Généralités sur le parallélisme

Laurence Viry et Violaine Louvet

Ecole d'Automne « Informatique Scientifique pour le Calcul »

3 Décembre 2008

Sommaire

Sommaire

Qu'est ce que le calcul Parallèle

- Plusieurs calculs simultanés
  - sur un ou plusieurs processeurs
  - sur un ou plusieurs calculateurs

Qu'est ce que le calcul Parallèle

- Plusieurs calculs simultanés
  - sur un ou plusieurs processeurs
  - sur un ou plusieurs calculateurs
- Quand ?
  - Quand les limites des performances et de la taille des problèmes traités sont atteintes sur un seul processeur

Qu'est ce que le calcul Parallèle

- Plusieurs calculs simultanés
  - sur un ou plusieurs processeurs
  - sur un ou plusieurs calculateurs
- Quand ?
  - Quand les limites des performances et de la taille des problèmes traités sont atteintes sur un seul processeur
- comment ?
  Par décomposition :

Qu'est ce que le calcul Parallèle

- Plusieurs calculs simultanés
  - sur un ou plusieurs processeurs
  - sur un ou plusieurs calculateurs
- Quand ?
  - Quand les limites des performances et de la taille des problèmes traités sont atteintes sur un seul processeur
- comment ?
  Par décomposition :
  - des données en particulier pour traiter de plus gros volumes de données : **"data parallelism"**

Qu'est ce que le calcul Parallèle

- Plusieurs calculs simultanés
  - sur un ou plusieurs processeurs
  - sur un ou plusieurs calculateurs
- Quand ?
  - Quand les limites des performances et de la taille des problèmes traités sont atteintes sur un seul processeur
- comment ?
  Par décomposition :
  - des données en particulier pour traiter de plus gros volumes de données : **"data parallelism"**
  - des tâches indépendantes : **"task parallelism"**

Sommaire

Motivations

- La vitesse des processeurs ne continuera pas augmenter (x2 tous les 18mois)

Motivations

- La vitesse des processeurs ne continuera pas augmenter (x2 tous les 18mois)
- Le calcul parallèle peut être :

Motivations

- La vitesse des processeurs ne continuera pas augmenter (x2 tous les 18mois)
- Le calcul parallèle peut être :
  - la seule solution pour traiter certains gros calculs en un temps donné
    - Teraflops et Petabytes challenges
    - Kilo-transaction/seconde pour des réseaux ATM, digital multimédia, visualisation, data-mining...

Motivations

- La vitesse des processeurs ne continuera pas augmenter (x2 tous les 18mois)
- Le calcul parallèle peut être :
  - la seule solution pour traiter certains gros calculs en un temps donné
    - Teraflops et Petabytes challenges
    - Kilo-transaction/seconde pour des réseaux ATM, digital multimédia, visualisation, data-mining...
  - la solution la plus simple ou la moins onéreuse pour traiter certains calculs en un temps donné

Motivations

- La vitesse des processeurs ne continuera pas augmenter (x2 tous les 18mois)
- Le calcul parallèle peut être :
  - la seule solution pour traiter certains gros calculs en un temps donné
    - Teraflops et Petabytes challenges
    - Kilo-transaction/seconde pour des réseaux ATM, digital multimédia, visualisation, data-mining...
  - la solution la plus simple ou la moins onéreuse pour traiter certains calculs en un temps donné
- Le calcul parallèle apporte une solution pour des problèmes nécessitant une haute tolérance aux fautes

Motivations

- La vitesse des processeurs ne continuera pas augmenter (x2 tous les 18mois)
- Le calcul parallèle peut être :
  - la seule solution pour traiter certains gros calculs en un temps donné
    - Teraflops et Petabytes challenges
    - Kilo-transaction/seconde pour des réseaux ATM, digital multimédia, visualisation, data-mining...
  - la solution la plus simple ou la moins onéreuse pour traiter certains calculs en un temps donné
- Le calcul parallèle apporte une solution pour des problèmes nécessitant une haute tolérance aux fautes
- L'univers est hautement parallèle

Motivations

- La vitesse des processeurs ne continuera pas augmenter (x2 tous les 18mois)
- Le calcul parallèle peut être :
  - la seule solution pour traiter certains gros calculs en un temps donné
    - Teraflops et Petabytes challenges
    - Kilo-transaction/seconde pour des réseaux ATM, digital multimédia, visualisation, data-mining...
  - la solution la plus simple ou la moins onéreuse pour traiter certains calculs en un temps donné
- Le calcul parallèle apporte une solution pour des problèmes nécessitant une haute tolérance aux fautes
- L'univers est hautement parallèle
- ...

Parallèlisme et calculateur

- Operating system
  - Exécution en arrière plan
  - Cron et batch

Parallèlisme et calculateur

- Operating system
  - Exécution en arrière plan
  - Cron et batch
- Arithmétique
  - Unité d exécution multiples
  - Instruction multiply and add
  - Arithmétique super-scalaire
  - Pipelining

Parallèlisme et calculateur

- Operating system
  - Exécution en arrière plan
  - Cron et batch
- Arithmétique
  - Unité d exécution multiples
  - Instruction multiply and add
  - Arithmétique super-scalaire
  - Pipelining
- La mémoire
  - Mémoire interlassée
  - Mémoire hiérarchique
  - Plusieurs ports d'accès la mémoire
  - Utilisation des caches

Parallèlisme et calculateur

- Operating system
  - Exécution en arrière plan
  - Cron et batch
- Arithmétique
  - Unité d exécution multiples
  - Instruction multiply and add
  - Arithmétique super-scalaire
  - Pipelining
- La mémoire
  - Mémoire interlassée
  - Mémoire hiérarchique
  - Plusieurs ports d'accès la mémoire
  - Utilisation des caches
- Les disques
  - Disques RAID
  - Striping

Sommaire

Taxonomie des architectures

- Une bonne connaissance de la classification des architectures parallèles permet une bonne approche dans la recherche des solutions aux problèmes de calculs intensifs

Taxonomie des architectures

- Une bonne connaissance de la classification des architectures parallèles permet une bonne approche dans la recherche des solutions aux problèmes de calculs intensifs
- Une classification sur des critères hardware induit une classification en terme de software

Sommaire

Classification des architectures parallèlles

Classification de Flynn(1966) basée sur le type d'organisation du flux de données et du flux d'instructions.

- Parallélisme de données
- Parallélisme d'instructions

$$\left\{ \begin{array}{c} S \\ M \end{array} \right\} I \left\{ \begin{array}{c} S \\ M \end{array} \right\} D$$

SI **Single Instruction**

MI **Multiple Instruction**

SD **Single Data**

MD **Multiple Data**

Classification des architectures parallèlles(suite)

SISD Single Instruction, une seule mémoire
Un ordinateur séquentiel qui n'exploite aucun parallélisme, Architecture de Von Neumann

SIMD Single Instruction, Multiple données
Tableau de processeurs où chaque processeur exéxute la même instruction sur ses propres données locales

MISD Multiple instructions, une seule mémoire
Une même donnée est traitée par plusieurs processeurs en parallèle.

MIMD Instructions multiples, Multiple données
Plusieurs processeurs traitent des données différentes, chaque processeur possède une mémoire distincte.
L'architecture parallèle la plus utilisée.

Sommaire

Organisation de la mémoire

- Mémoire distribuée
- Mémoire partagée
- Mémoire partagée distribuée

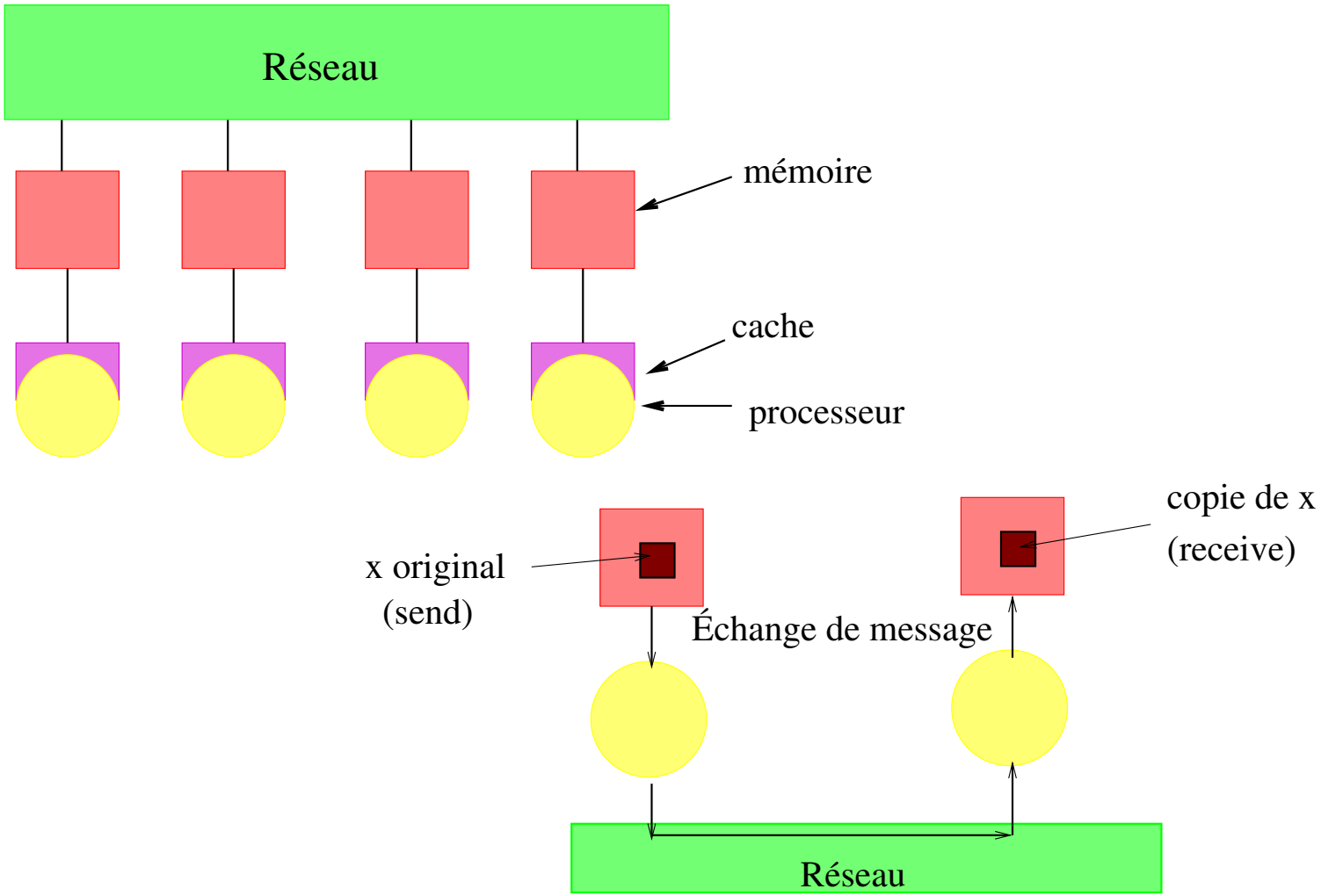
Sommaire

Système à mémoire distribuée

Un espace mémoire est associé à chaque processeur

- L'accès à la mémoire du processeur voisin se fait par échange de messages à travers le réseau entre les processeur
- Les algorithmes utilisés devront minimiser les communications

Système à mémoire distribuée



Système à mémoire distribuée

Avantages :

- La mémoire est scalable avec le nombre de processeurs
- Chaque processeur a un accès rapide à sa propre mémoire
- Coût réduit et facile à construire

Inconvénients :

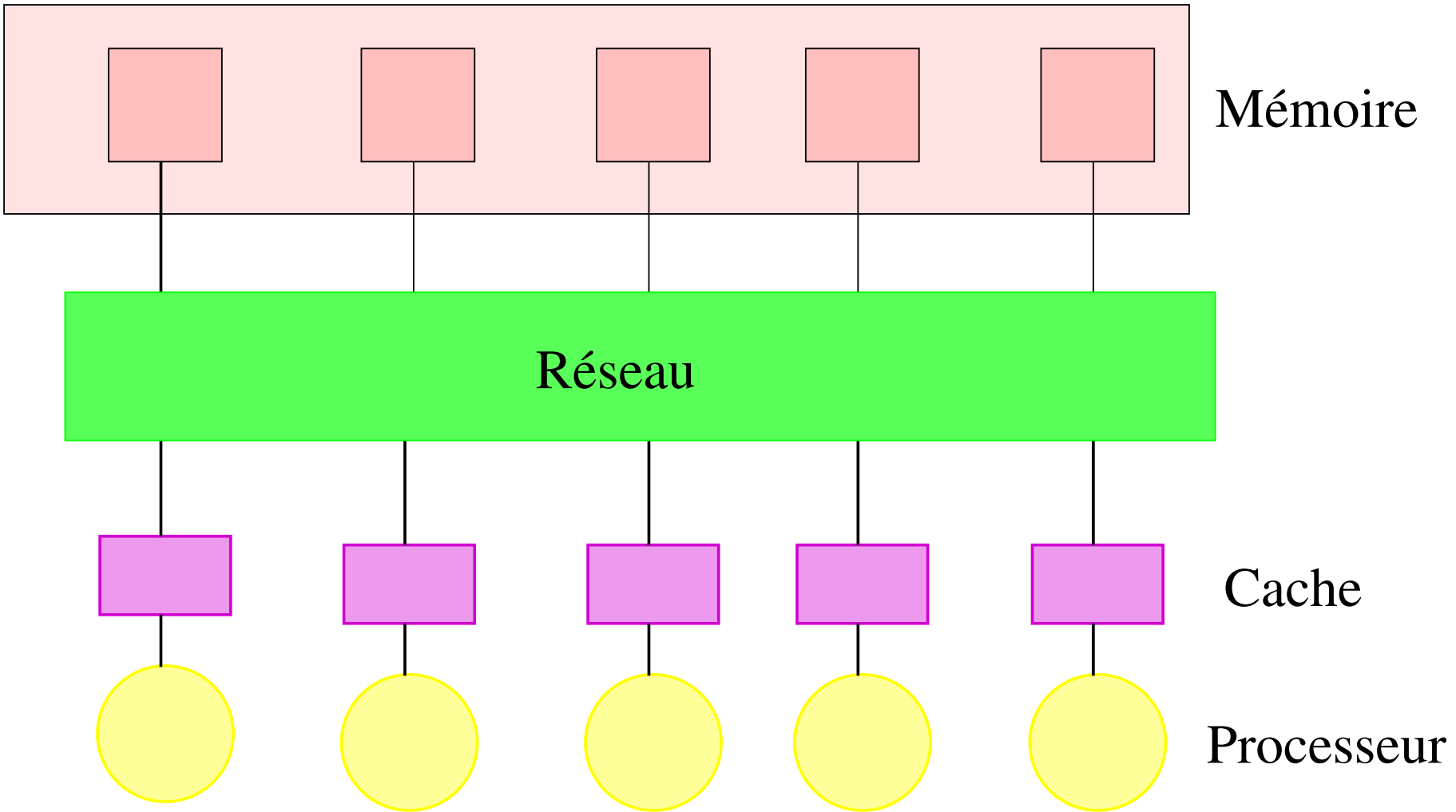
- Le programmeur devra gérer lui-même les communications
  - Risque d'erreurs
  - complexité des algorithmes
  - Source de perte de performances
- Les performances seront parfois au prix d'un réseau d'interconnexion couteux

