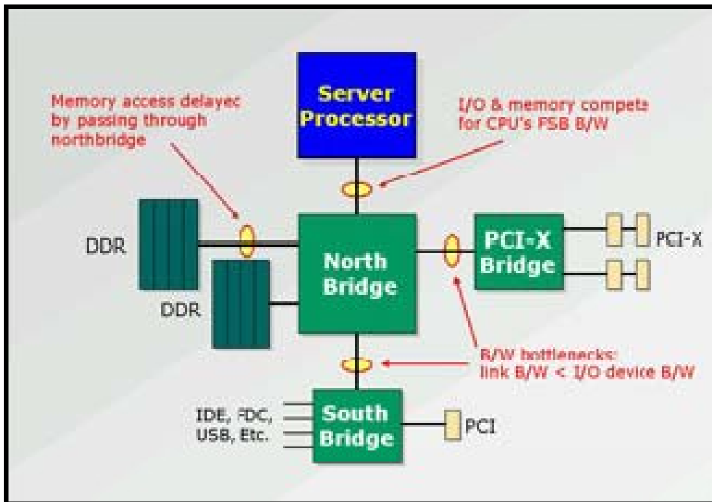
Prefetching

Le prefetching consiste à anticiper les données et les instructions dont le processeur aura besoin, et ainsi les précharger depuis la hiérarchie mémoire (le L2 ou la mémoire centrale).

Différents type de prefetching :

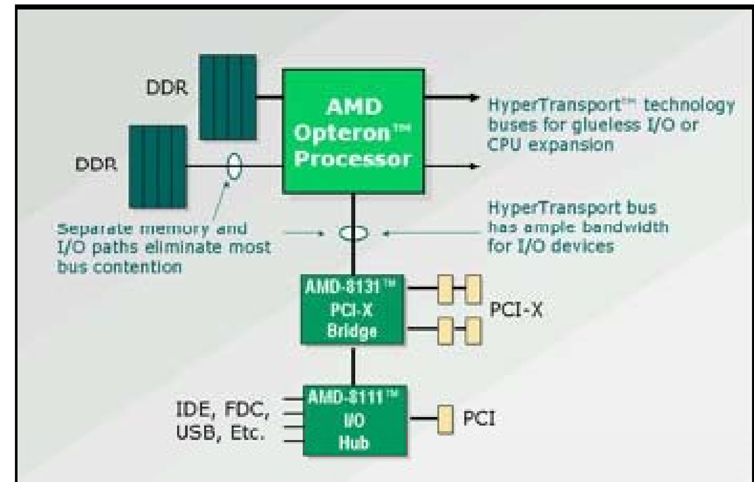
- mécanisme hardware de préchargement,
- mécanisme software de préchargement

Les communications internes au processeur



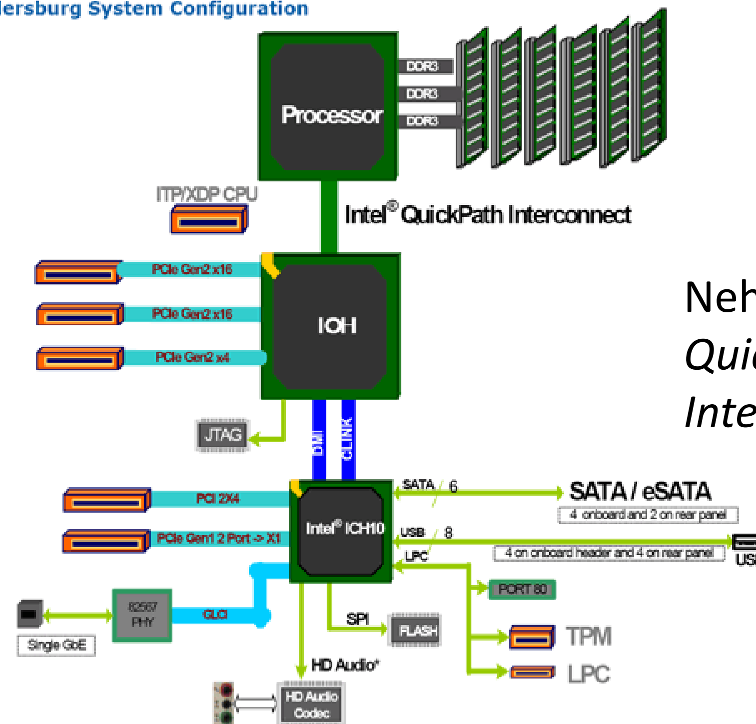
Xeon

Northbridge



Opteron

Hypertransport



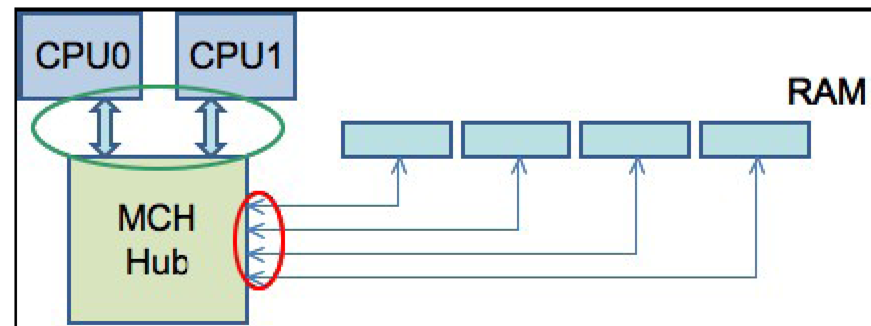
Nehalem

QuickPath

Interconnect

Communications CPU - Mémoire

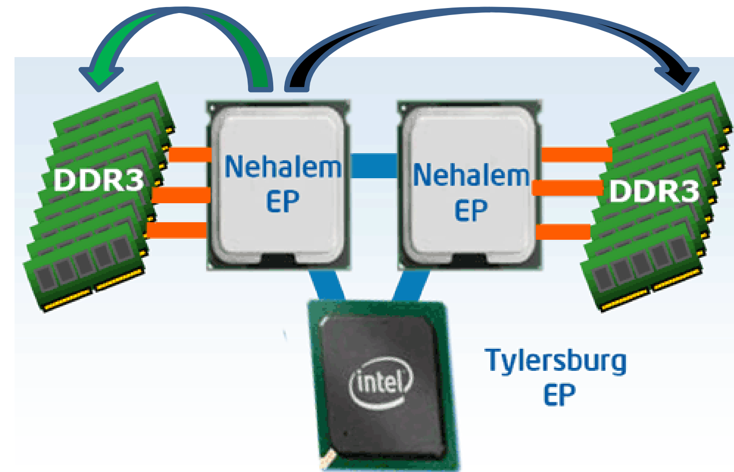
Exemple du Northbridge sur architecture Intel:
Pour évaluer la bande passante entre CPU et mémoire, il faut
Connaître la fréquence du FSB.
Fréquence la + élevée : 1 600 MHz.



Exemple : calcul de la bande passante mémoire locale sur un noeud bi-sockets Intel Harpertown :
Bande Passante au niveau de l'interface processeurs : sur une carte bi-sockets
2 FSB 64 bits à 1 600 MHz, soit 2*1600*8 = 25.8 Go/s.
Bande Passante au niveau de l'interface mémoire : la mémoire est adressée via 4 canaux à 800 MHz (si FBDIMM 800 MHz). La largeur du bus est de 72 bits, dont 8 bits de contrôle, soit 64 bits de données. Ce qui donne : 4*800*8 = 25.6 Go/s.

Communications CPU - Mémoire

Exemple du Nehalem avec le QuickPathInterconnect :
Le contrôleur mémoire est intégré au CPU
.



Exemple : calcul de la bande passante mémoire locale sur un noeud bi-sockets Intel Nehalem
Bande Passante vers mémoire locale au CPU :
3 canaux 64 bits vers mémoire DDR3 1066 MHz, soit 3*1066*8 = 25.6 Go/s.
Un processeur peut également adresser la mémoire connecté à un autre processeur NUMA (Non Uniform Memory Access)
Bande Passante vers mémoire d'un autre CPU : le QPI à une bande passante de 6.4 GT /s par lien, unité de transfert de 16 bits, soit 12.8 Go/s dans chaque sens, soit une bande passante totale de 25.6 Go/s.

Communications CPU – IO

Bus interne d’extension

Bus PCI-X

Evolution du bus PCI, préserve la compatibilité.
Interconnect parallèle. Interface entre le bus processeur et le bus d’entrées/sorties. Bus 64 bits, Fréquence jusqu’à 533 MHz.

Bus série full duplex PCI-Express

Utilise une interface série (1 bit donc) à base de lignes bi-directionnelles. En version 2, chaque ligne fonctionne à 500 Mo/s . Le bus PCI-Express peut être construit en multipliant le nombre de lignes

Par ligne : sur PCI-e V 1.x : 20 Mo/S, V2.x 500 Mo/s, V3.0 1 Go/s
Limite de la bande passante sur PCI-Express (16 lignes) :
v 2.0 : 8 Go/s, V3.0 : 16 Go/s

Evolution des architectures

- Loi de Moore : nb de transistors par circuit intégré * 2 tous les 2 ans.
- En 10 ans, la finesse de gravure est passée de 0,25 μm (Pentium III) à 45 *n*m (Core 2 Penryn), Nehalem.
- 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 cores.

Les architectures multicoeurs

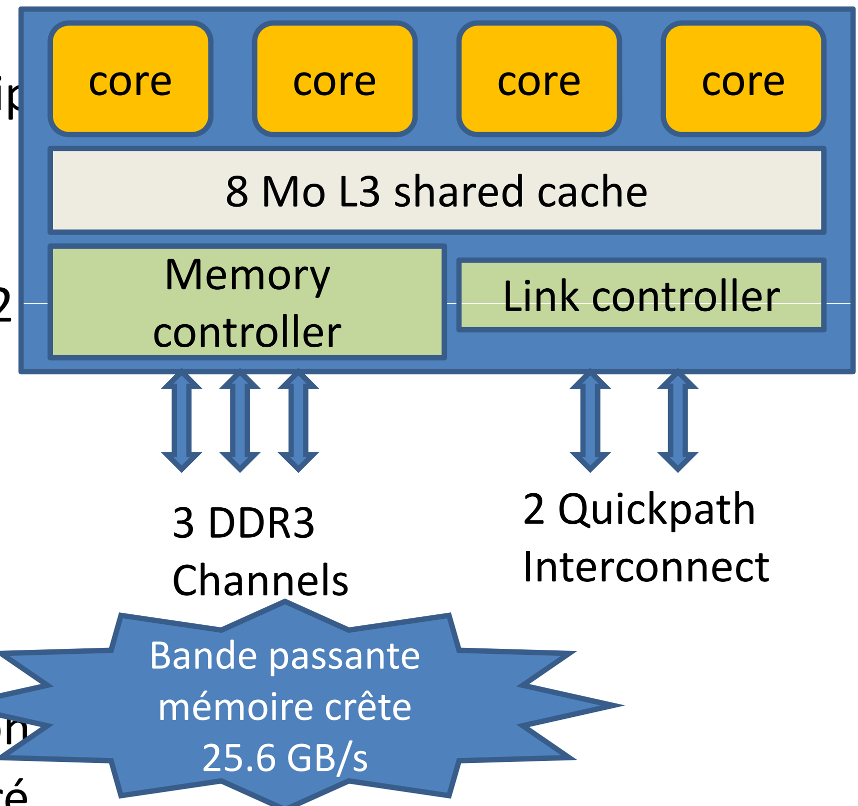
- Un processeur composé d'au moins 2 unités centrales de calcul sur une même puce.
- Permet d'augmenter la puissance de calcul sans augmenter la fréquence d'horloge.
- Et donc réduire la dissipation thermique.
- Et augmenter la densité : les coeurs sont sur le même support, la connectique reliant le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

Pourquoi les multicoeurs ?