Sommaire

Système à mémoire partagée

- Un espace mémoire global visible par tous les processeurs
- Les processeurs auront leur propre mémoire locale(cache,...) dans laquelle sera copié une partie de la mémoire globale
- La cohérence entre ces mémoires locales devra être gérée par le hardware, le software et parfois l'utilisateur

Système à mémoire partagée



Temps d'accès à la mémoire partagée

Deux classes d'architectures parallèles à mémoire partagée basées sur le temps d'accès à la mémoire

UMA(Uniform Memory ACCESS) :

- Machines SMP (Symmetric Multi processors), processeurs identiques
- Temps d'accès à la mémoire identique depuis chaque processeur
- Certains systèmes sont **CC-UMA (cache coherent UMA)** : si un processeur met à jour une variable dans la mémoire globale,tous les processeurs connaissent cette mise à jour
- exemples : dual /quad core x86 (PowerPC,SPARC,...)...XS

Temps d'accès à la mémoire partagée(suite)

NUMA(Non-Uniform Memory Access) :

- Construit à partir d'un lien physique entre deux ou plusieurs machines SMP
- Une noeud SMP accède directement à la mémoire d'un autre noeud SMP sans échange de message
- Le temps d'accès à la mémoire globale n'est pas uniforme, l'accès à la émoire via le lien physique est plus long
- Système de cohérence des caches : **CC-NUMA**
- exemples : OriginXXX,...

Système à mémoire partagée

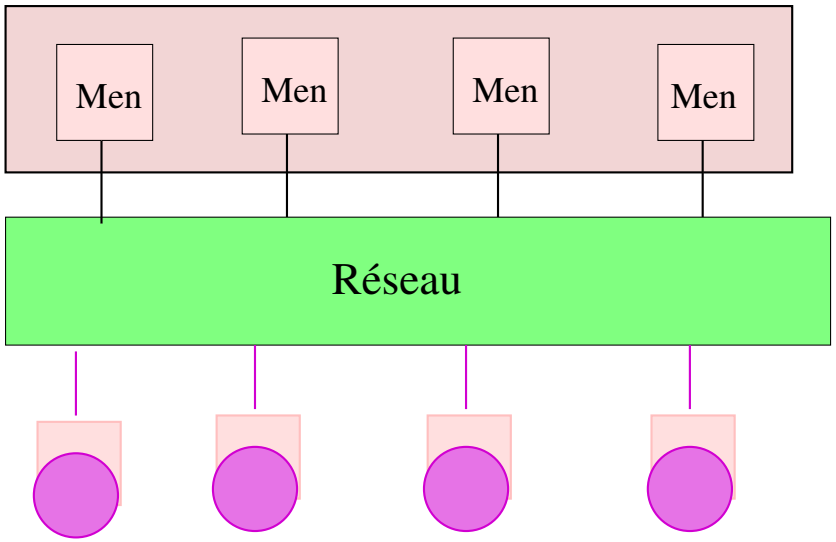
Avantages :

- Programmation simple
- Le partage des données entre les tâches est rapide

Inconvénients :

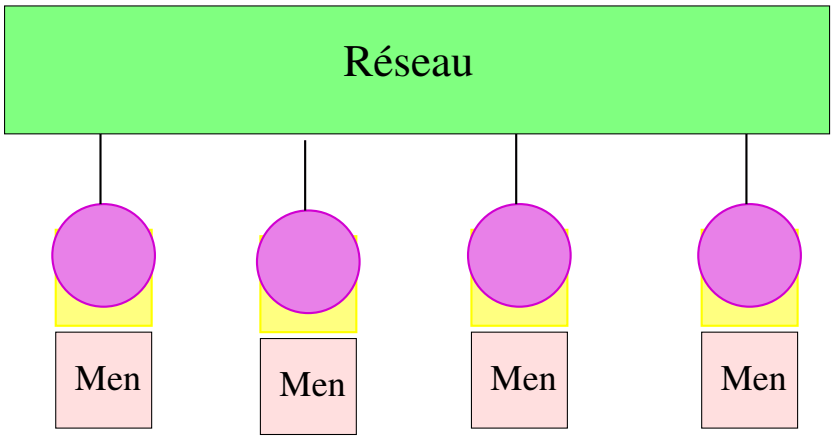
- Ne favorise pas la scalabilité des accès mémoire
- Le programmeur est responsable de la validité des synchronisations

Système à mémoire partagée vs. Système à mémoire distribuée



Mémoire partagée

Processeur + Cache



Mémoire distribuée

Mémoire partagée sur Mémoire distribuée

- Comme on le verra plus loin, la parallélisation d'une application pour une architecture à mémoire partagée est plus simple
- Cepandant, beaucoup d'architecture sont à mémoire distribuée pour des raisons de coût

Pour bénéficier de ces deux avantages, en utilisant la couche software, des investigations sont faites :

- des systèmes à mémoire virtuellement partagée
- ou des langages à espace global adressable (Unified Parallel C, Co-array Fortran,...)

Performances sur des système à mémoire virtuellement partagée

- Importance du temps de communication sur les machine à mémoire distribuée
- L'accès à la mémoire est non-uniforme, il dépend de la localisation de la donnée, sur la mémoire locale du noeud qui la traîte ou sur la mémoire d'un autre noeud
- Pour obtenir des performances, on devra tenir compte du caractère physiquemen distribué de la mémoire en utilant un modèle adapté $\Longrightarrow$ algorithm plus compliqué

Sommaire

Multi-Core

CPU : Chip qui effectue les opérations de base d'un système. Les connexions au bus système se fait par un socket

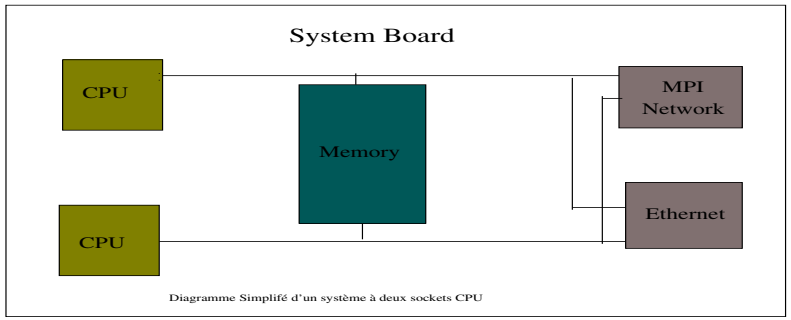


Diagramme Simplifié d'un système à deux sockets CPU

CORE : Une unité de calcul contenu dans un CPU. Le core a ses propres registres et son cache L1. Plusieurs cores dans un CPU sont attachés à un même socket

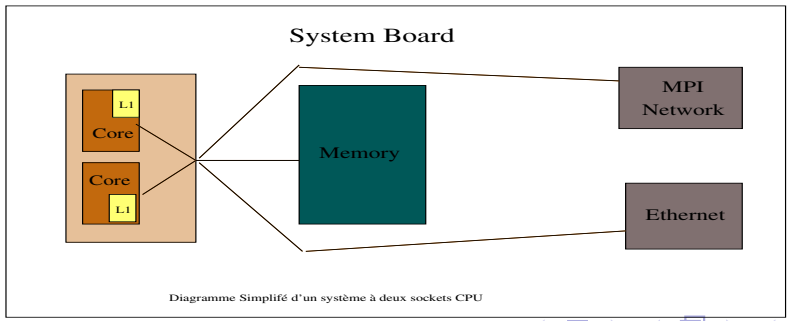
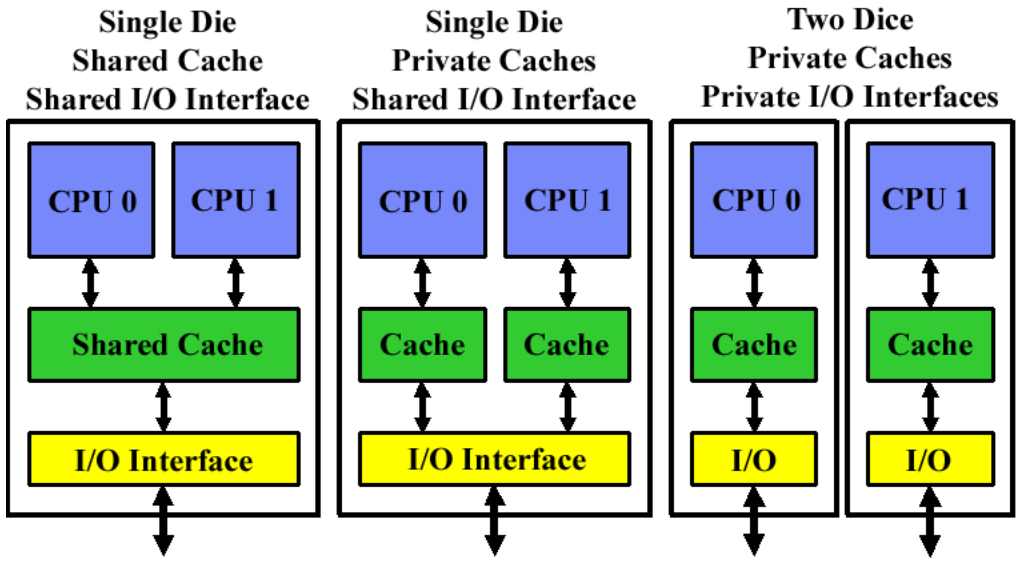


Diagramme Simplifié d'un système à deux sockets CPU

A first simple view

Source: © Realworldtech



Organisation des caches

- Jamais de partage du cache L1, il est sur un chemin critique
- Partage des caches L2 et/ou L3
  - Meilleure communication entre les cores
  - Meilleure utilisation des caches(?)
  - Facilite la migration des threads entre les cores
- Pas de partage
  - Pas de contention : espace et bande passante mémoire
  - Communication et migration sont moins performantes(tout passe par la mémoire)
- Dans tous les cas
  - Partage les sockets : contention accès mémoire, partage de la bande passante
  - Pas de mécanismes de synchronisation rapides

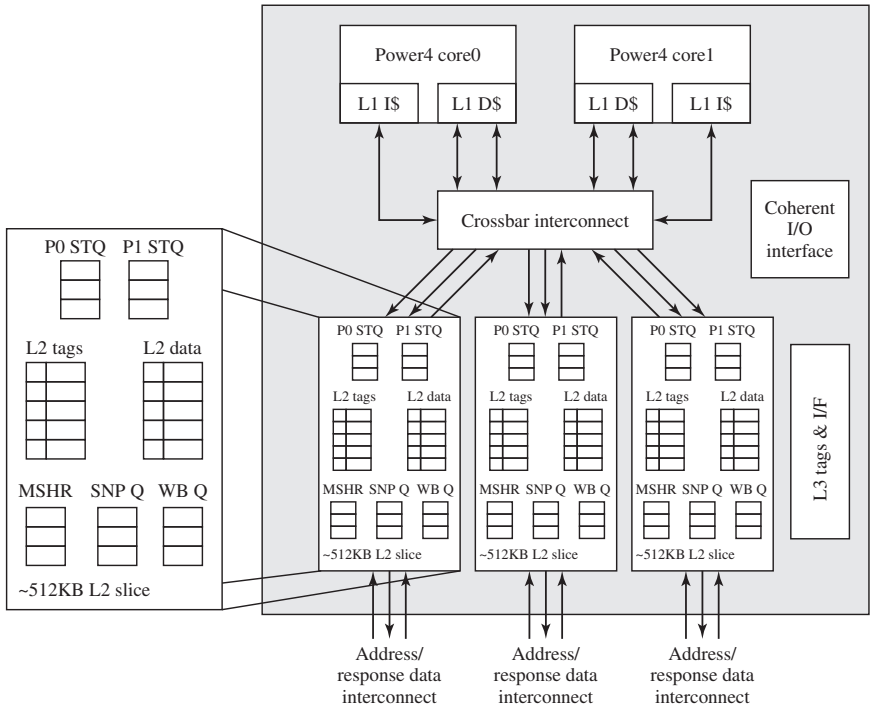
SMP et Multi-Core

Ce qui différencie un Multi-Core d'une machine SMP

- Coût plus faible : 1 socket pour le multi-core, autant de sockets que de CPU pour une machine SMP
- organisation des caches (partagé ou pas)
- fréquence d'horloge plus faible qu'un mono-core CPU : minimise la puissance absorbée et dissipée

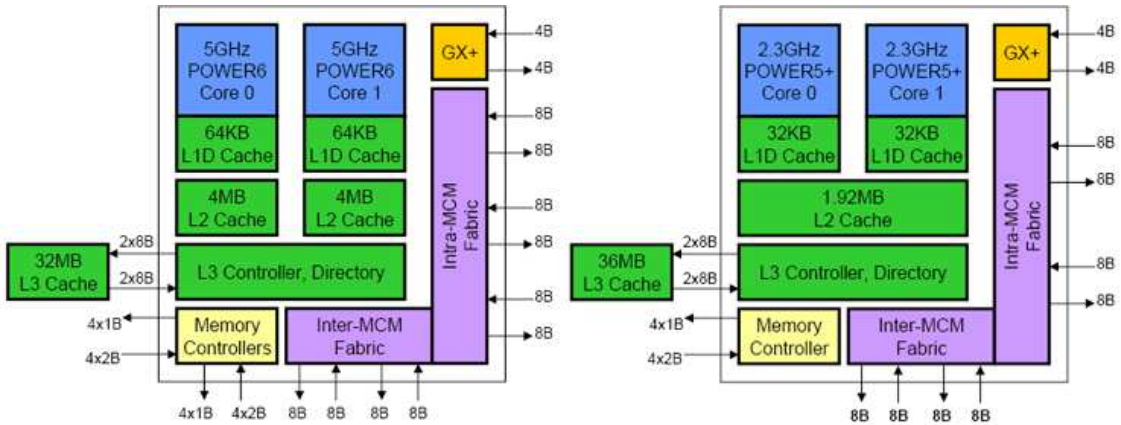
Exemple de dual-core : IBM Power4

IBM Power4: Example bicore



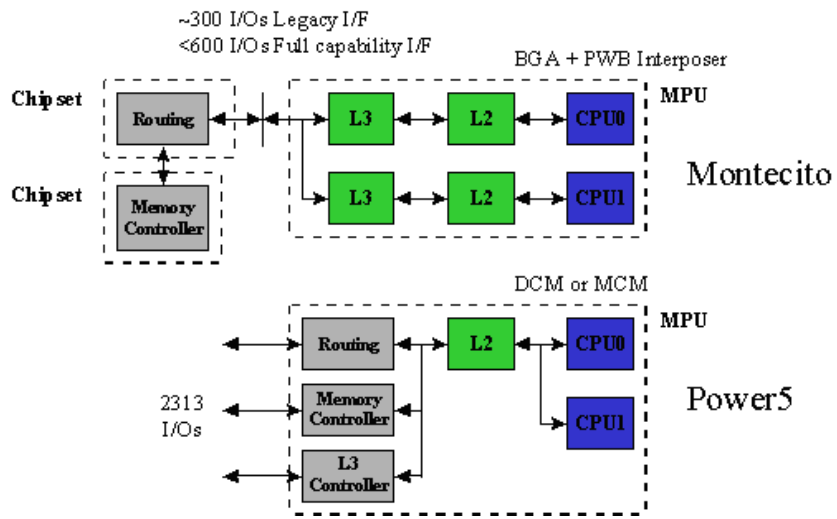
Power5+/Power6 Organization

Source: © Realworldtech

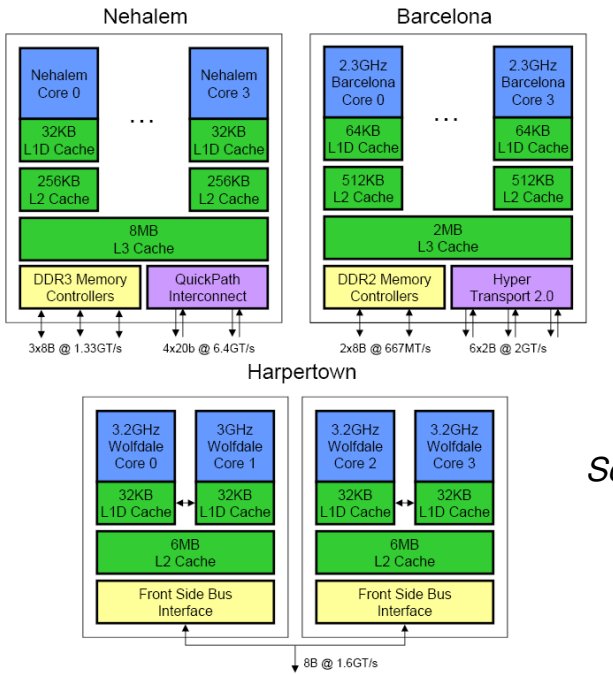


Power5/Montecito Organization

Source: © Realworldtech



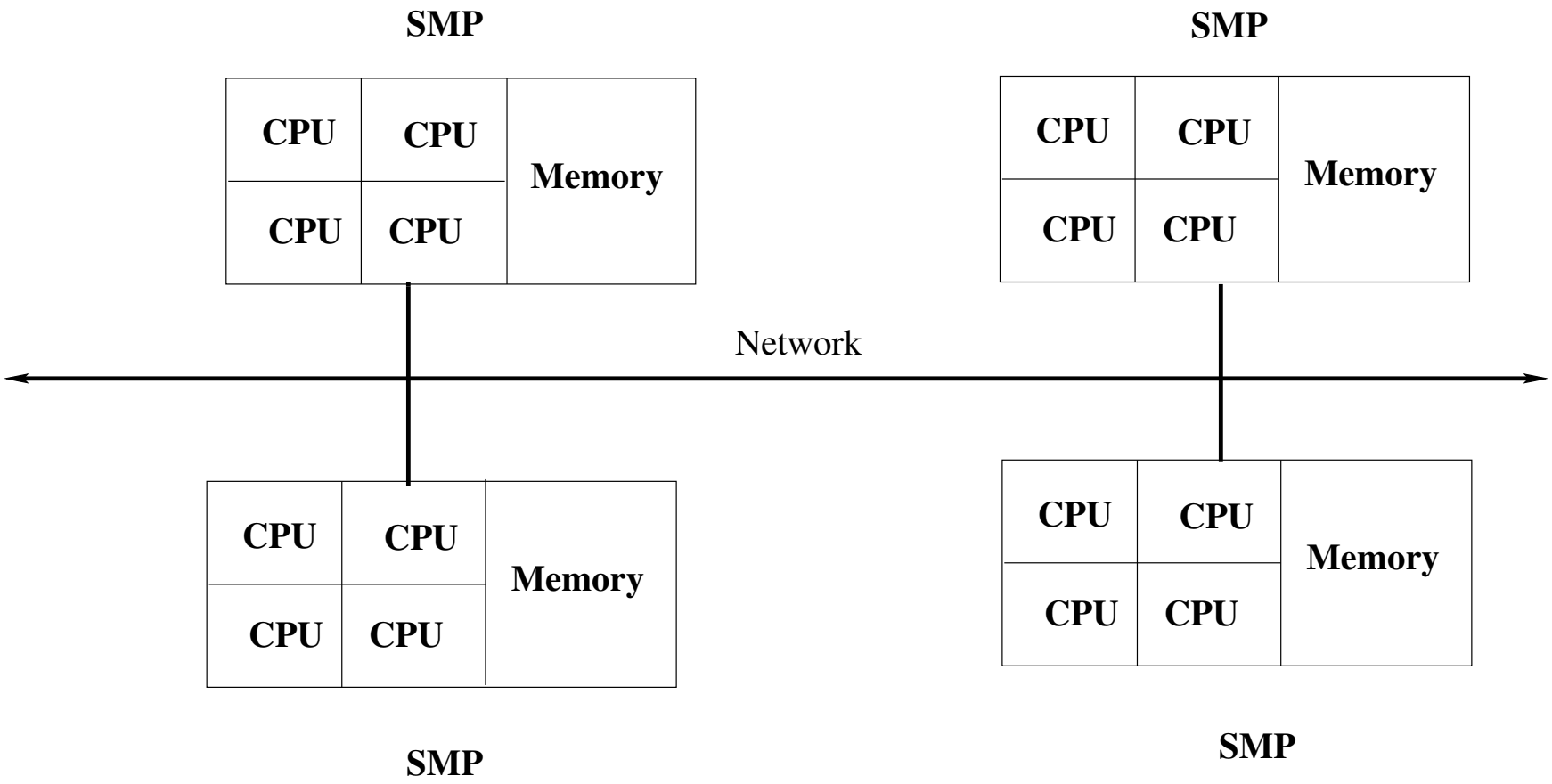
X86 Organizations



Source: © Realworldtech

Sommaire

Architecture hybride



CPU: Single core or Multi core

Système hybride

- Les composants “mémoire partagée” sont des noeuds CC (Cache coherent) SMP avec un accès à la mémoire uniforme (CC-UMA) ou non-uniforme(CC-NUMA)
- CC : si un processeur modifie une variable de la mémoire partagée, tous les autres processeurs connaissent la modification
- Le composant “mémoire distribuée” est un cluster de noeuds de multiple SMP
- Les noeuds SMP peuvent accéder à leur propre mémoire mais ne peuvent accéder à la mémoire des autres noeuds SMP
- Un réseau de communication est nécessaire entre les noeuds SMP

Sommaire

Aperų dy sustème BlueGene

- Massivement parallèle, architecture à mémoire distribuée
- Haute densité physique
  - Atteint à partir de processeur de faible puissance
- Interconnects très performants
  - Plusieurs réseaux dédiés à des tâches différentes
- O/S très allégé
  - destiné au calcul et à rien d'autre
- Software Standard (Fortran,C,C++,MPI...

Architecture de la BlueGene

- System-on-a-chip design, based on Power440 core
  - Permet certaines customisations sans avoir le coût du développement
  - Customisation de la mémoire,de l'interface réseau et des "floating point"
- ON-Chipe Node contient 2 core Power 440 avec les caches associés
  - Fréquence d'horloge faible pour une faible dissipation

Blue Gene Chip

Blue Gene chip

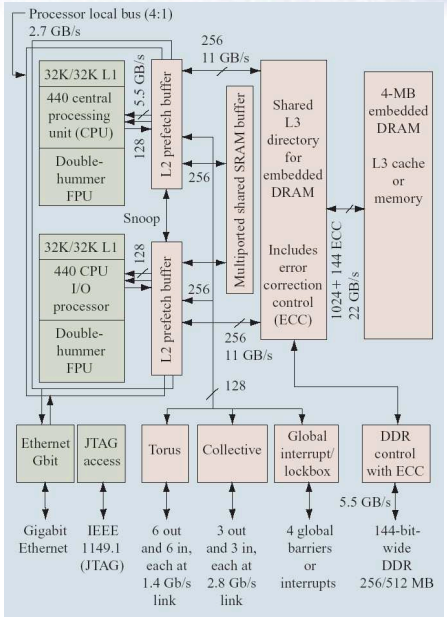


Diagram reproduced from IBM

Pourquoi une faible puissance

- C'est la clé pour obtenir une faible taille
- Le pouvoir de dissipation d'un Chip dépend de la fréquence d'horloge
- Beaucoup d'applications HPC sont "memory bound" et non "CPU bound
  - la puissance du système de mémoire n'est pas sous-estimée
- Un rapport flops/watt très élevé
- Un cluster p690 avec les mêmes performances Linpack occuperaient environ 35 racks, un p575 environ 5

Blue Gene Système

- Deux cores constituent un “compute card”
- 16 “compute card” constituent “node board”
  - Chaque “compute card” a 1 ou 4 I/O processeurs
- 16 “node board” forment un “midplane” (512 compute cards)
- 2 “midplane” dans un rack (1024 compute cards)
- jusqu’à 64 racks

Blue Gene system

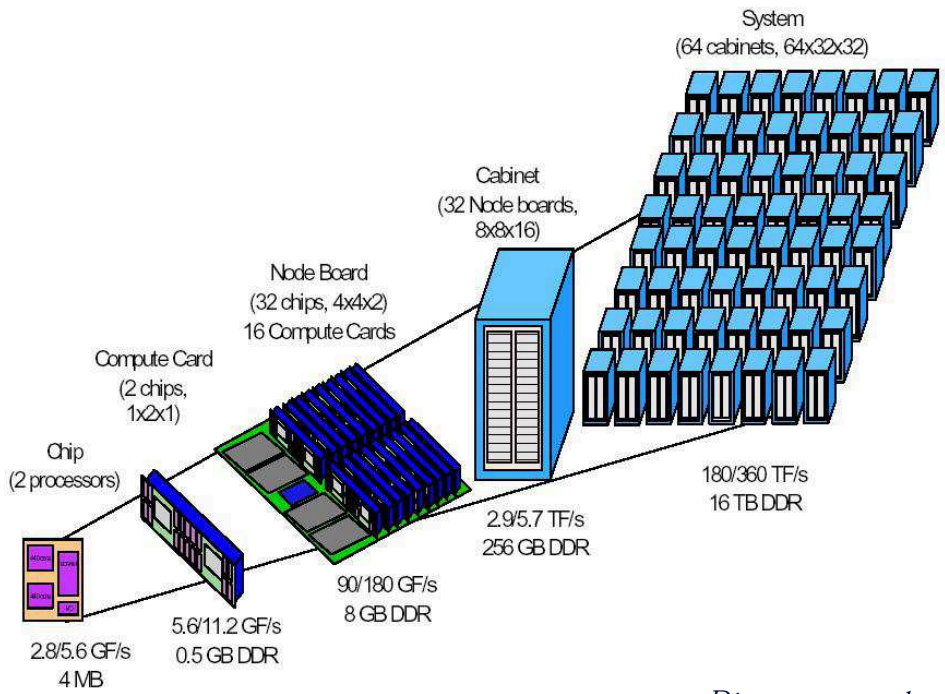


Diagram reproduced from IBM

Sommaire

GPU

On attend en priorité :

d'un CPU qu'il exécute une tâche (éventuellement plusieurs) le plus rapidement possible
$\Longrightarrow$ complexification des unités de contrôle et une augmentation de la mémoire cache embarqué

d'un GPU : de pouvoir traiter une tâche sur un maximum de données, dans un temps réduit
$\Longrightarrow$ la multiplication des unités de traitement

Ce processeur GPU s'attaque principalement à des problèmes qui peuvent être résolus avec un modèle "data-parallel", un même programme est exécuté sur un grand nombre de données différentes

- intensif en calculs arithmétiques, les accès mémoire peuvent être cachés par le calcul
- le même programme exécuté sur des données différentes $\Longrightarrow$ peu de flot de contrôle

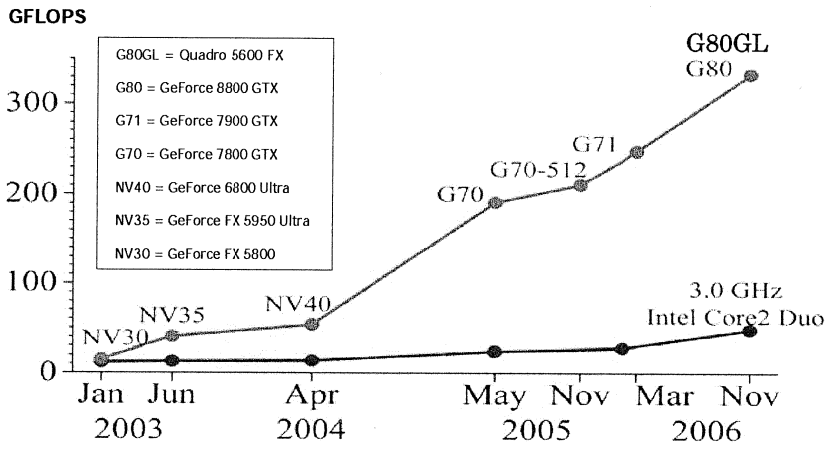
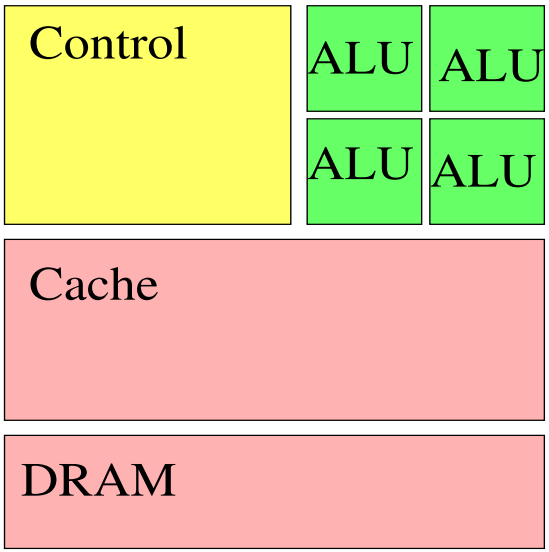