Quelques ordres de grandeur			
	Single Core Génération de gravure1	**Dual Core** Génération de gravure2	**Quad Core** Génération de gravure3
Core area	**A**	**~ A/2**	**~ A/4**
Core power	**W**	**~ W/2**	**~ W/4**
Chip power	**W + O**	**W + O’**	**W + O”**
Core performance	**P**	**0.9P**	**0.8P**
Chip performance	**P**	**1.8 P**	**3.2 P**

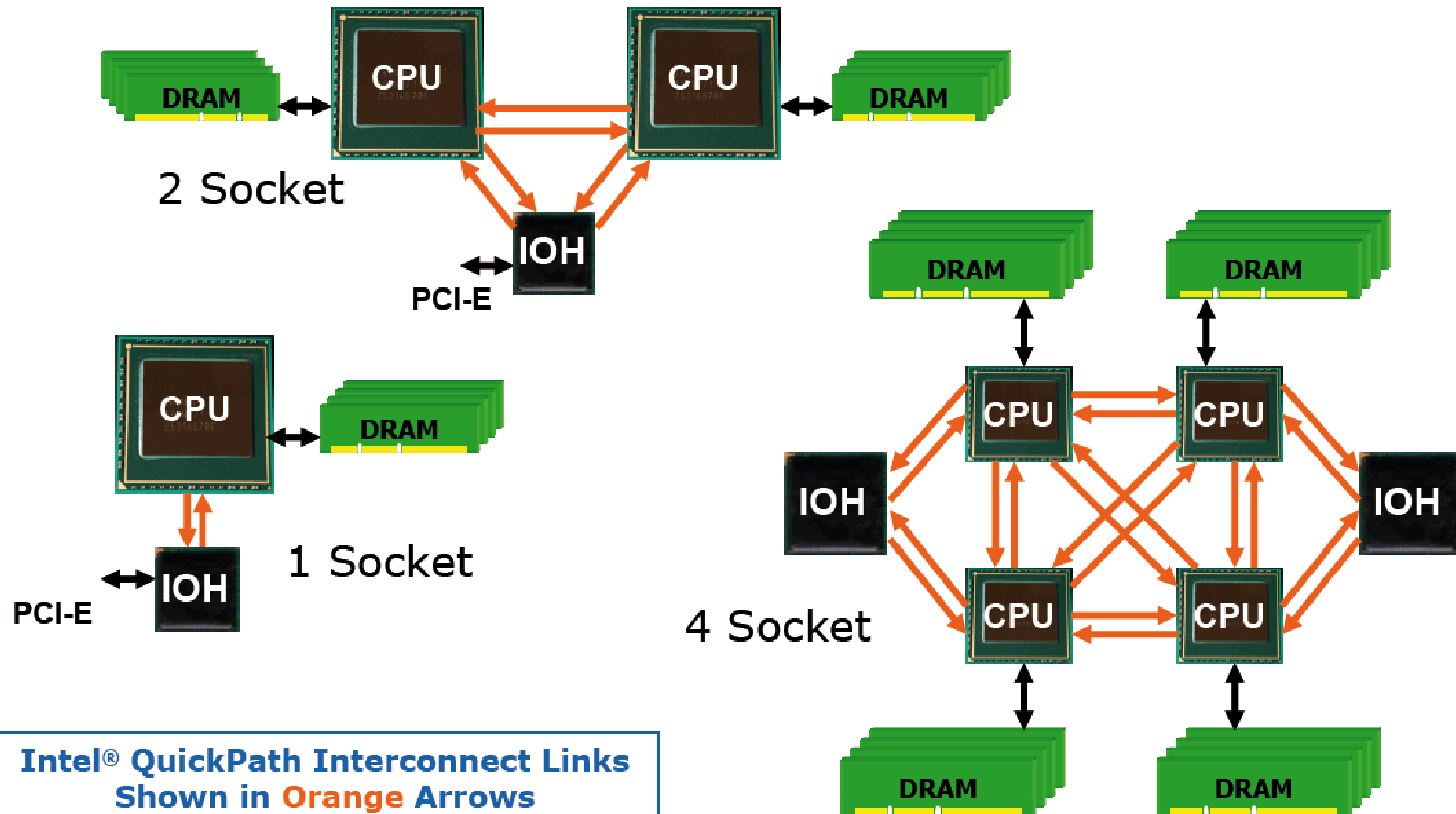
Architecture Nehalem-EP

- 4 cores
- Cache L3 partagé 8 Mo on-chip
- 3 niveaux de cache
  Cache L1 : 32k I-cache + 32k D-cache
  Cache L2 : 256 k par core
  Cache inclusif : cohérence de cache on-chip
- SMT
- 732 M transistors, 1 seule puce de 263 mm2
- QuickPathInterconnect
  – Point à Point
  – 2 liens par socket CPU
  – 1 pour la connexion à l'autre socket
  – 1 pour la connexion au chipset
  – Jusqu'à 6.4 GT/Sec dans chaque direction
- QuickPath Memory controller (DDR3) intégré



Nehalem

Example Platform Topologies



Architecture Sandy Bridge-EP

Devrait arriver début 2012 avec :

- 8 cores par processeur
- 3 niveaux de cache

  Cache L1 : 32k I-cache + 32k D-cache

  Cache L2 : 256 k / core, 8 voies associatif

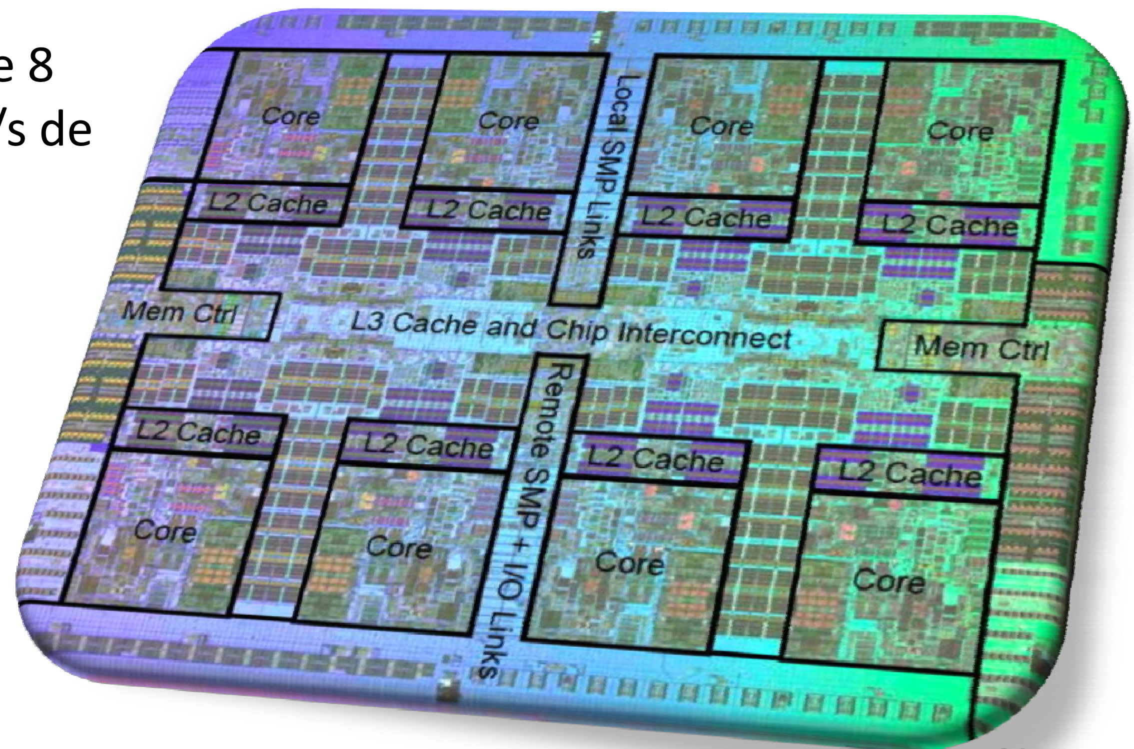
  Cache L3 partagé et inclusif 16 Mo on-chip
- 4 DDR3 memory controller
- Instructions AVX, registres 256 bits
- 8 flop DP/cycle (double du Nehalem)
- 32 lignes PCI-e 3.0 de base
- QuickPathInterconnect

  2 QPI par proc

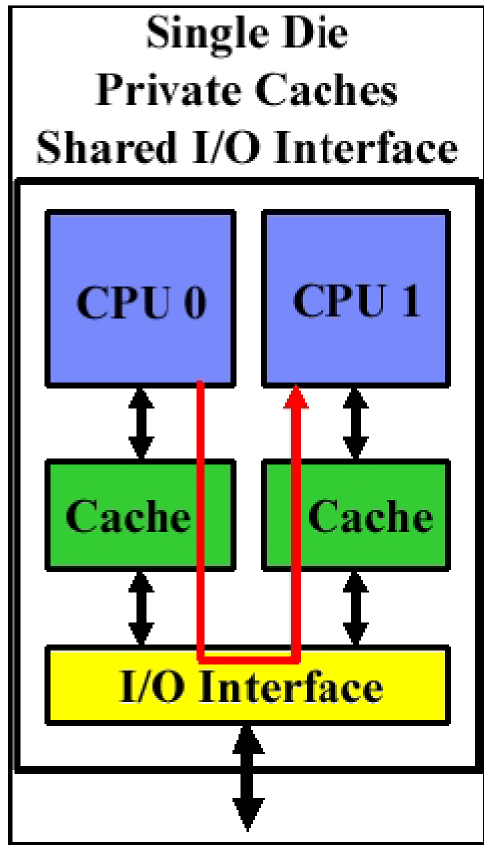
Architecture Power7

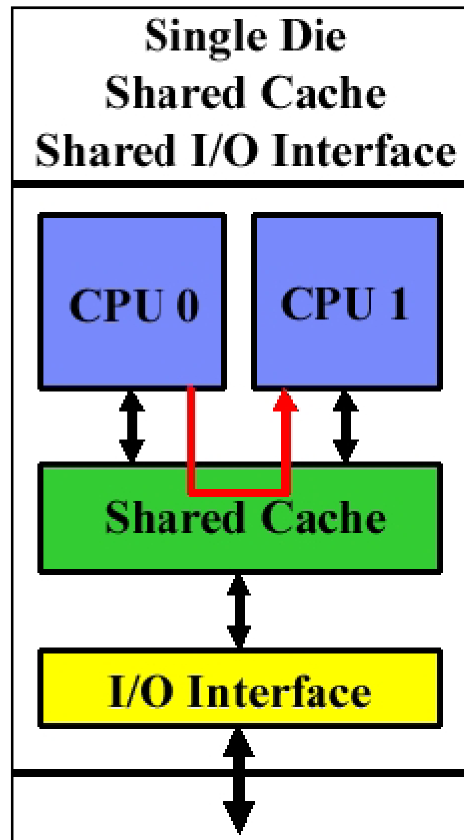
- 1200 M de transistors, 567 mm^2 par die
- jusqu'à 8 cores
- 4 voies SMT ⇨ jusqu'à 32 threads simultanées
- 12 unités d'exécution, dont 4 FP
- Scalabilité : Jusqu'à 32 sockets de 8 cores par système SMP , ↗360Go/s de bande passante/chip
⇨ jusqu'à 1024 threads /SMP
- Cache L2 de 256Ko /core
- Cache L3 partagé en technologie eDRAM (embeddedDRAM)