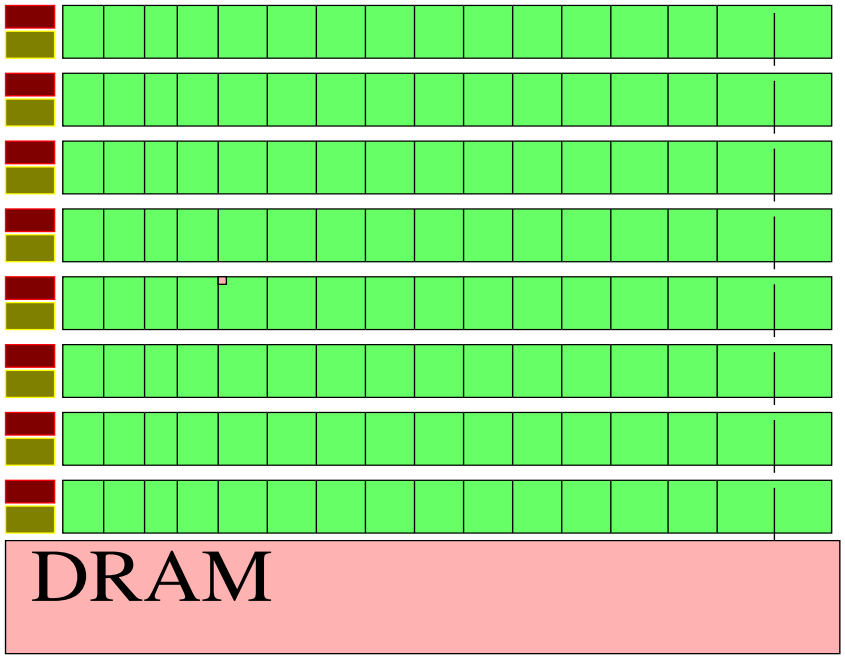
Figure 1-1. Floating-Point Operations per Second for the CPU and GPU

CPU vs GPU

Depuis quelques années, les processeurs graphiques (GPU) ont rapidement évolués, avec les multi-cores et leur grande bande passante mémoire, ils peuvent résoudre de façon performante les applications graphiques et non-graphiques



CPU

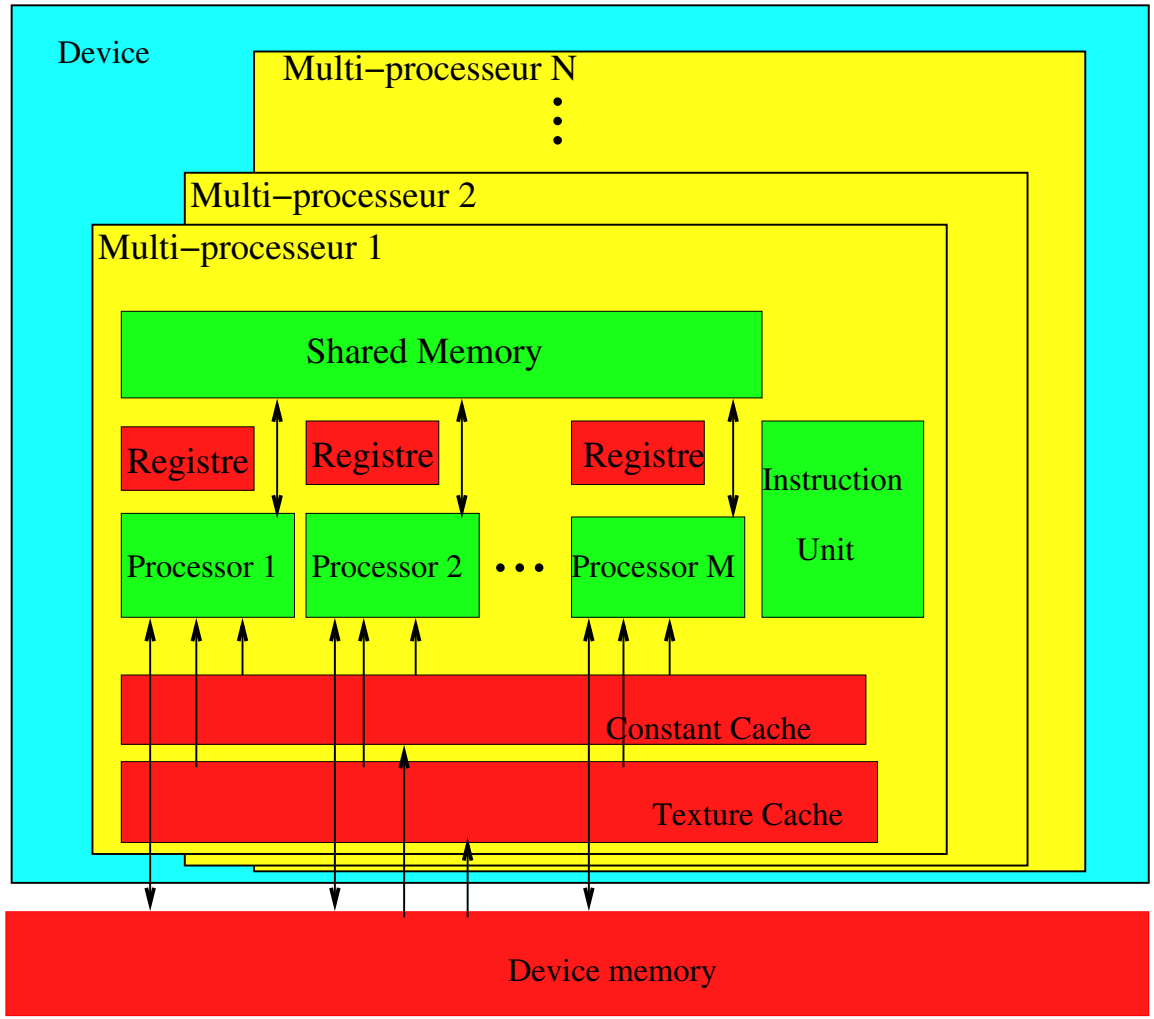


GPU

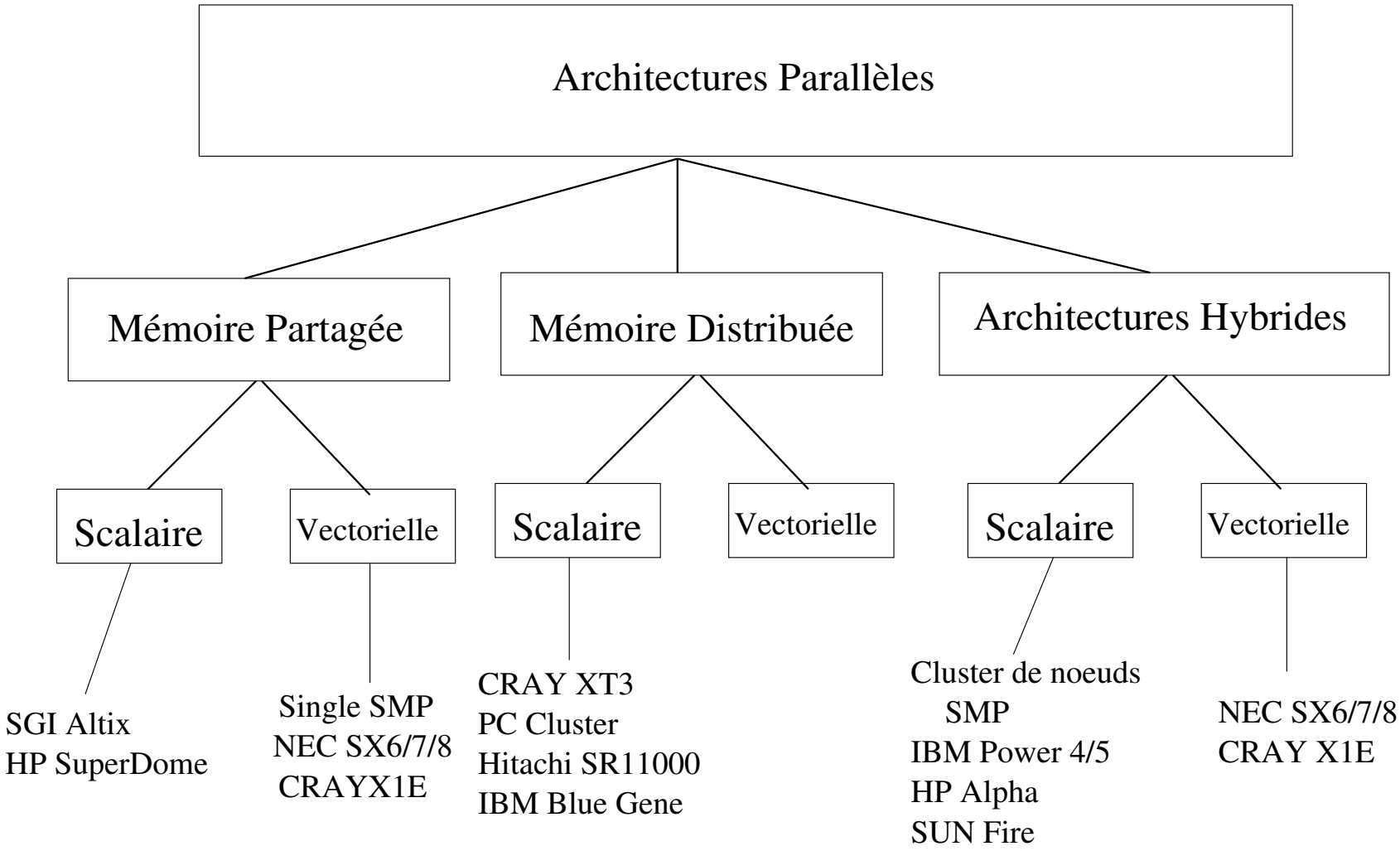
Architecture GPU

- Ensemble de multi-processeurs(mémoire partagée)
- Chaque multi-processeurs à un architecture SIMD
- À chaque cycle, chaque processeur d'un multi-processeur exécute la même instruction sur des données différentes
- Chaque multi-processeur à des mémoires on-chip du type :
  - registres
  - "data cache" parallèle (shared memory) partagés par tous les processeurs
  - Un "constant cache" en lecture uniquement, partagé par tous les processeurs $\Longrightarrow$ accélère la lecture
  - un "texture cache" partagés par tous les processeurs
- La mémoire locale et globale sont implémentées en RW (sans cache)

Architecture GPU



Taxonomie des architectures Parallèles



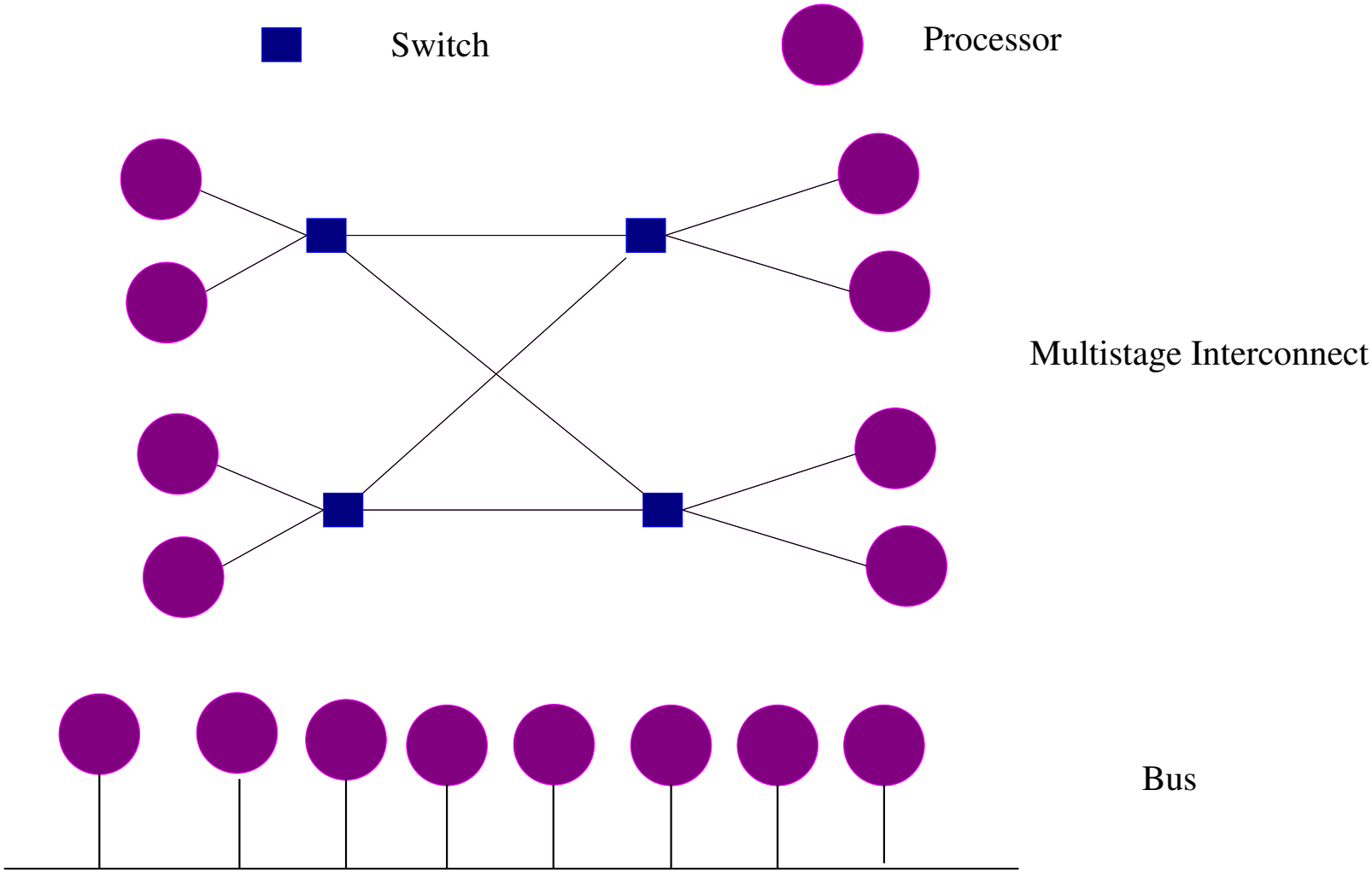
Sommaire

Réseau de communications

Il y a de nombreux moyens d'interconnexion entre plusieurs processeurs mais pour l'utilisateur, les différences sont mineurs. On considère principalement deux classes :