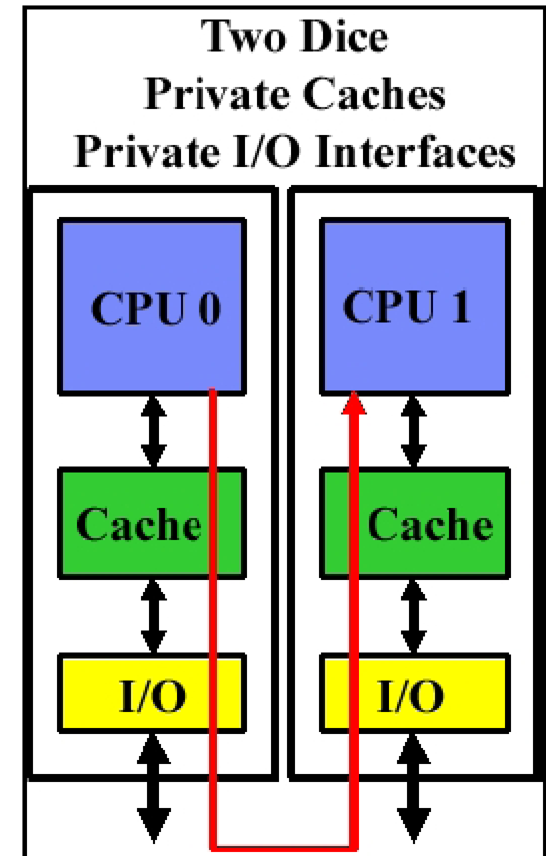
- Contrôleur cache L3 et mémoire sur le chip
- Jusqu'à 100 Go/s de BP mémoire



Architecture des caches







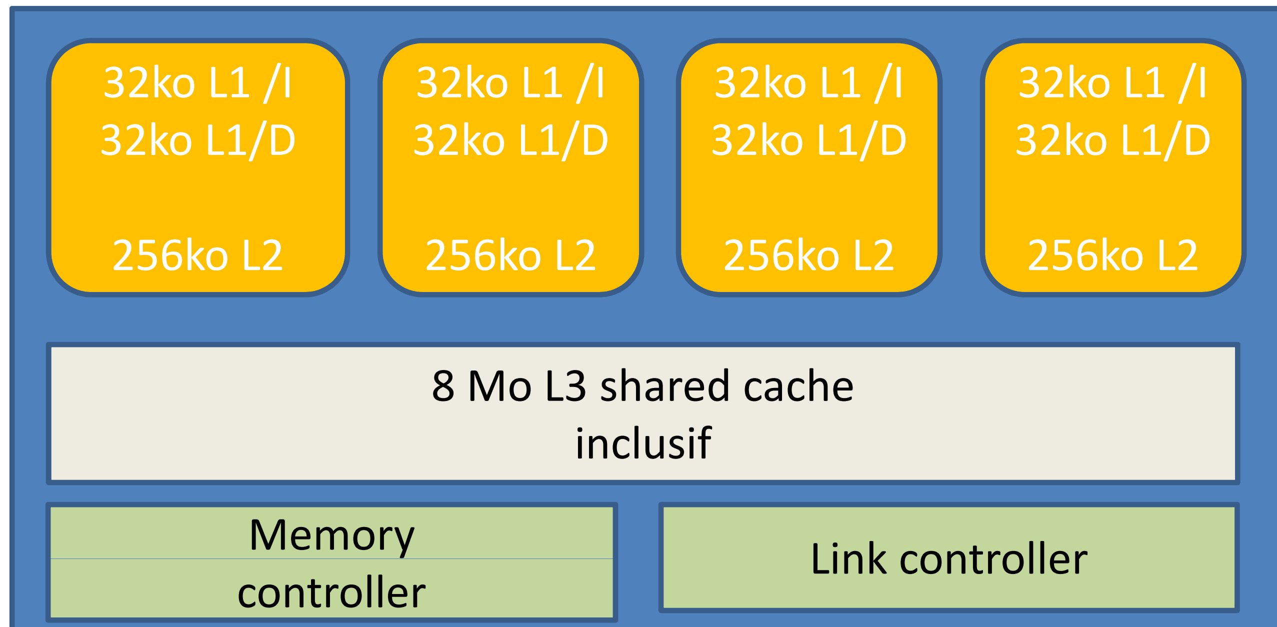
Partage des caches L2 et L3

- Partage du cache L2 (ou L3) :
  - ✓ ☺ communications plus rapides entre coeurs,
  - ✓ ☺ meilleure utilisation de l'espace,
  - ✓ ☺ migration des threads plus facile entre les coeurs,
  - ✓ ☹ contention au niveau de la bande passante et des caches (partage de l'espace),

- Pas de partage entre les caches :
  - ✓ ☺ pas de contention,
  - ✓ ☹ communication/migration plus coûteuse, passage systématique par la mémoire.

  - cache L2 privé, cache L3 partagé :IBM Power5+ / Power6, Intel Nehalem
  - tout privé : le Montecito

Exemple du Nehalem : une hiérarchie de cache à 3 niveaux

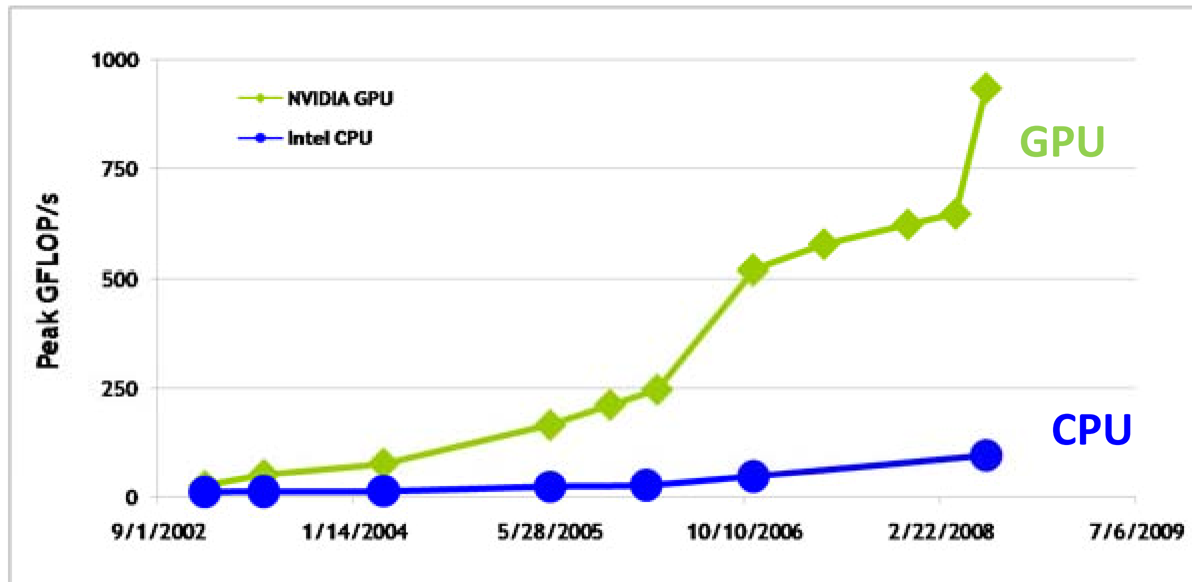


- Cache de niveau 3 inclusif de tous les niveaux précédent

4 bits permettent d'identifier dans quel cache de quel proc se trouve la donnée

  - ✓ ☺ limitation du traffic entre cœurs
  - ✓ ☹ Gaspillage d'une partie de la mémoire cache

Les GPUS
Evolution des performances CPU vs GPU



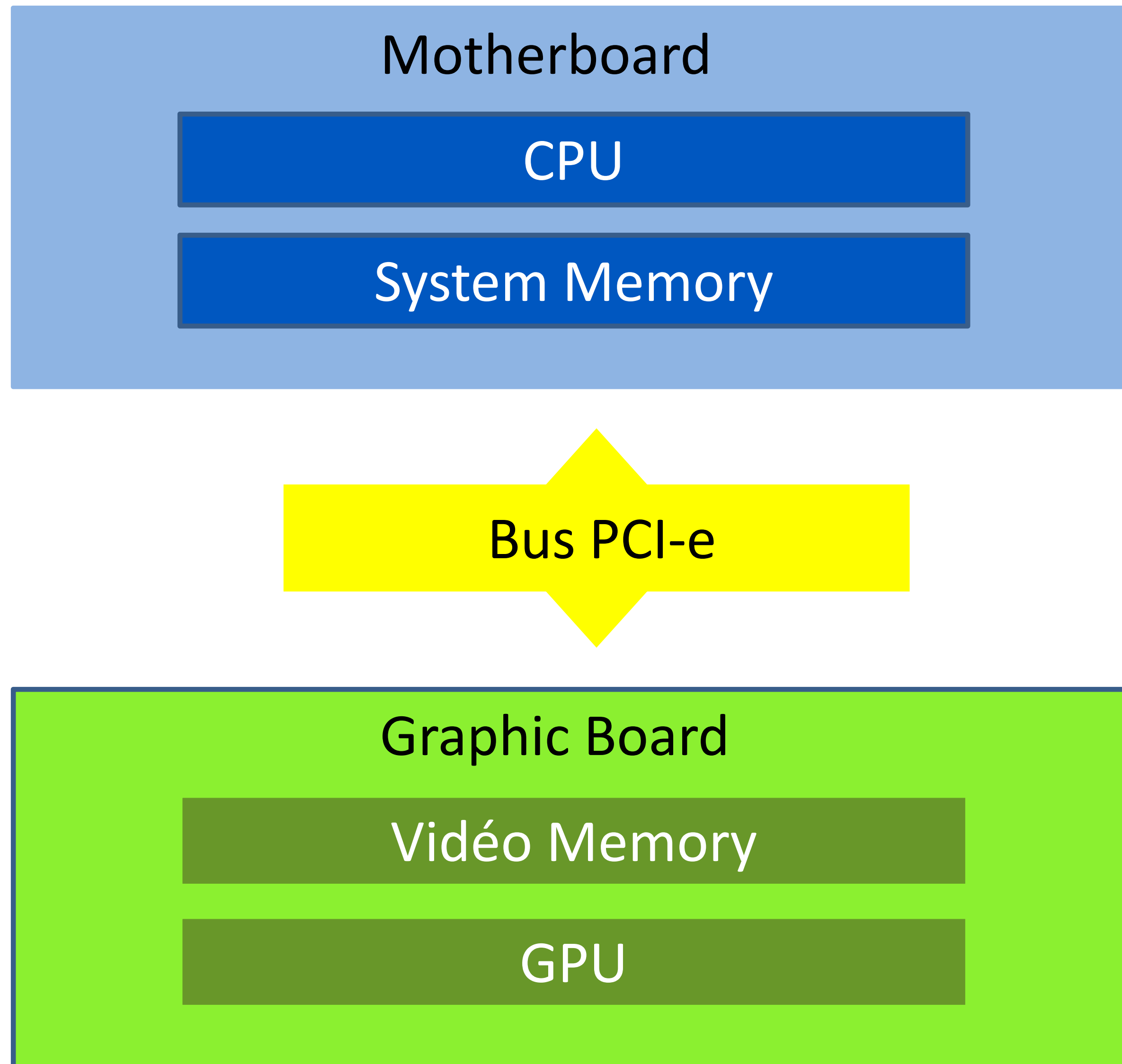
Vitesse des processeurs classiques *** 2 tous les 16 mois**

Vitesse des processeurs graphiques (GPU) ***2 tous les 8 mois**

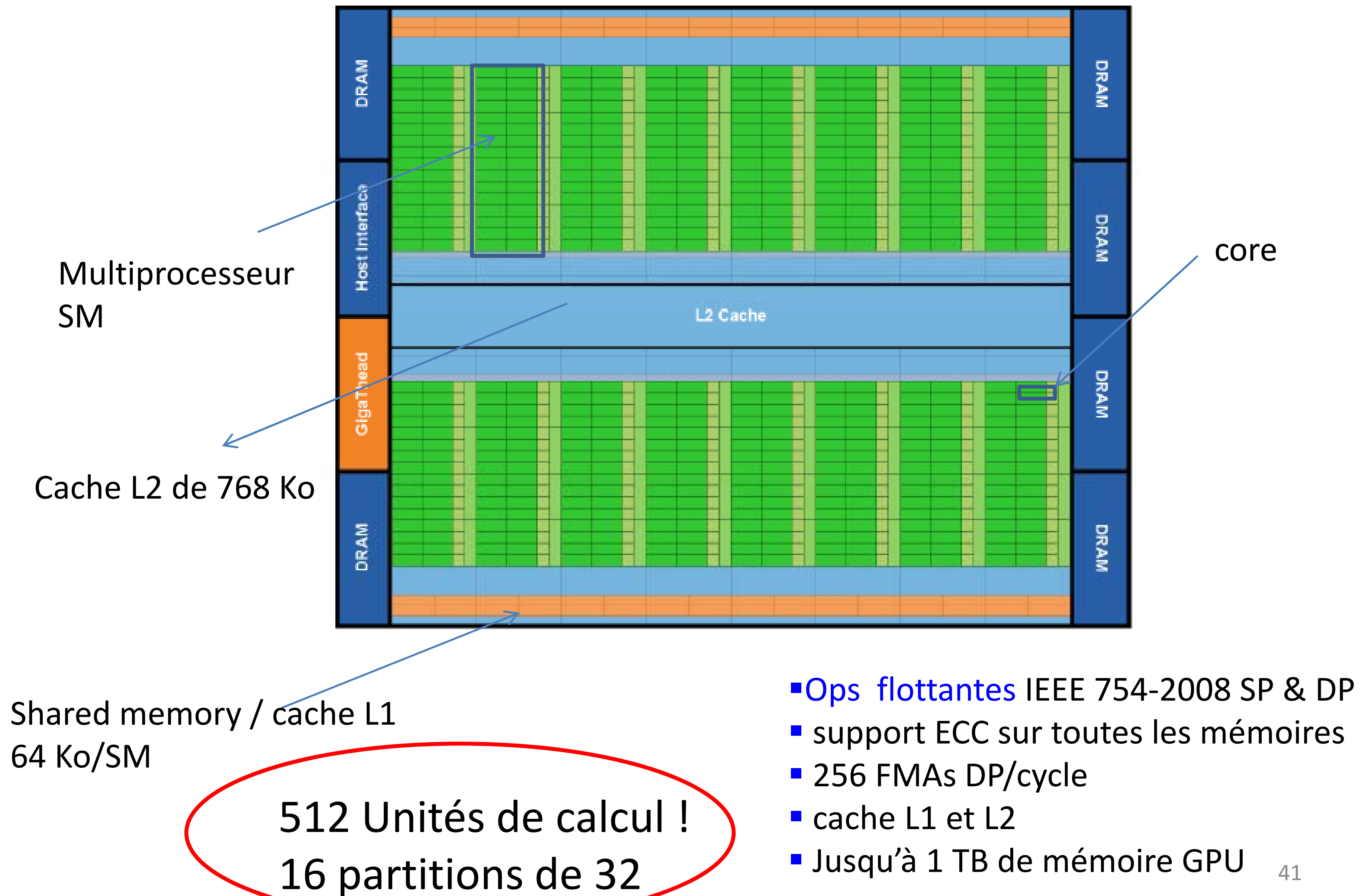
Les processeurs graphiques

- Performance théorique GeForce 8800GTX 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 quelques années, uniquement programmable via des APIs graphiques. Aujourd'hui, des modèles de programmation disponibles : CUDA , OpenCL, HMPP.

Architecture de PC classique

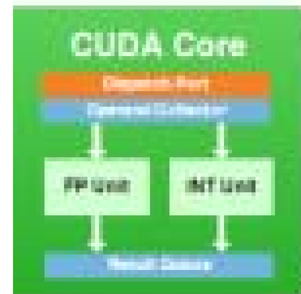


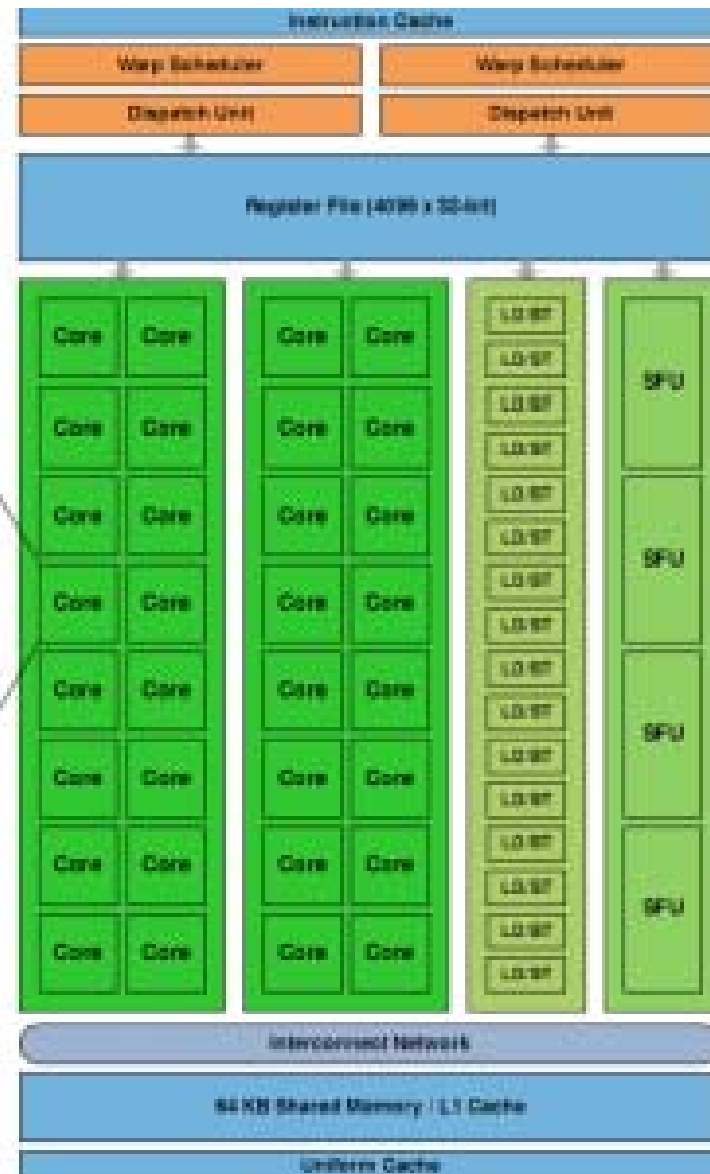
Architecture du processeur Fermi de NVIDIA



Architecture du processeur Fermi de NVIDIA

Le *SM* (Streaming Multiprocessor) du Fermi :
Chaque *SM* a 32 cores.
Un *SM* ordonnance les threads par groupe de 32 threads //





Une évolution importante :
64 Ko de mémoire on-chip
(48ko shared mem + 16ko L1) ,
qui permet aux threads d'un même *block* de coopérer.
Des unités 64 bit.

Structure d'un programme CUDA

- Un programme CUDA comprend :
  - Des portions de code s'exécutant sur l'*hôte* (CPU)
    Typiquement des phases séquentielles, IOs.
  - Des portions qui s'exécutent sur le(s) *device(s)* (GPU)

- Les éléments de calcul sont appelés *kernels* (portion de code s'exécutant sur le(s) *device(s)*)
- Un *kernel* est exécuté en // par de multiples *threads* d'un *block* de thread.
Un *block* de threads est un ensemble de threads qui peuvent coopérer entre elles via des barrières de synchronisation et la mémoire partagée.

Autres langages ou outils

- Brook + (implémentation AMD- des specs de Brook (Stanford)
  - ✓ extension du langage C
- OpenCL (Open Computing Language):
  - ✓ interface de programmation basée sur du C avec des extensions du langage
  - ✓ portable
  - ✓ calcul parallèle sur architecture hétérogène : CPUs, GPUs, et autres processeurs.
- HMPP (CAPS Entreprise)
  - ✓ Expression du parallélisme et des communications dans le code C ou Fortran à l'aide de directive à la « OpenMP »
  - ✓ HMPP gère les communications CPU-GPU
- StarPU (projet INRIA)
  - ✓Gestion dynamique des tâches sur différents types d'accélérateurs de calcul multi-cœurs, un modèle unifié d'exécution
- S_GPU (projet INRIIA)
  - ✓ Partage l'utilisation de plusieurs GPU avec plusieurs processus MPI placés sur des CPU

Comparaison GPU /CPU

A performance intrinsèque égale , les plateformes à base de GPUs :

- occupent moins de place
- sont moins chères
- sont moins consommatrices d'électricité

Mais

- sont réservées à des applications massivement parallèles
- Transferts par le PCI-e limitent fortement les performances
- nécessitent l'apprentissage de nouveaux outils
- quelle est la garantie de pérennité des codes et donc de l'investissement en terme de portage ?

Le processeurs MIC (Many Integrated Core) d'Intel : une réponse au GPU ?