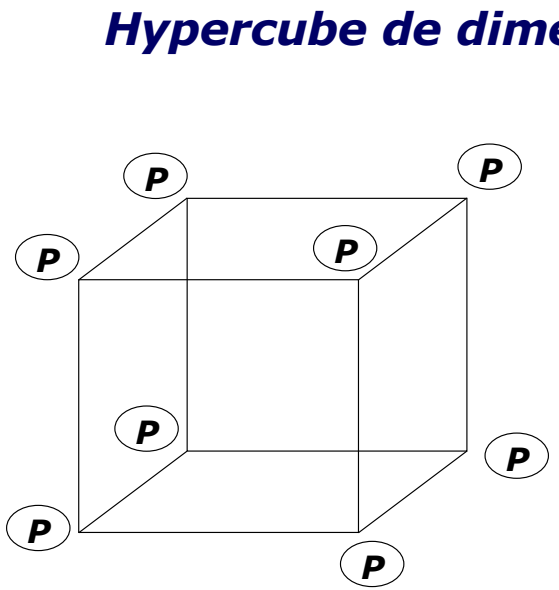
- Bus : Les processeurs(et la mémoire) sont connectés à un bus commun
  - Simple à construire
  - Coût faible
  - Technologie mature
  - Les accès mémoire ne sont pas "scalable" du aux contentions
- Switching Network : Les processeurs(et la mémoire) sont connectés à un switch effectuant le routage comme dans un système téléphonique, le système de routage détermine le chemin optimal
  - Meilleure Scalabilité
  - Surcoût important

Réseaux



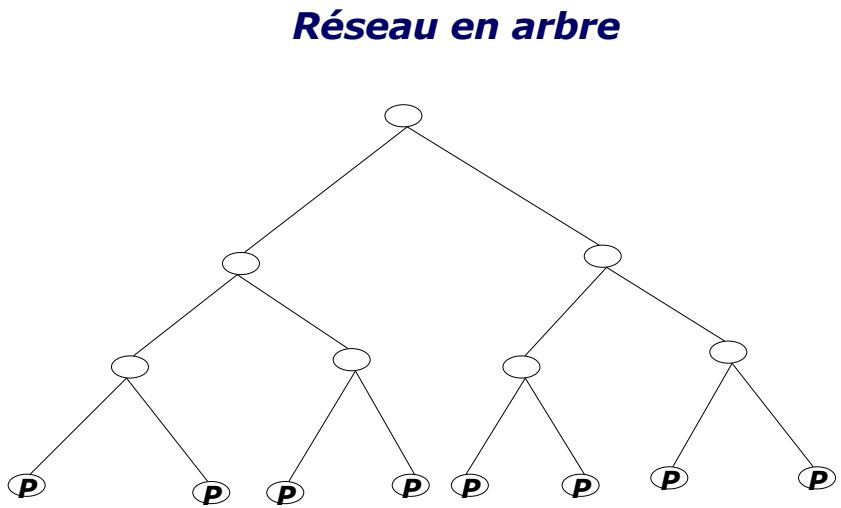
Hypercube

- Chaque noeud est le sommet d'un hypercube de dimension N
- Chaque est directement connecté à N autres noeuds



Réseau en arbre

- Les processeurs sont noeuds d'un arbre
- Pour accéder à une certaine mémoire, le processeur devra parcourir l'arbre

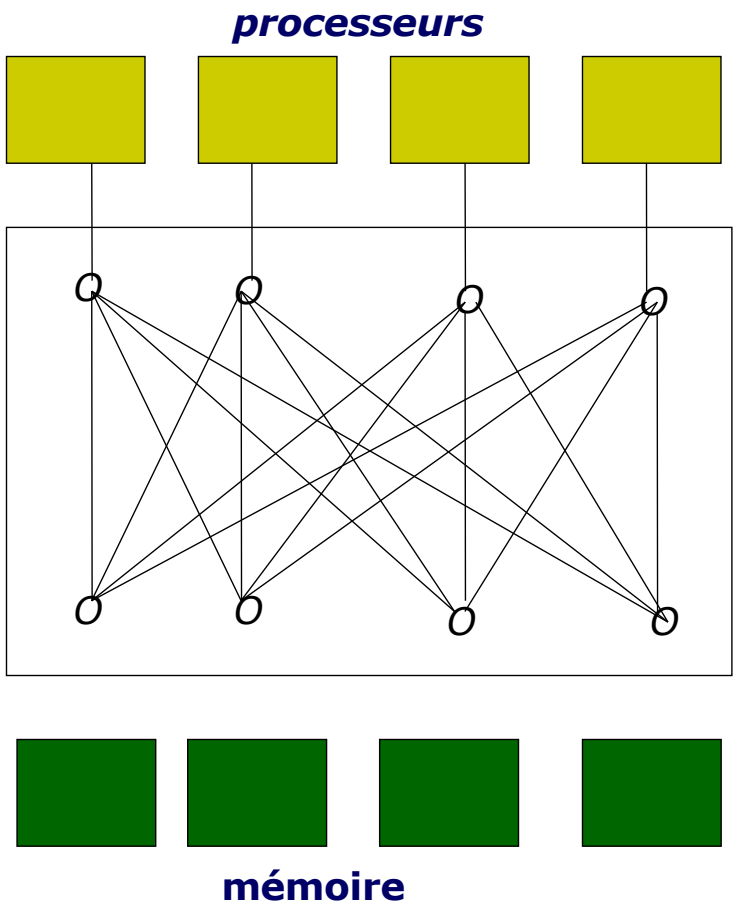


Réseau d'interconnexion complet

- On divise la mémoire en bancs de mémoire
- Adressage global : chaque processeur a accès à chaque banc de mémoire
- Le nombre de liens augmente avec le nombre de processeurs et le nombre de bancs mémoire $\Longrightarrow$ Réseau non extensible