- Des processeurs « manycores », ≥ 50 cores sur la même puce

- Compatibilité x86
  Support des logiciels Intel

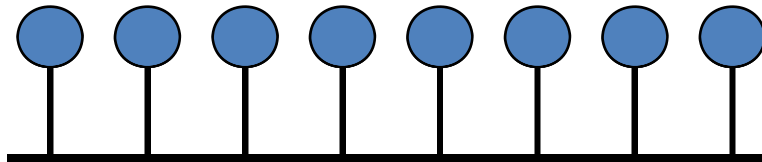
- Actuellement à l'état de prototype,
  sortie du « Knights Corner » fin 2012 ou début 2013

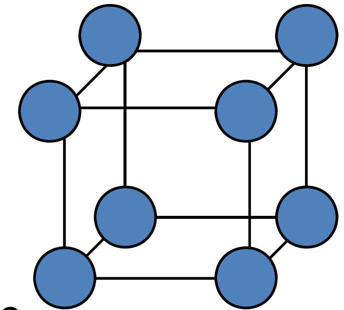
Technologie réseau pour le HPC

- Les processeurs ou les machines peuvent être reliés de différentes façons
- Les caractéristiques importantes :
  - La bande passante (bandwidth) est le débit maximal
  - Latence : temps d'initialisation de la communication
  - Distance maximum entre deux nœuds
- Différents types d'interconnexion:
  - Bus
  - Switch crossbar
  - Hypercube
  - Tree

Topologie réseau

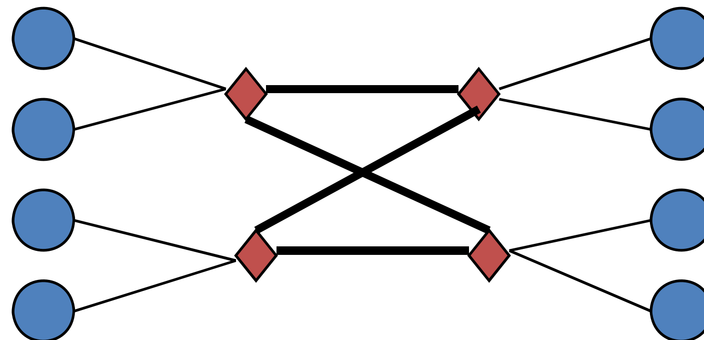
Bus



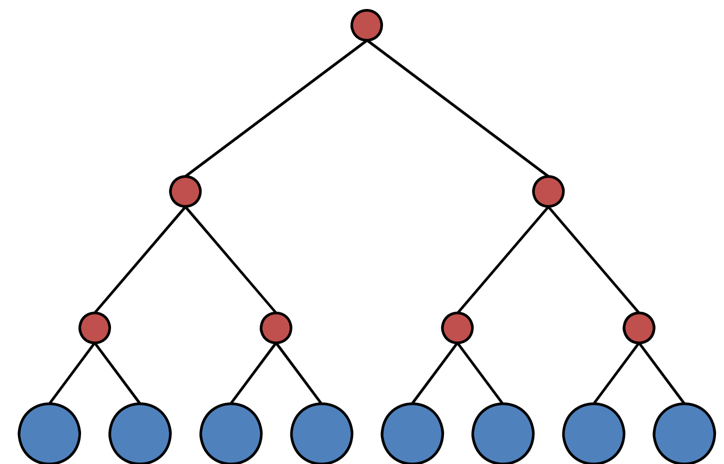


Hypercube
Chaque proc est le
Sommet d'un hypercube
de dim n

Réseau à
plusieurs
étages



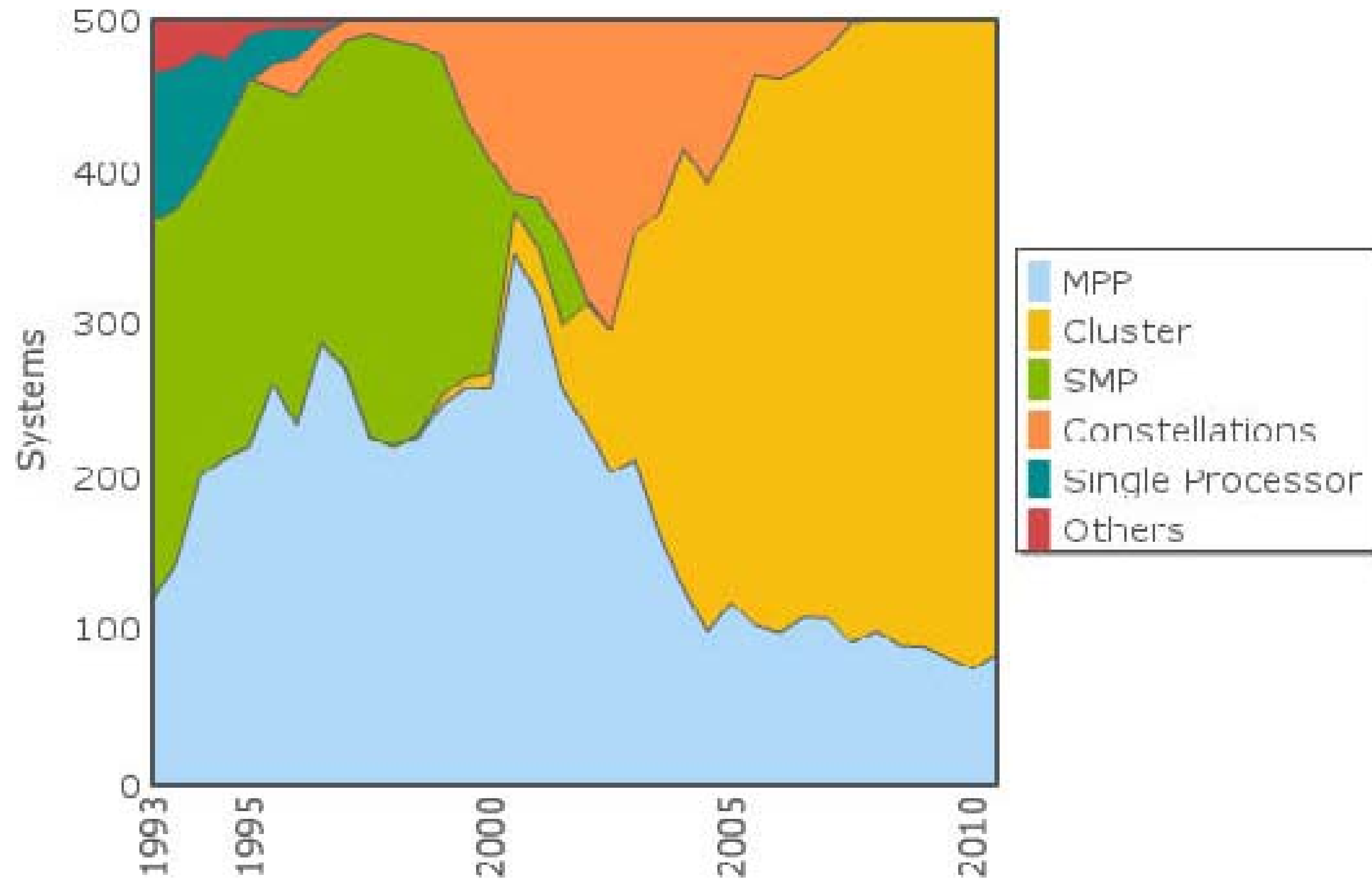
Arbre



Technologie réseau pour le HPC

Technologie	Vendeur	Latence MPI usec, short msg	Bande passante par lien unidirectionnel, MB/s
NUMAlink 4	SGI	1	3200
QsNet II	Quadrics	1.2	900
Infiniband	Mellanox, Qlogic, (Voltaire)	1.07	2500 (SDR) Double speed DDR Quad speed QDR Fourteen speed FDR (14Gb/s)
High Performance Switch	IBM	5	1000
Myri-10G	Myricom	2.3	2500
Ethernet 10/40 Gb		4	2600
Ethernet 1 Gb			50 à 80

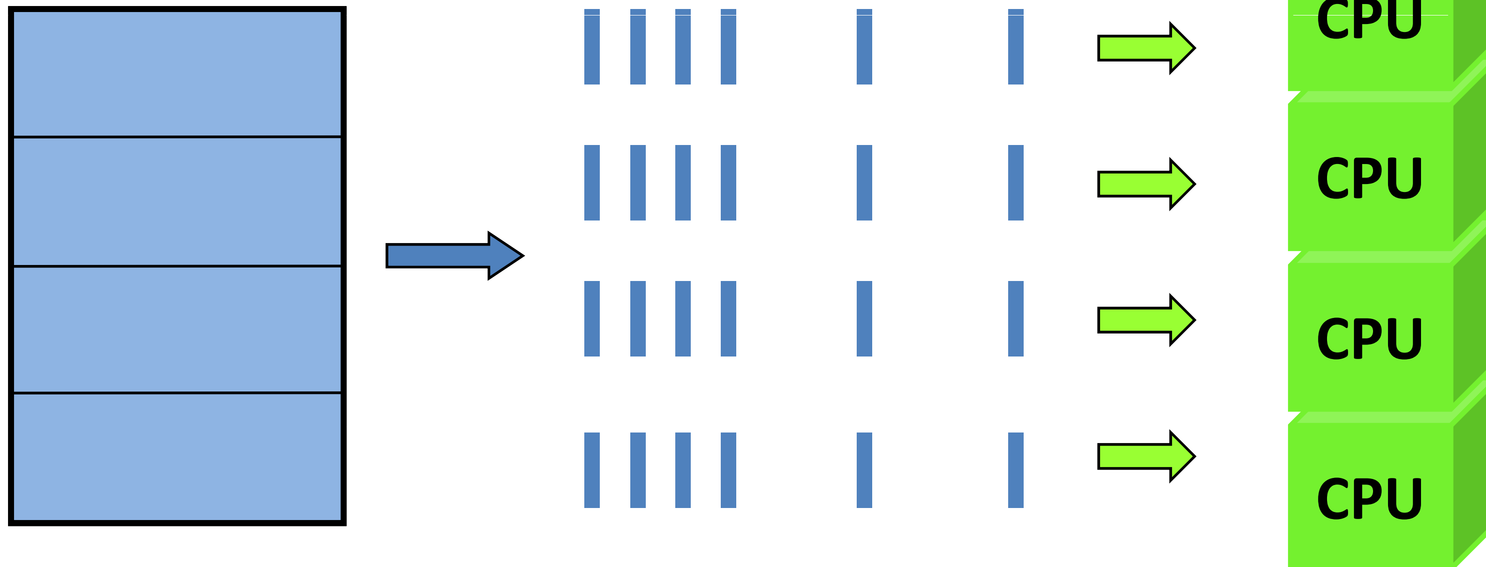
Evolution proportion des différents types de systèmes ces dernières années



Concepts du parallélisme

Calcul parallèle

Un calcul parallèle est exécuté sur plusieurs unités de calcul
Le calcul est découpé en plusieurs parties pouvant s'exécuter simultanément
Chaque partie est découpée en séquences d'instructions
Des instructions de chaque partie s'exécutent simultanément sur # CPU



Calcul paralléle

Les ressources de calcul peuvent être

- Une seule machine avec un ou plusieurs processeurs
- Plusieurs machines avec un ou plusieurs processeurs interconnectées par un réseau rapide
- Une grille de calcul : plusieurs ensembles de plusieurs machines

Si un CPU peut exécuter les opérations arithmétiques du programme en temps t, alors n CPU peuvent idéalement les exécuter en temps t/n

Pourquoi utiliser le parallélisme ?

- Economiser du temps (à la montre)
- Résoudre des problèmes de + grande taille
- Avoir accès à + de mémoire et + de puissance en cumulant les ressources matérielles
- Pouvoir traiter plusieurs choses en même temps
- Les limites du séquentiel :
  - Limites de la vitesse de transmission (transmission des données entre les différents éléments du hardware)
  - La taille mémoire
  - Les limites de la miniaturisation
  - Les limites économiques : + cher d’augmenter la rapidité d’un processeur que de multiplier les processeurs ou les cores

Classification de Flynn

- Classification des architectures selon 2 dimensions : instruction, data.

SISD Single Instruction, Single Data	SIMD Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	MIMD Multiple Instruction, Multiple Data

Classification de Flynn

- Flux/flot d’instructions : séquence d’instructions exécutées par la machine
- Flux/flot de données : séquence de données appelées par le flux d’instructions

SISD : architecture séquentielle avec un seul flot d’instructions, un seul flot de données, exécution déterministe
SIMD : architecture // , ttes les unités de traitement exécutent la même instruction à un cycle d’horloge donnée, chaque unité peut opérer sur des données différentes, exécution déterministe

Arrays processors
Machines vectorielles (instructions sur des vecteurs de données)
Exécution pipelinée

Classification de Flynn

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

MIMD: architecture la plus courante aujourd'hui,

Chaque unité de traitement peut gérer un flot d'instructions différent

Chaque unité peut opérer sur un flot de données différent

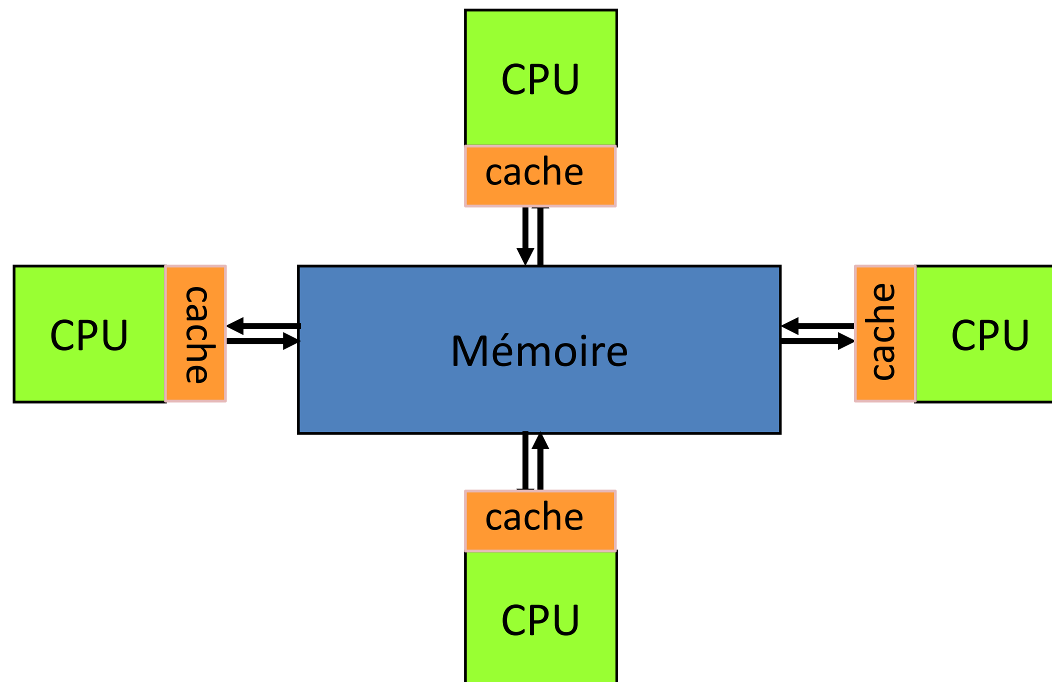
L'exécution peut être synchrone ou asynchrone, déterministe ou non-déterministe

Terminologie

- **Tâche parallèle** : portion de travail qui peut s'exécuter sur plusieurs CPU, sans risque sur la validité des résultats
- **Mémoire partagée** :
  - ✓ D'un point de vue hard, réfère à une machine dont tous les CPUs. ont un accès direct à une mémoire ϕ commune
  - ✓ 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.
- **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 d'une machine distante, sur laquelle d'autres tâches s'exécutent.

Mémoire partagée

- Tous les CPUs accède à toute la mémoire globale avec un même espace d'adressage global
- Chaque CPU 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 CPUs ont leur propre mémoire locale (cache)



Mémoire partagée : UMA / NUMA

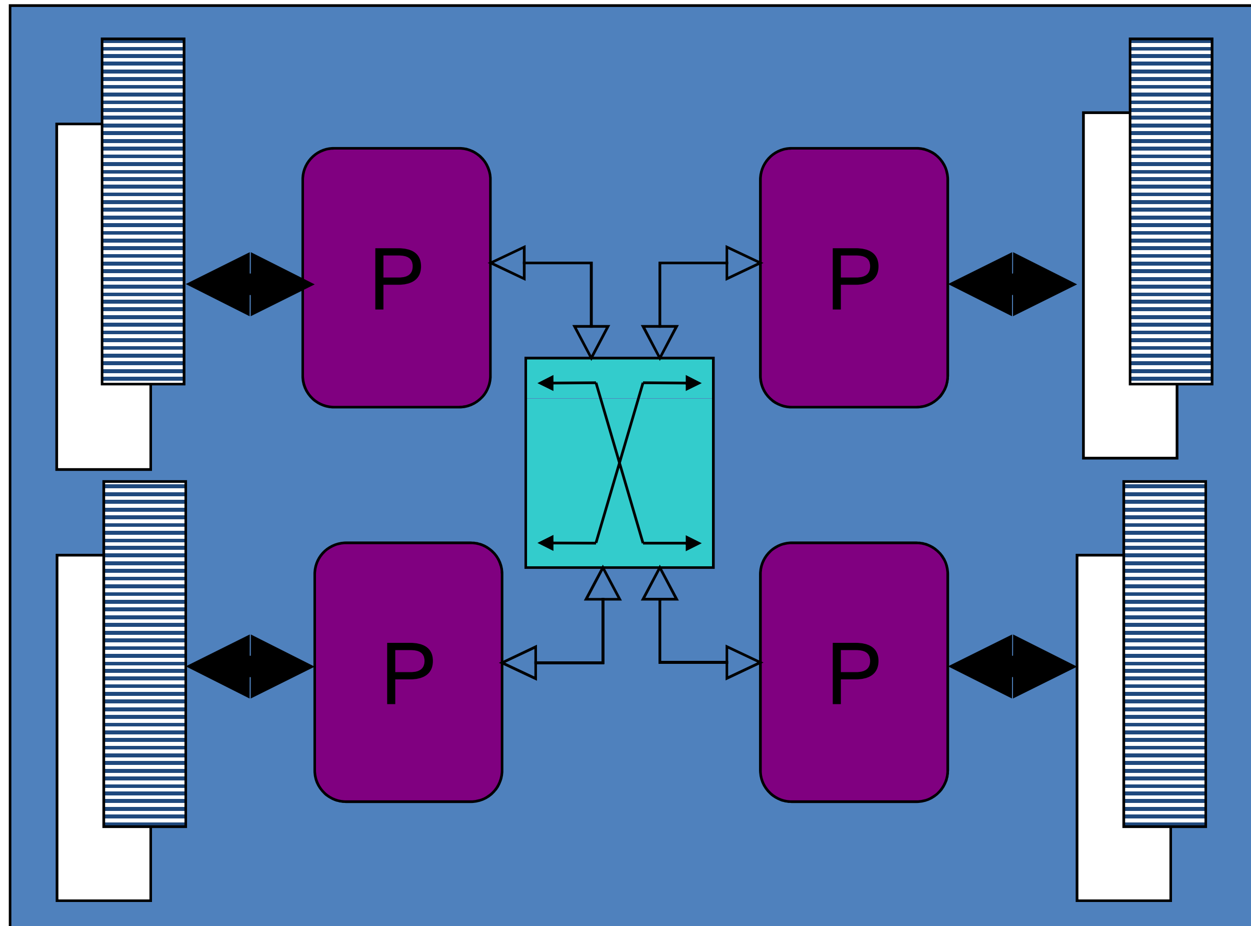
2 classes d'architecture à mémoire partagée, différenciées par les temps d'accès mémoire :

- **UMA** **(**Uniform memory access) :
  - Des CPUs identiques, ayant tous le même temps d'accès à la mémoire

- **NUMA** (Non Uniform memory access) :
  - Conçue pour pallier aux problèmes d'accès mémoire concurrents via un unique bus .
  - En général, plusieurs blocs SMP interconnectés
  - Les 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

Architecture NUMA



Une architecture mémoire physiquement distribuée mais logiquement partagée

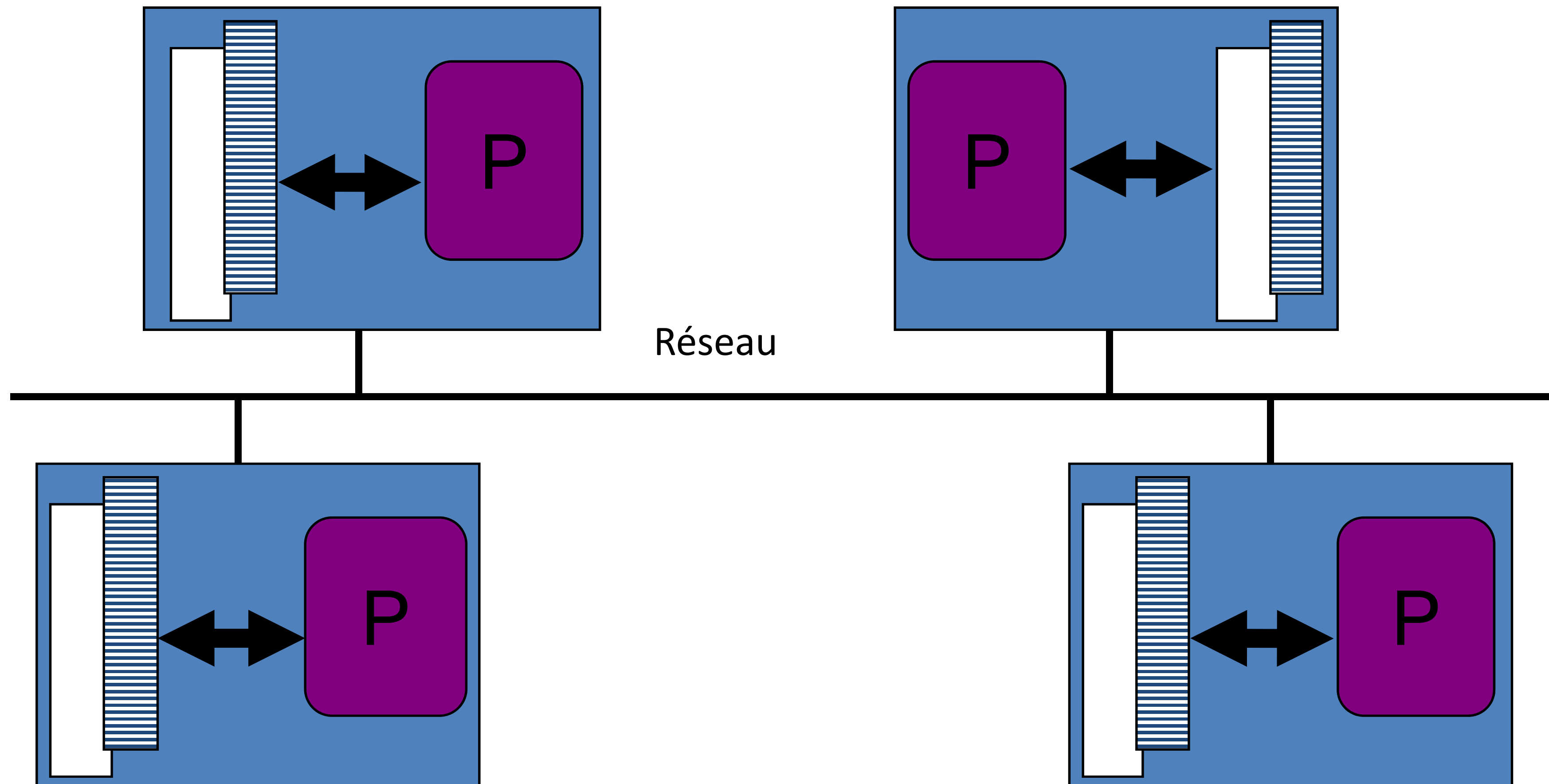
Mémoire partagée

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

Mais :

- Manque de scalabilité : augmenter le nombre de CPU accroît le traffic sur le chemin d'accès à la mémoire
- Le programmeur doit gérer la synchronisation pour un accès correcte à la mémoire
- Coûteux de construire des architectures à mémoire partagée avec un grand nombre de processeurs

Mémoire distribuée



Le point commun de ces architectures : elles possèdent un réseau d'interconnexion pour communiquer entre les mémoires des différents processeurs.