$\Longrightarrow$ Crossbar-switch

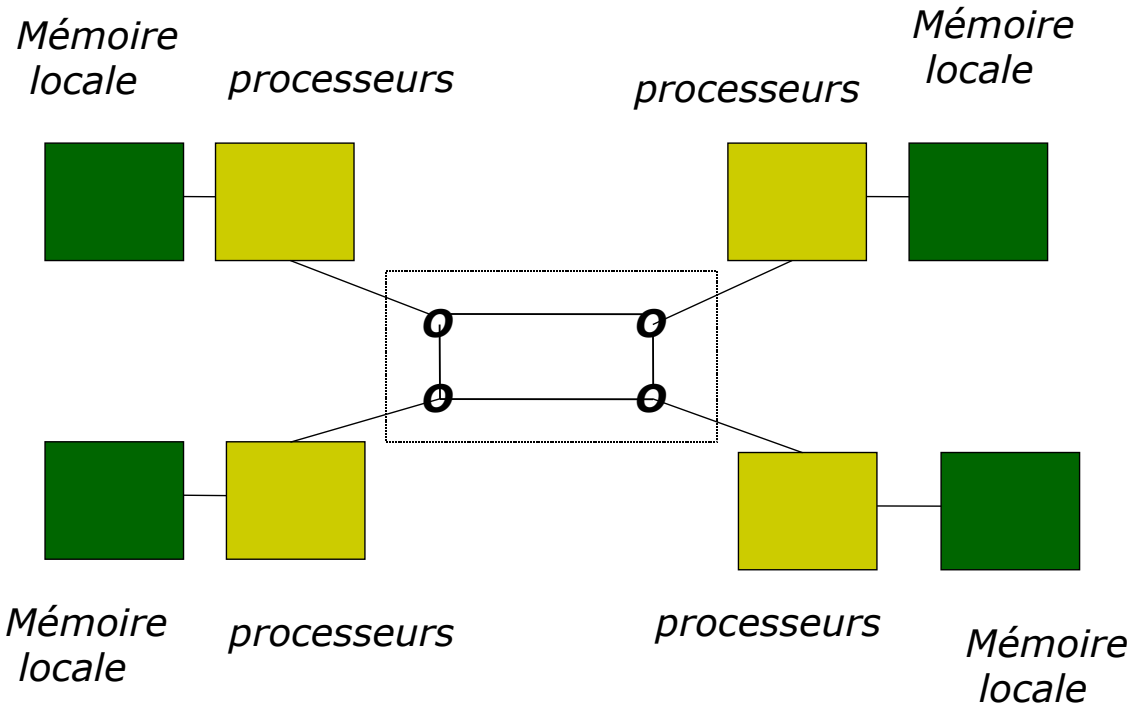
Réseau d'interconnexion complet



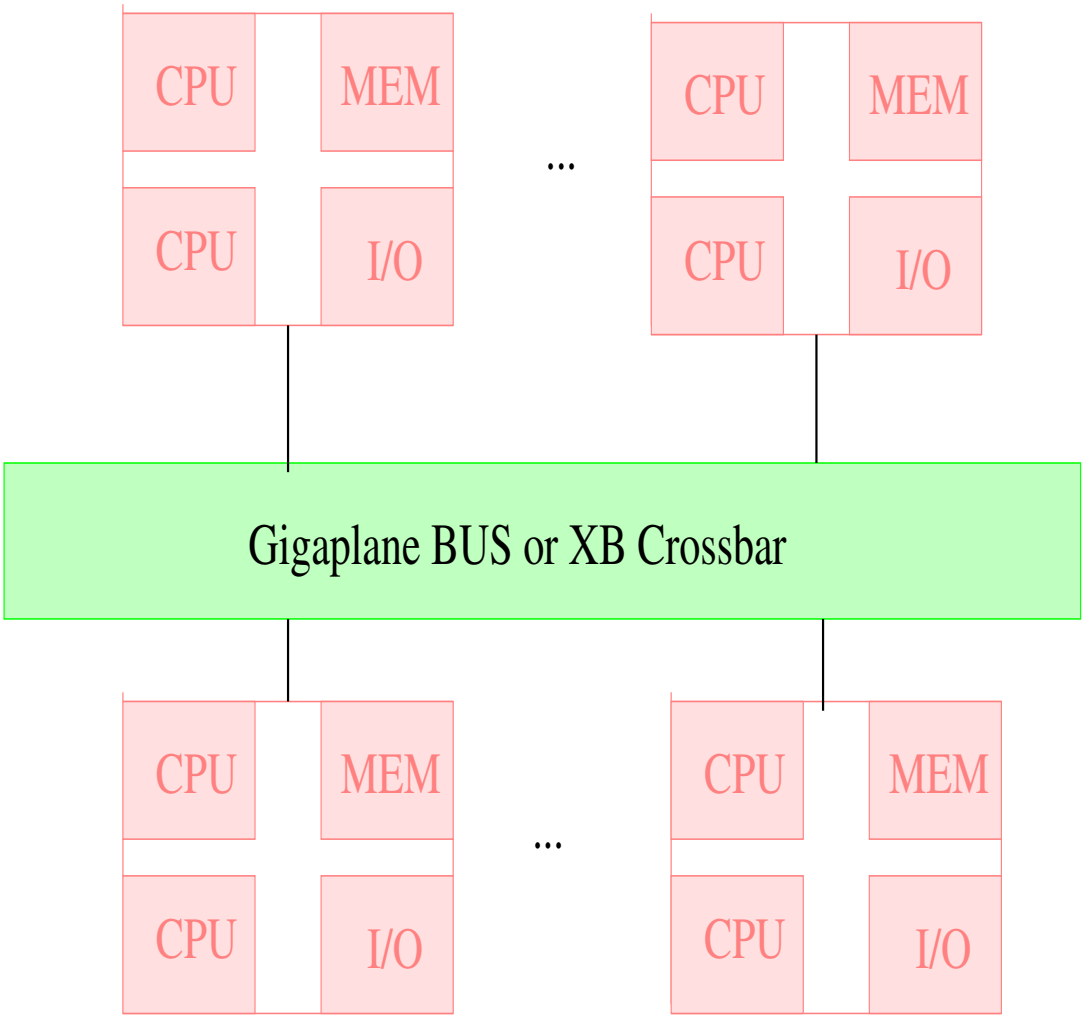
Trop de liens

Réseau d'interconnexion par processeur

Réseaux à interconnexion de processeurs



SUN Enterprise x500/E10000



Interconnect crossbar bus

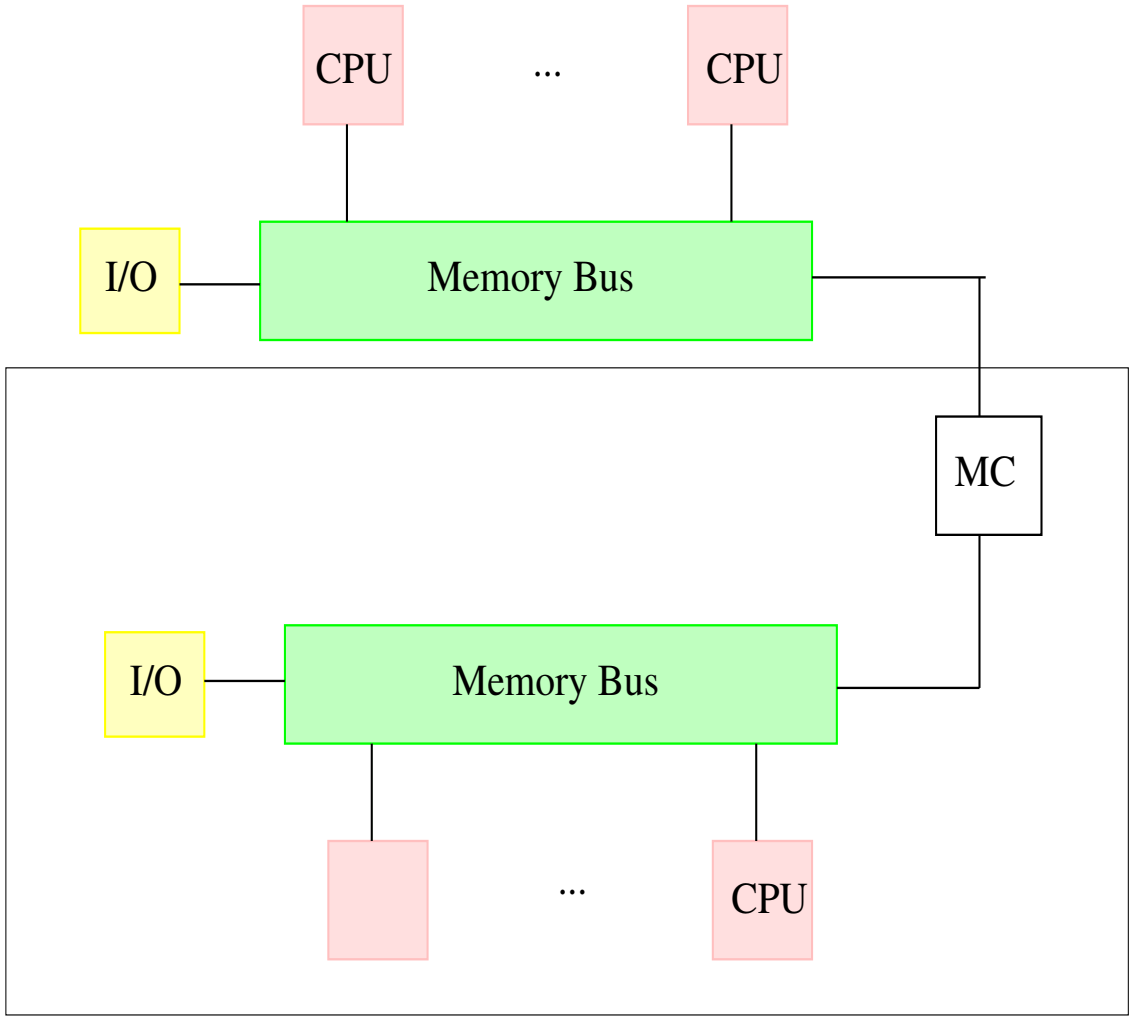
Maw #CPUS: 30/64 @466Mhz

Max Memory 60/64 GB NUMA

Node boards: 8/16

2/4 CPUs

Hewlett Packard L N-class

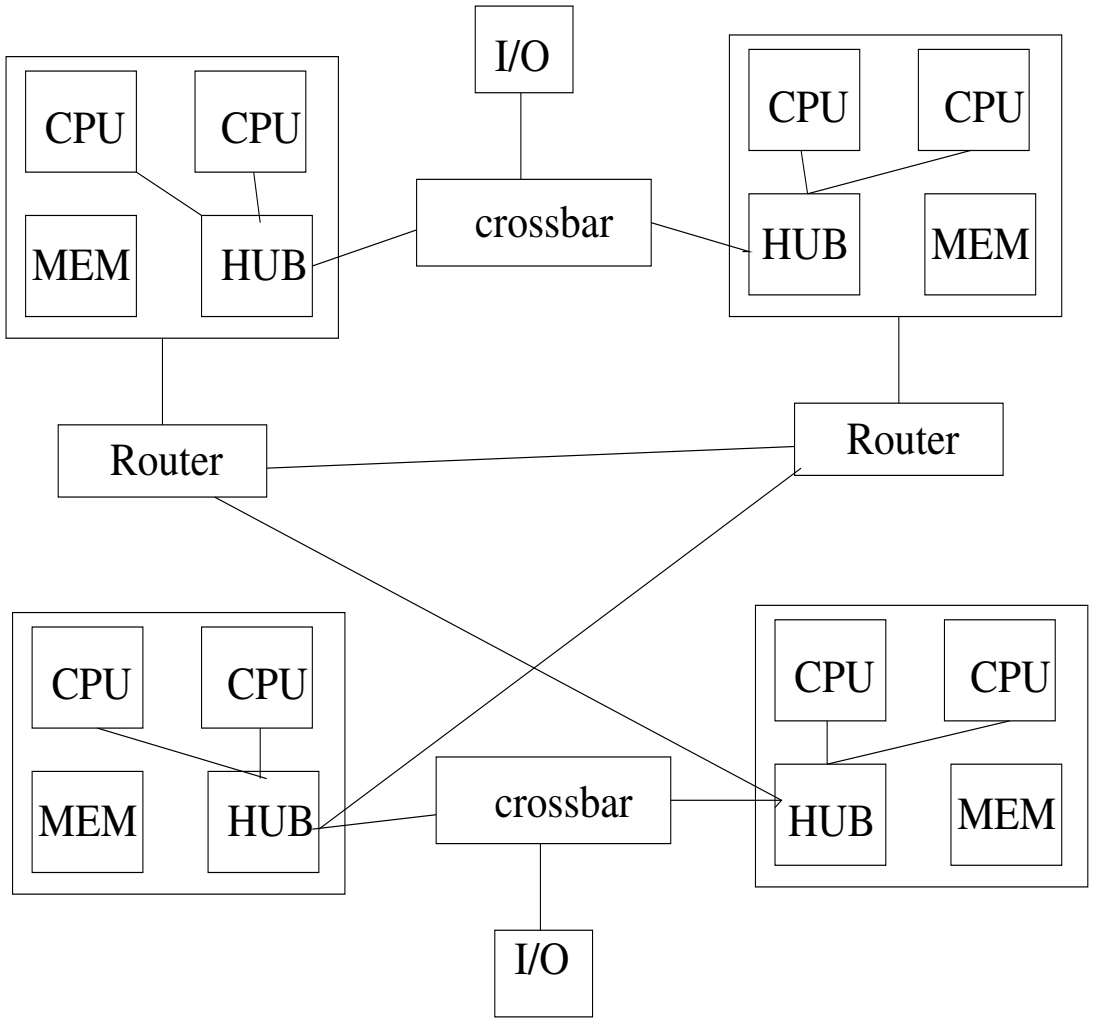


Interconnect 2 X Bus

Max # CPUs 4/8 550MHz

Max Memory 16/32 GB /UMA

Série OriginX000



Interconnect crossbar /Hypercube

Max #CPUs : 64–512 500Mhz

Max Memory 1TB /NUMA

Node Boards: 32–256
2 CPUs

Sommaire

Modèle de programmation parallèle

- Décomposer un problème en plusieurs petits problèmes affectés à plusieurs processus. Deux types de décomposition :
  - Décompositon de données ou **“data parallélisme”**
  - Décompositon de tâches
- Dépend essentiellement de l’organisation de la mémoire
  - Mémoire distribuée
  - Mémoire partagée
  - Mémoire partagée distribuée
- Un critère supplémentaire dans l’analyse de performance de l’application // : les performances des communications.

Sommaire

Modèle de programmation sur architecture à mémoire distribuée

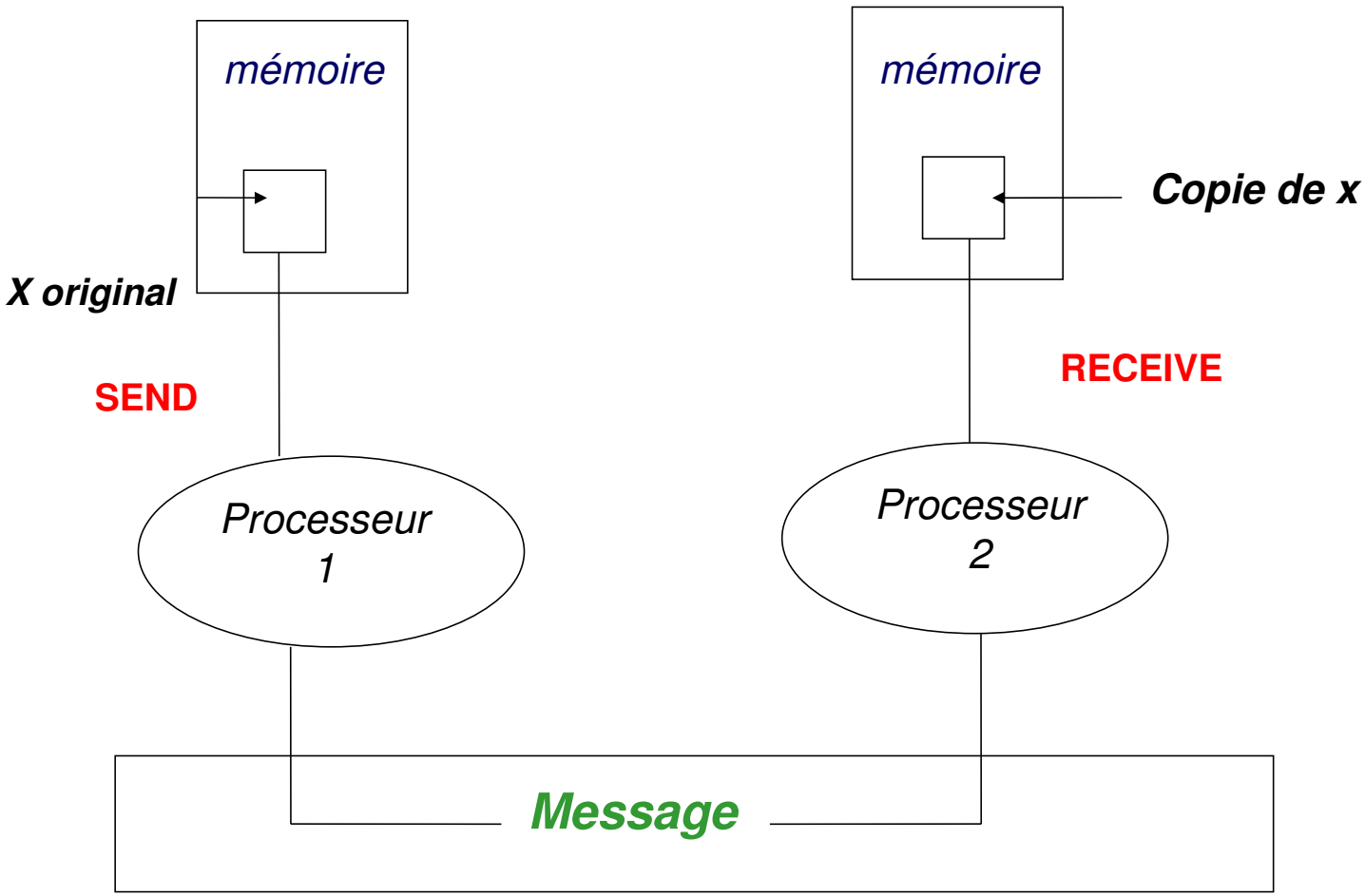
- Control Model : un nombre fixe de processus(threads) exécutant le même code sur chacun des processeurs
- Message Passing : permet uniquement la référence à la mémoire locale du processus, l'accès à la mémoire du processus voisin se fait par des échanges de messages
- Partitionned Global Address Space(PGAS) : permet la référence à la mémoire locale du processus et à la mémoire des processus voisins

Message Passing

Un espace mémoire est associé à chaque processus/processeur

- L'accès à la mémoire du processus/processeur voisin se fait par échange de messages entre les processus
- Chaque processus travaille sur des variables privées qui résident dans sa propre mémoire locale
- Les algorithmes utilisés devront minimiser les communications entre les processus(distribution des donnés, minimisation des envois,...)
- L'implémentation de ce type de modèle peut se faire sur tout type de machines //
- Utilisation des bibliothèques MPI (Standard), PVM(plus maintenu) ou des bibliothèques constructeurs(non portable)

Modèle par échange de messages



Message Passing : Mesure des performances

Trois critères permettront d'évaluer les performances d'une application sur une architecture donnée

Exemple : Power5 d'IBM, 1.9GHz avec une version optimale de MPI

- Vitesse du processeur : 1900 cycles/μsec, 4 flops/cycle, 7600flops/μsec
- Latence des messages $\simeq 5\mu sec$= 38,000 flops
- Bande passante (limité en général par le hardware) $\simeq 1700 bytes/\mu sec = 4.5 flops/byte$

Message Passing : améliorer les performances

- Réduction de la latence
  - Réduire le nombre de messages en localisant sur un même processeur des entitées qui communiquent
  - Envoyer en un seul message plusieurs messages qui ont les mêmes émetteurs récepteurs
  - Envoyer des données avant que le processus qui en a besoin les demande et utiliser des données reçues le plus tard possible pour augmenter la probabilité que les données soient arrivées
- Superposer l'envoi de messages et les calculs en utilisant les communications asynchrones
  - Attention au deadlock, l'asynchronisme est difficile à gérer quand le nombre de processus augmentent
  - Tous les systèmes ne permettent pas de le faire
  - La latence est principalement due au logiciel, l'initialisation d'un message occupe le processeur

Message Passing Interface - MPI

MPI interface Message Passing utilisée pour les nouveaux développement à grande échelle(PVM est la version la plus ancienne) :

- Interface standard amené à évoluer
- Bibliothèque de fonctions et de sous-programmes en C,C++ et Fortran
- Implémentée et optimisée sur un grand nombre d'architectures parallèles, optimisation à la charge des constructeurs
- peut être implémenté efficacement sur des architectures à mémoire partagée, n'exploite pas toutes les possibilités de la mémoire partagée (copies)
- Deux versions du standard MPI-1 (1994) et MPI-2 ()
- De nombreux tutoriaux, outils d'aide au développement accessibles sur Internet : www.mpi-forum.org, www-unix.mcs.anl.goc/mpi,...
- De nombreux logiciels sont dévelopés en utilisant MPI
- MPI devrait passer à l'échelle (Petascale Computing)

MPI-1 : quelques fonctionnalités

- Communications collectives : optimisation, clareté, réduit les risques d'erreur
- Opérations de réduction
- Création de sous communicateurs : communications collectives sélectives
- Support pour l' utilisation de systèmes hétérogènes : conversion de types intrinsèques,...
- Types dérivés utilisateur permettant la communication de structures : optmisation, clareté
- ...

MPI-2

L'implémentation du standard MPI-2 n'est pas complète dans toutes les versions de MPI. Les apports concernent principalement :

- E/S parallèles : scalabilité des applications effectuant des E/S intensives
- One-sided communication : essentiellement des **put** et des **get**, augmente l'efficacité des communications
- Gestion dynamique des processus
- ...

Sommaire

Programmation mémoire partagée

- Espace mémoire global, accessible par tous les processeurs
- Les processeurs peuvent avoir dans leur mémoire locale une copie d'une partie de la mémoire globale
- la consistence entre ces copies est en général maintenu par le hardware, le software et parfois l'utilisateur
- Avantages :
  - Un espace d'adressage global est plus facile à utiliser
  - l'échange de données est rapide
- Inconvénients :
  - L'utilisateur devra s'assurer que les conflicts d'accès à la mémoire sont synchronisés
  - Défaut de scalabilité des accès mémoire
  - Problèmes de 'Race detection', les outils permettant de les détecter ne sont pas encore matures

Programmation mémoire partagée

- Utilisation de solutions spécifiques constructeurs via des directives de parallélisations gérées par le compilateur jusqu'au milieu des année 90
- Extensions concernant le "data parallélisme" dans le langage Fortran, Fortran90, High Performance Fortran(HPF)
- À partir de 1997 : OpenMP devient le standard pour la programmation sur des architectures à mémoire partagée avec pour objectifs :
  - d'être un standard
  - d'être facile à utiliser : un nombre restreint de directives simples de parallélisation permettant la parallélisation incrémentale d'un programme séquentiel là où MPI impose une approche plus globale
  - d'être portable

Mémoire partagée : standard OpenMP

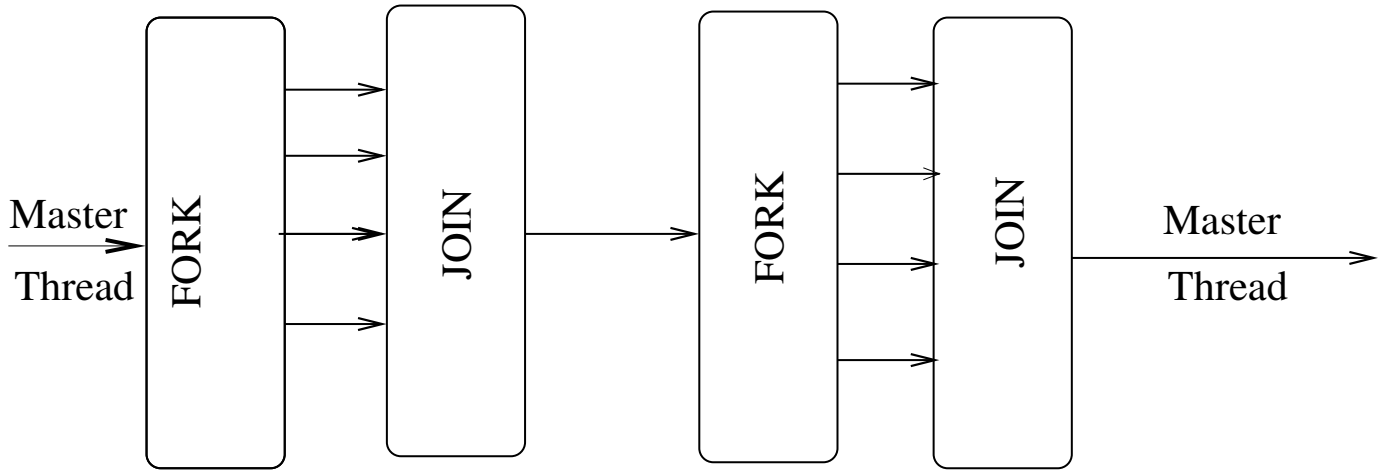
- API (Application Program Interface) permettant la parallélisation mémoire partagée multi-threaded
- Basée sur une combinaison de directives de parallélisation, de routines et de variables d'environnement
- Compilateurs C,C++ et Fortran
- Modèle de programmation explicite : le programmeur a le contrôle sur la parallélisation
- OpenMP utilise le modèle d'exécution parallèle "fork_join "
- 2005 : spécifications de la version 2.5, plus de clareté, intégration dans la plupart des compilateurs
- OpenMP home page : http ://www.openmp.org
- tutoriaux et documentation : http ://www.llnl.gov/computing/tutorials/openMP , http ://ci-tutor.ncsa.uiuc.edu/login.php

OpenMP : 3 types de composantes

- Directives de parallélisation
  - Régions parallèles (PARALLEL)
  - Boucle parallélisée (PARALLEL DO)
  - Sections parallèles (PARALLEL SECTION)
  - Section exécutée par un seu processeur) (SINGLE
  - Synchronisation (BARRIER,CRITICAL,ATOMIC,...)
  - Propriétés des données (PRIVATE,SHARED,REDUCTION,...)
- Run-time library routines
  - OMP_SET_NUM_THREADS
  - OMP_GET_NUM_THREADS
- Variables d'environnement positionnées avant l'exécution
  - OMP_NUM_THREADS
  - ...

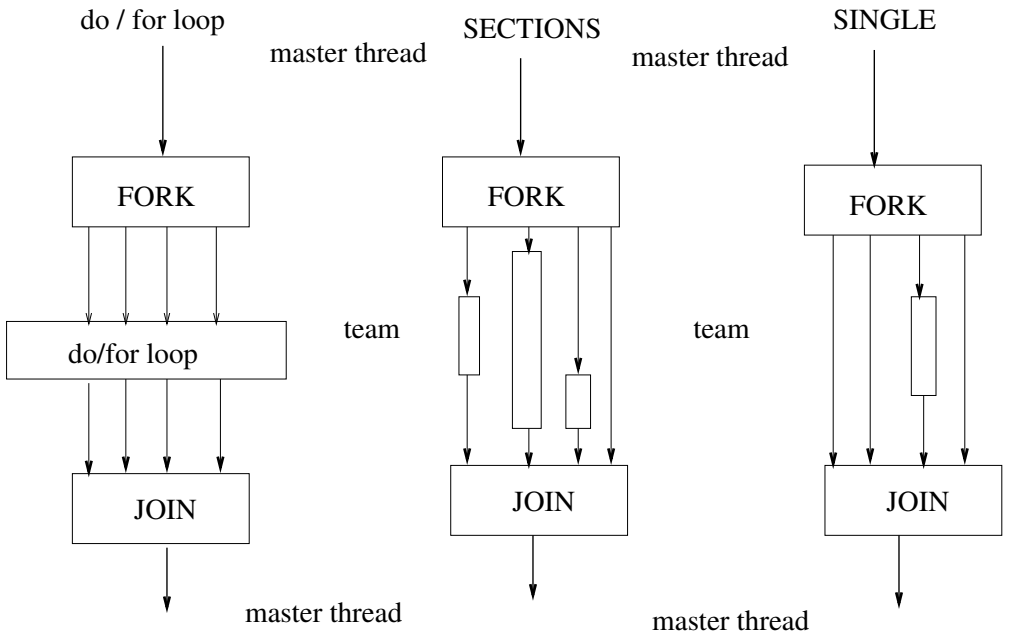
OpenMP : modèle fork_join

- Le thread maître s'exécute séquentiellement jusqu'à la première région parallèle
- FORK : le thread maître crée un ensemble de threads qui s'exécutent en //
- JOIN : Lorsque tous les threads ont terminé leur travail, ils se synchronisent et le thread maître continue séquentiellement
- Le nombre de threads est indépendants du nombre de processeurs



Constructions définissant le partage du travail

- DO/for loops : data parallélisme
- SECTION : découpe le travail global en section indépendantes qui sont exécutées en //
- SINGLE : sérialise une partie de code. Utile pour des sections qui ne sont pas threadsafe (ex : E/S).



OpenMP : Traps

OpenMP est souvent un moyen très rapide d'écrire des codes parallèles mais c'est souvent difficile à déboguer

- Les threads communiquent en partageant des variables
- La portée des variables : partie complexe de la parallélisation en mémoire partagée. Elles sont :
  - les variables partagées
  - les variables privées
- utiliser de préférence des bibliothèques threadsafe
- éviter les E/S séquentielle dans une région parallèle : ordre des E/S indéterministe

Mémoire partagée - False Sharing

- Cache : zone mémoire à accès rapide entre la mémoire du processeur à accès lent et les registres à accès très rapide
- Sur une machine à mémoire partagée, la cohérence entre les caches et la mémoire devra être maintenue, elle le sera par unité d'une ligne de cache
- la modification d'une seule donnée dans la ligne de cache invalide toute la ligne
- pendant la mise à jour d'une ligne de cache, toutes les données de cette ligne sont inaccessibles

Cette situation, appelée false sharing peut être la cause d'une dégradation des performances et de la scalabilité d'une application quand :

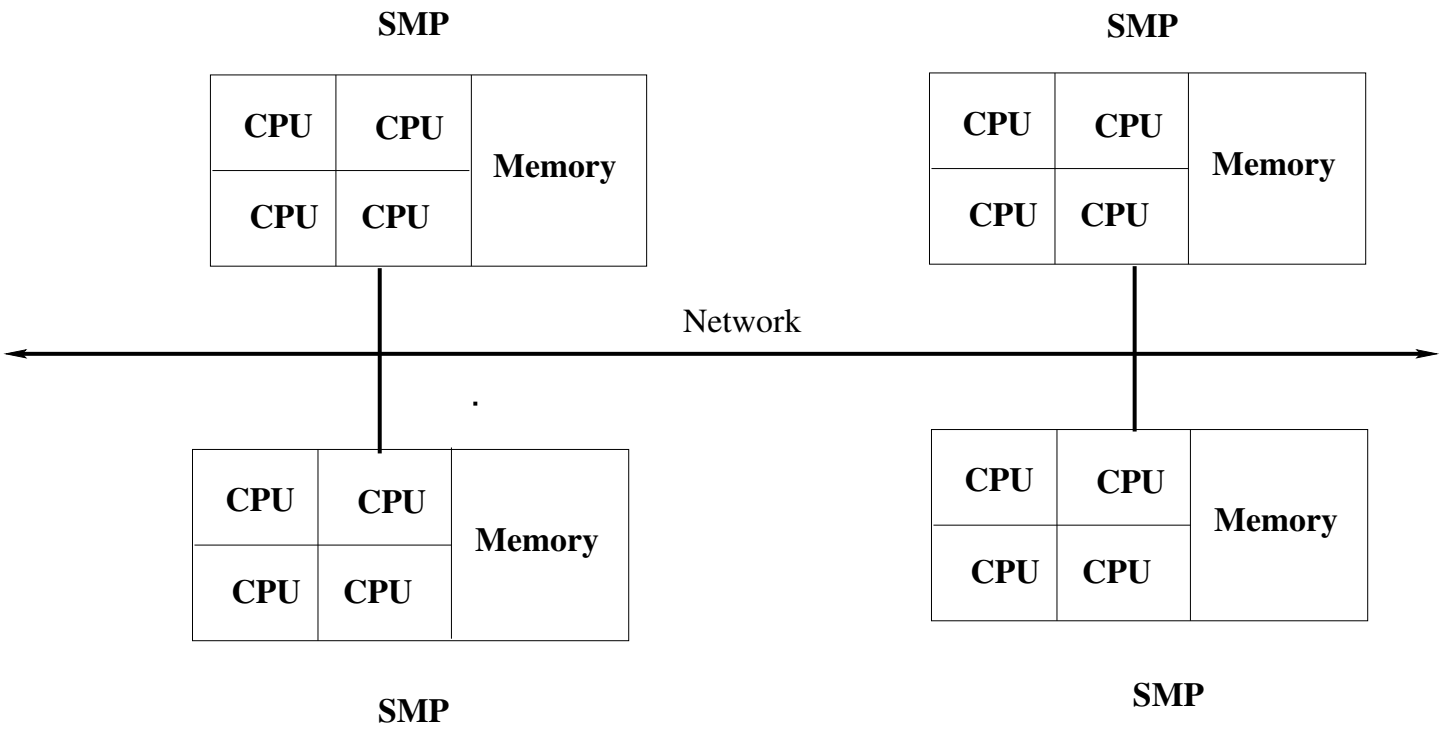
- Plusieurs processeurs modifient des données dans la même ligne de cache
- Ces modifications interviennent fréquemment
-

Mémoire partagée - Race Condition

En mémoire partagée, une principale cause d'erreur est lorsqu'un processeur lit une valeur à partir d'un espace mémoire qui n'a pas été mis à jour

- Erreur qui dépend de l'ordre d'exécution des tâches. Cet ordre est souvent non déterministe
- Souvent difficile à déboguer, le déboqueur excute le programme de façon séquentielle et déterministe
- Pour s'assurer que la lecture ne se fait pas avant l'écriture, l'utilisateur devra lui-même gérer la synchronisation
- Dans les système à mémoire distribuée, les messages effectuent souvent la synchronisation(message bloquant dans MPI...)
- Dans les systèmes à mémoire partagée, on utilisera les barrières, les sémaphores, les locks...

Architecture hybride



CPU: Single core or Multi core

Un processus par core

ou

Un processus MPI par noeud, OpenMP à l'intérieur des noeuds

Architecture hybride - Modèle de programmation

Souvent : les applications ont deux niveaux naturels de parallélisme. Si possible, en tenir compte tout en exploitant le parallélisme mémoire partagée sur les noeuds SMP

Pourquoi ?

- Les performances MPI se dégradent quand :
  - les domaines sont trop petits
  - la latence des messages dominent le calcul
- OpenMP
  - latences plus faibles
  - peut maintenir un speedup pour une faible granularité

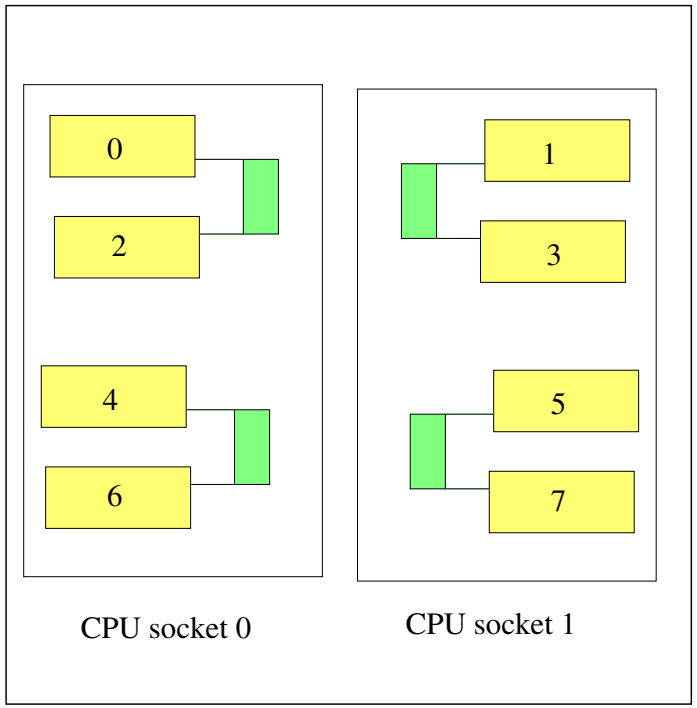
Inconvénients

- Le programmeur devra connaître MPI et OpenMP
- Le code peut devenir plus difficile à déboguer, à analyser et à maintenir