Mémoire distribuée

- Chaque processeur à sa propre mémoire locale, il n’y a pas de notion d’espace d’adressage globale 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

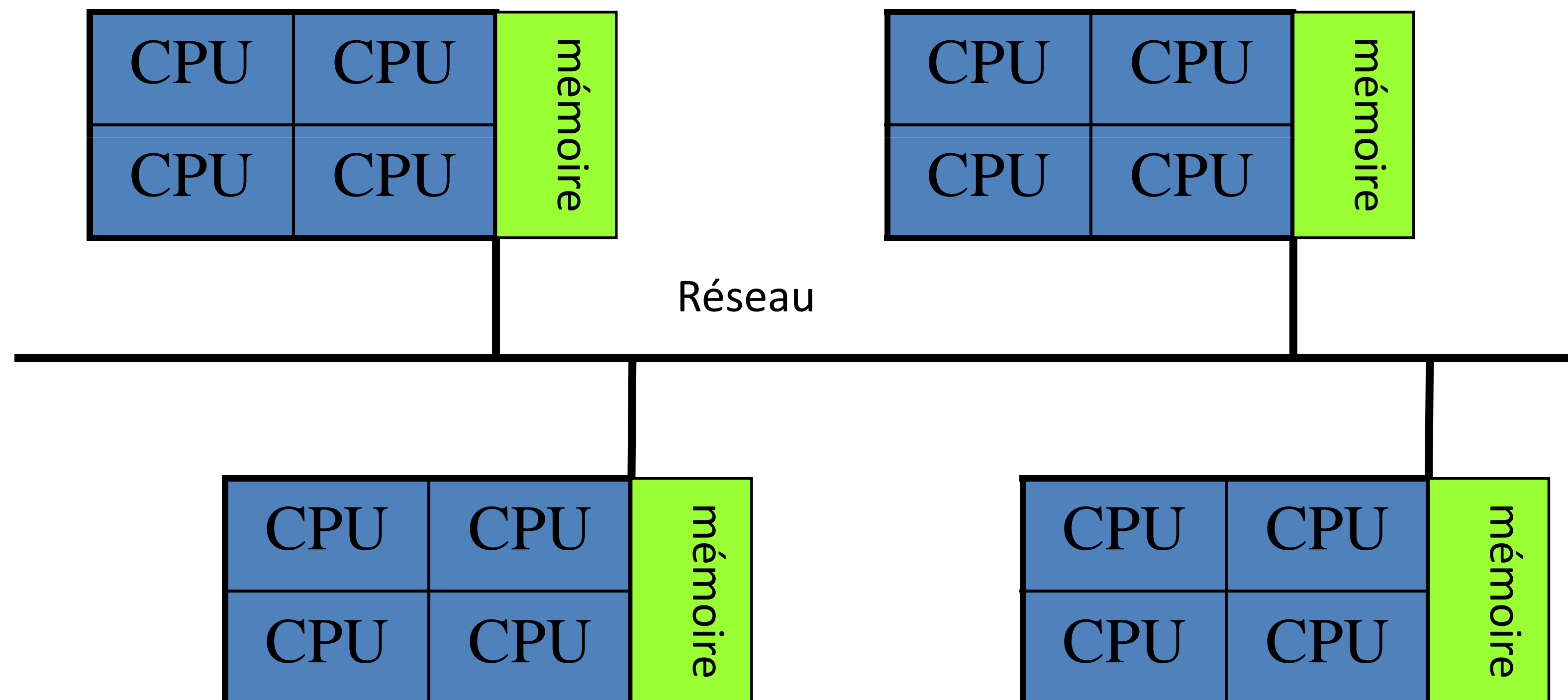
- On peut augmenter le nombre de processeur et la mémoire proportionnellement
- Accès rapide à la mémoire locale sur chaque proc
- Pas de problème de cohérence des accès
- Coût raisonnable ex : PCs en réseau

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 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, …)



Modèles de programmation parallèle

Modèles de programmation parallèle

- Modèle à mémoire partagée
- « threads »
- Passage de messages
- Parallélisme de données
- Modèle hybride

Ces modèles existent au dessus du hardware et des architectures mémoire et peuvent théoriquement être implémentés sur n'importe quel hardware.

Les + implémentés :

✓ MPI (passage de messages) sur architectures distribuées et hybrides (mais souvent aussi sur architecture à mémoire partagée)

✓ OpenMP (mémoire partagée) sur architecture à mémoire partagée

Modèle à mémoire partagée

- Les tâches partagent un espace d'adressage commun, dans lequel elles lisent et écrivent de façon asynchrone
- # mécanismes tels *locks* (verrous), *sémaphores* sont utilisés pour l'accès à la mémoire partagée
- L'avantage du point de vue de l'utilisateur est qu'il n'y a pas la notion de « propriétaire d'une donnée », il n'a pas à décrire explicitement les communications
- Mais attention, la localité des données a une impacte sur les performances mais n'est pas facile à gérer

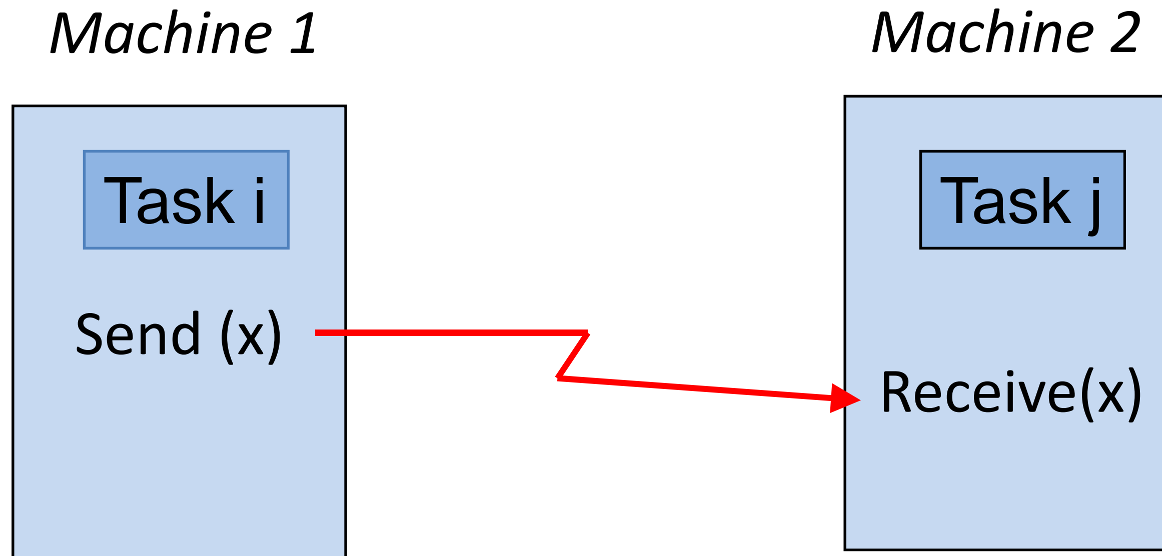
Threads

- Dans ce modèle, un unique processus peut initier plusieurs tâches concurrentes, appelées threads
- Chaque thread a sa mémoire locale, mais partage également la mémoire globale du processus
- Les threads communiquent entre elles via cette mémoire globale
- Le programme principal est présent durant toute l'exécution, alors qu'une thread peut apparaître puis disparaître et réapparaître.

Modèle Passage de Messages

- Un ensemble de tâches qui utilisent leur mémoire locale, plusieurs de ces tâches pouvant être sur une même ou sur plusieurs machines
- Ces tâches échangent des données via des communications de messages (send/receive)
- Un transfert de données requiert en général une coopération entre les 2 tâches, les opérations sont spécifiées par le programmeur

Modèle Passage de Messages



A un envoi doit correspondre une réception
Le programmeur doit spécifier les opérations de façon explicite
Implémentation : librairie de routines auxquelles le programmeur peut faire appel dans son code
Ex : PVM (le standard jusqu'en 92), MPI (le standard actuel)
Versions domaine public : MPICH, LAM/MPI, OpenMPI, MVAPICH

Modèle Passage de Messages

- Si les communications sont entre 2 tâches sur 2 machines #, elles passent par le réseau
- Si les communications sont entre 2 tâches sur une même machine, les implémentations de MPI choisissent en général de les faire passer par la mémoire partagée (copies mémoire) pour des raisons de performance

Modèle “Data Parallel”

- Un ensemble de tâches travaillent collectivement sur une même structure de données, chaque tâche effectue la même opération mais sur une partie différente de la structure
- Un programme data parallel est souvent écrit en utilisant des constructions spécifiques dites « data parallel », et/ou des directives de compilation
- Ex :

  HPF (High Performance Fortran) : extension du fortran90 pour le parallélisme de données

Modèles hybrides

- Ils sont basés sur l'utilisation combinée de plusieurs des modèles précédents
  - Typiquement la combinaison du modèle Message Passing avec MPI et du modèle des threads ou du modèle mémoire partagée avec OpenMP.

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

Des langages émergents

- Architectures à mémoire partagée plus simples à programmer
- Mais beaucoup d 'architectures sont à mémoire distribuée pour des raisons de coût et de performance

Plusieurs solutions s'appuyant sur une couche software :

- Des systèmes virtuellement partagées
- Des langages de programmation basés sur le modèle PGAS: langages à espace d'adressage global partitionné

  Accès aux données via des références locales et distantes

  Des structures de données distribuées avec adressage global, prise en compte de l'affinité mémoire

  Ex : UPC, Coarray Fortran, X10,Titanium, Chapel

Conception d’un programme parallèle

Conception d'un programme parallèle

- Loi d'Amdhal

  Speedup max ?

Soit S la fraction séquentielle du programme,
P la fraction qui peut être parallélisée P=1-S
N le nombre de processeurs

Soit t le temps d'exécution en séquentiel (temps du prog. // sur 1 proc)
En théorie, le temps minimum en parallèle est

$$(S + (1-S)/N)*t$$

$$\text{Speedup maximum} = \frac{1}{\frac{1-S}{N} + S}$$

Speedup linéaire

Coût d’un programme parallèle

- En pratique les algorithmes des versions séquentielle et parallèle peuvent être différents
- L’accélération parallèle peut être calculée à partir
  - du temps d’exécution du prog. // sur 1 proc.
  - d’un temps d’exécution uniprocesseur de référence

- De plus, le calcul du coût théorique (par analyse de complexité) peut difficilement prendre en compte les spécificités des architectures

- D’où parfois, en pratique, des accélérations surlinéaires :
  - Sur des architectures hybrides distribuée/partagée lorsque la taille du problème excède la capacité de la mémoire locale
  - Lorsque l’algorithme parallèle est plus efficace

Coût d'un programme parallèle

- Influence du coût de communication sur l'accélération

Supposons la fraction séquentielle S=0

alors $T_{séq} = T_{calcul}$

$$T_{par} = \frac{T_{calcul}}{N} + T_{comm}$$

$$\frac{T_{séq}}{T_{par}} = \frac{T_{calcul}}{\frac{T_{calcul}}{N} + T_{comm}} = \frac{N}{1 + N \frac{T_{comm}}{T_{calcul}}}$$

$\frac{T_{comm}}{T_{calcul}}$ **Est un paramètre important**

Conception d’un programme parallèle

Vérifier d’abord si un code // n’est pas déjà disponible pour ce problème, dans une bibliothèque // :

Sinon :

Bien analyser le problème à résoudre :

Identifier les parties coûteuses en temps de calcul

Déterminer si le problème peut réellement être parallélisé
Identifier les opérations qui interdisent la parallélisation (par ex dépendance de données)

Si on part d’un programme séquentiel, rechercher des algorithmes éventuellement mieux adaptés à la parallélisation, événtuellement restructurer le code

Conception d'un programme parallèle

Une des premières étapes consiste à décomposer le problème en parties approximativement de même taille qui seront affectées sur différentes tâches

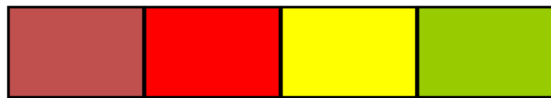
2 méthodes pour décomposer et affecter à ≠ tâches // :
Décomposition de domaine : les données associées au pb sont décomposées
Décomposition fonctionnelle : le problème est décomposé en fonction du travail à effectuer, chaque tâche exécute une portion de la totalité du travail.
Le choix de la méthode est dicté par l'application.

Si la tâche est petite on parle de grain fin, quand la taille augmente on parle de grain moyen, puis de gros grain

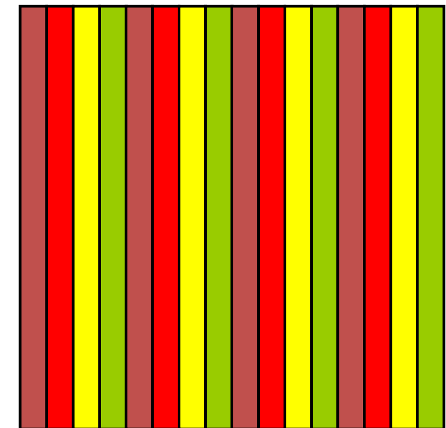
Décomposition de domaines

Ce modèle sera d'autant plus efficace que le rapport :
(taille des sous-domaine/taille des frontières) est grand
(calculs / communications)

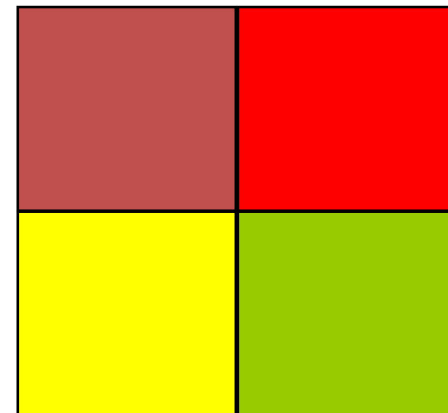
1D
block



1D
cyclic



2D
Bloc,*



Décomposition fonctionnelle

- Ce modèle est adaptée aux problèmes qui peuvent se découper naturellement en # tâches

Le problème est vue comme un ensemble d’instructions, on identifie des groupes d’instructions indépendants que l’on affecte à différentes tâches avec les données associées

C’est le placement des calculs qui détermine le placement des données

Communications

2 cas de figure :

- Les tâches travaillent indépendamment les unes des autres, avec très peu de communications entre tâches : le problème est dit «embarrassingly parallel »
- Les tâches ont des données à échanger,
  - ➢ un surcoût si les comm. ne peuvent être masquées par les calculs
  - ➢ des synchronisations entre tâches

Temps de communication de B octets = $\boldsymbol{\alpha + B\,\beta}$

Où α temps de latence et β temps de transfert d'1 octet

Trop de petites communications ⇨ domination des temps de latence (en général $\boldsymbol{\alpha \gg \beta}$)

Concaténer les petits messages pour envoyer de plus gros paquets

Définir les structures de communication et les algos appropriés

Communications synchrones/asynchrones

- Les communications synchrones requièrent un « handshake » entre tâches qui échangent les données. Elles sont dites bloquantes.
- Les comm. asynchrones sont non bloquantes. Une tâche ne sera pas bloquée sur l'instruction de comm, elle peut continuer à traiter les instructions suivantes pendant que la comm. s'effectue.

Communications synchrones/asynchrones

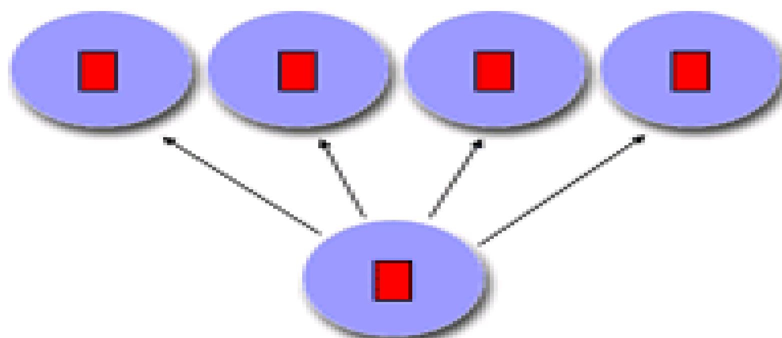
- Point à point

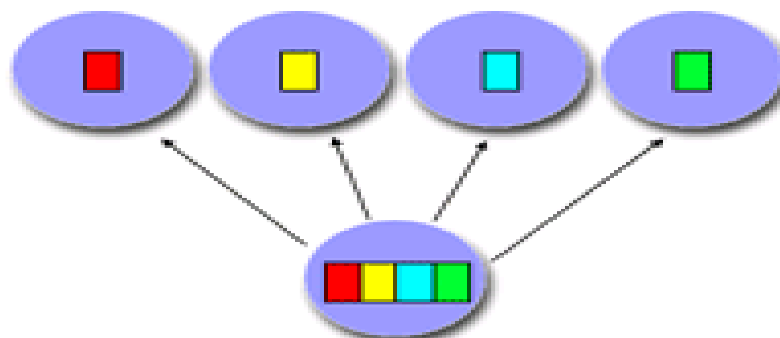
  une tâche agit en tant que émetteur/producteur de données, et l'autre en tant que récepteur/consommateur.

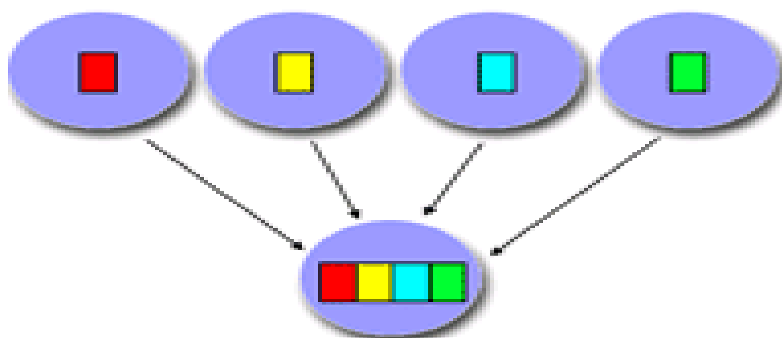
- Collective

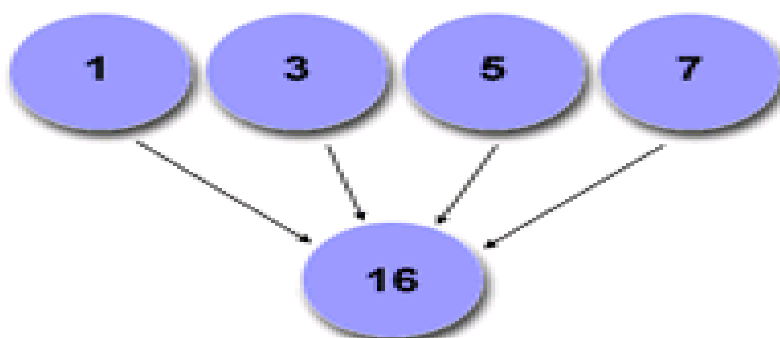
  met en jeu le partage de données entre plusieurs tâches (plus que 2), qui sont souvent spécifiées comme étant membres d'un groupe. Des exemples classiques : broadcast, gather, reduction, scatter

Communications collectives









Granularité et load balancing

- Notion de load balancing : consiste à distribuer le travail de façon à ce que toutes les tâches soient occupées un maximum du temps.

➢ Grain fin

Peu de calcul entre les communications
Facilite le load balancing
Risque de surcoût dû aux comm. et aux synchros

➢ Gros grain

Beaucoup de calculs entre 2 comm.
Plus facile d'améliorer les performances
Plus difficile d'améliorer le load balancing

Programme // versus Programme séquentiel

- + de CPUs

- Souvent + de mémoire pour la duplication des données

- Sur des exécutions de courte durée, la version // peut être plus lente que la version séquentielle (surcoût associé à la mise en œuvre de l'environnement //)

Scalabilité

- Généralement, des limites intrinsèques à l’algorithme
- Le hardware joue un rôle important (débit bus mémoire-cpu sur SMP, débit réseau de comm., taille et architecture mémoire, vitesse d’horloge du proc., …)
- Les systèmes et les hardware peuvent jouer sur la scalabilité et donc sur la portabilité

Conclusion

- Vers une augmentation du nombre de coeurs plutôt que de la fréquence d'horloge
- Vers des coûts de transferts de plus en + élevés par rapport au calcul
- Programmation de plus en plus complexe :
  - Nécessité d'une bonne connaissance des architectures
  - parallélisation obligatoire pour un gain de performance, en tenant compte des flots de données
  - Nécessité d'exploiter le parallélisme aux différents niveaux de hardware
  - Programmation sur des architectures hybrides avec éventuellement différents types de processeurs, et différents modèles de programmation

Conclusion

- Exploitation des performances des processeurs graphiques ou des manycores: produire un code extrêmement parallélisé, sinon on atteint que quelques % de la puissance disponible.

- Des architectures qui évoluent très vite, il faut s'adapter continuellement au niveau des softs
- Des efforts au niveau recherche pour fournir des outils pour faciliter le travail du programmeur

Références

Un certain nombre d'éléments de ces slides sont issus des présentations ou des documentations présentes sur les sites suivant :

- http://www.realworldtech.com
- http://calcul.math.cnrs.fr
- http://www.ci-ra.org séminaire « trends in computer architecture » W. Jalby
- https://computing.llnl.gov/tutorials/parallel_comp
- http://www.citutor.org ncsa (national center for supercomputing)
- http://www.irisa.fr/ORAP (organisation associative pour le parallélisme)
- http://www.idris.fr séminaires de l'IDRIS
- www.intel.com
- www.nvidia.fr