Support MPI pour les threads

- Toutes les implémentations MPI ne sont pas thread-safe mais la plupart le deviennent
- 4 niveaux de supports (MPI-2)
  - MPI_THREAD_SINGLE : un seul thread par process
  - MPI_THREAD_FUNELLED : plusieurs threads de calcul, uniquement le thread maître peut faire des appels MPI (openMP dans le thread maître)
  - MPI_THREAD_SERIALIZED : tous les threads peuvent faire des appels MPI mais un seul à la fois (peut faire des appels MPI dans les “critical” sections ou les “single” section)
  - MPI_THREAD_MULTIPLE : tous les threads peuvent faire des appels MPI concurrents (pas de contrainte d’OpenMP)
- Modèle MPI
  - Évite l’utilisation de deux modèles de programmation différents
  - Ne nécessite aucun changement lorsque le nombre de cores par chips augmentent

Performance sur Multi-core



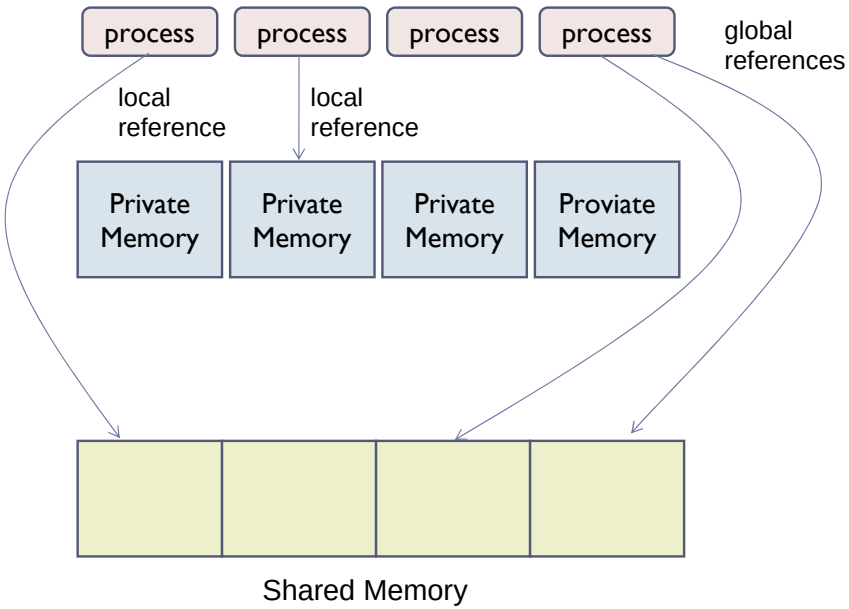

```
#define N 65536 /*64K/
int main(void)
                        /* cache.c*/
{
      char source[N],destination[N];
      int j;
#pragma omp parallel for    shared destination
      for (j=0;j<1000000;j++)
          memcpy(destination, source,N)
      return(0)
  }
```

```
icc –openmp cache.c
setenv OMP_NUM_THREADS 2
time  taskset  –c 3,4  a.out
time  taskset  –c 3,1  a.out
time  taskset  –c 5,1  a.out
```

- Dans une architecture partageant le cache L2, grande passante mémoire pour les cores qui partagent des données à travers le cache L2
- Dans les architectures ne partageant pas le cache L2 : pénalisation due au traitement de la cohérence des caches
-

Partitioned Global Address Space

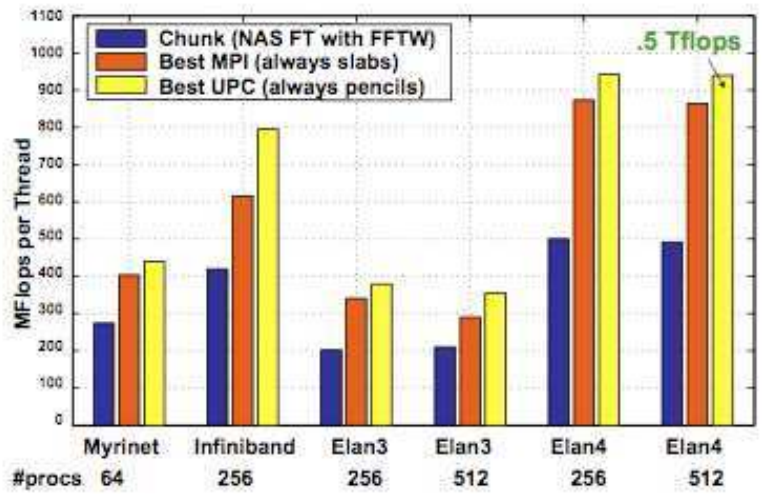
PGAS Memory Model



Unified Parallel C (UPC)

- V1.0 publié en 2001, V1.2 dernière version
- Développé par un consortium sous l'influence des agences gouvernementales
- Version commerciale et Domain Public disponible

What Performance Do We Get? (1)



3D-FFT (Yellick et al., Berkeley compiler)

Co-Array Fortran

- Développé par Numrich en 1998
- Supporté par Cray, implémentation OpenSource
- Proposition d'inclusion dans la norme Fortran2008
- Même modèle d'exécution qu'UPC

Migration from MPI to PGAS

- PGAS languages have same execution model as MPI (assuming no work-sharing used)
- Replace arrays by shared arrays (allocated independently on each thread)
- Replace send/receive with memcopy to/from shared arrays
- Use local references for local code
  - Need an extra level of indirection in UPC
- If code is well-structured, should affect only small fraction of code (lower levels of hierarchy)
- Possible gain: better performance for fine-grain communication
  - Compiler can map directly to communication HW
  - Compiler can perform optimizations across multiple communications

PGAS langages : conclusion (Marc Snir)

PGAS Summary

- Languages still need to evolve
  - But are already in a useful state
- Do not really provide a higher-level programming level
- Can provide lower communication overhead
- Can better hide the shared-memory/distributed-memory distinction – well-suited to handle multi-core
- Main additions needed:
  - Teams, Collectives, Adaptivity, **Good implementations**

Threading Building Blocks (TBB)

- Bibliothèque publiée par Intel en 2007
- Développée en C++, elle permet d'abstraire au maximum les détails délicats de la programmation multithreading en utilisant les concepts objets (template,programmation générique). Elle se compose principalement :
  - d'algorithmes parallèles
  - de containeurs

  Ces algorithmes pourraient être utilisés pour reprogrammer une version parallèle de la STL
- Sous double license, commerciale et libre(GNU GPL) (même version sans support technique)
- Fonctionne sur différents compilateurs(Intel, Microsoft, gcc), elle se veut indépendante de l'architecture

TBB : conclusion

- TBB a :
  - Containeurs parallèles (queue, hash table)
  - Locks
  - User control on scheduling
  - Gestion “scalable” de le mémoire
- Utilisation intensive de C++ (templates, classes, overloading)
- Intel est derrière
- TBB est adpaté aux architectures à mémoire partagée (multi-core en particulier), on pourrait avoir une approche similaire sur des architectures à mémoire distribuée

Sommaire

Définition des métriques d'efficactité

Soit A(n) un problème de taille n

SerTemps(n) : temps du meilleur algorithme séquentiel permettant de résoudre A(n)

ParTemps(n,p) : temps pour un algorithme parallèle donné permettant de résoudre A(n) sur p processeurs

on définit :

$$\textbf{Speedup} = \frac{\textbf{SerTemps(n)}}{\textbf{ParTemps(n, p)}}$$

$$\textbf{Efficacite} = \frac{\textbf{SerTemps(n)}}{\textbf{p} * \textbf{ParTemps(n, p)}}$$

En général :

$$0 < Speedup < p$$

$$0 < Efficacite < 1$$

Speedup Linéaire ou Superlinéaire

L'efficacité est dite :

Linéaire si il existe une constante $c > 0$ telle $speedup = c * p$, on utilise souvent $c = 1$

Superlinéaire lorsque $speedup > p$ ou $efficacite > 1$, très rare.
Quelques raisons pour qu'un $speedup > p$

- un programme parallèle a p fois plus de RAM, certains algorithmes vont améliorer les accès mémoire
- l'algorithme parallèle est plus performant que l'algorithme séquentiel

Loi d'Amdahl

Pour un problème donné, on pose :

- **f** : fraction du programme qui doit être exécuté en séquentiel
- **p** : nombre de processeurs utilisés

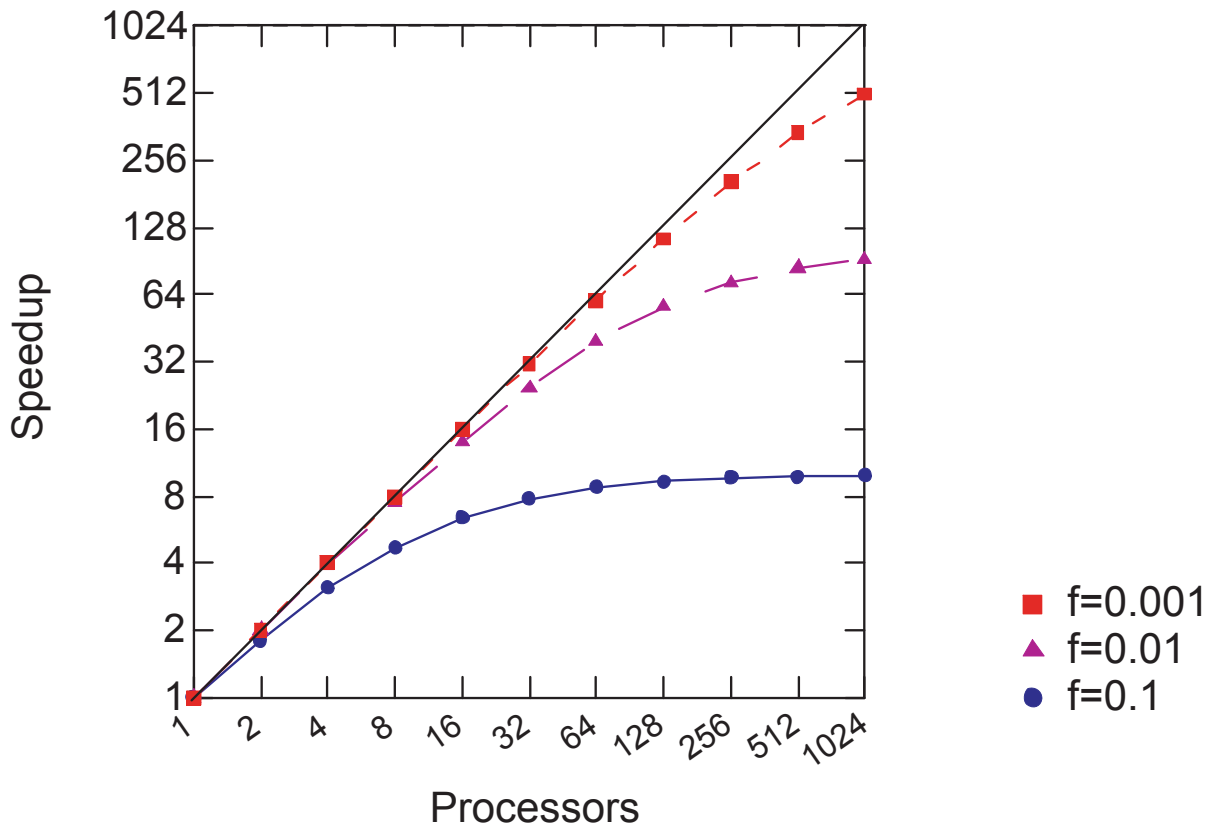
$$\mathbf{Speedup(p)} \leq \frac{\mathbf{1}}{(\mathbf{f} + \frac{\mathbf{1-f}}{\mathbf{p}})}$$

Quel que soit le nombre de processeurs, on a :

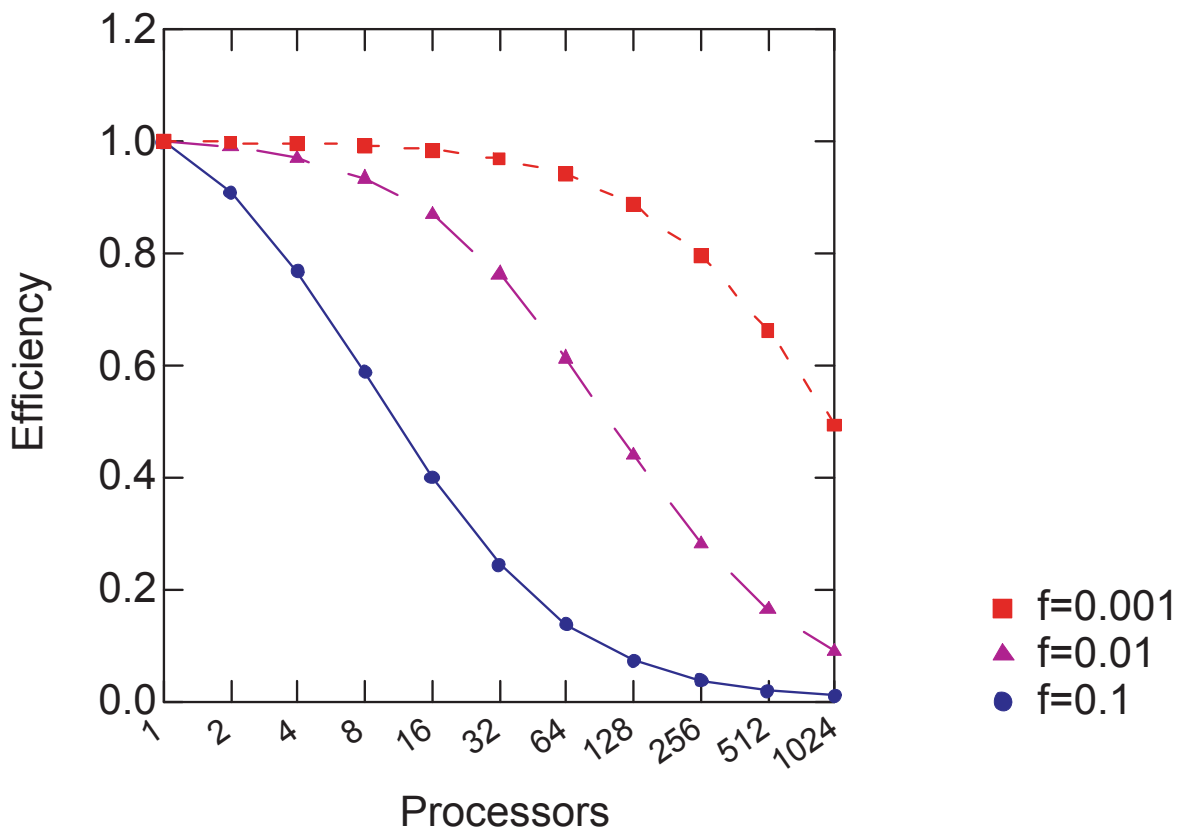
$$\mathbf{Speedup(p)} \leq \frac{\mathbf{1}}{\mathbf{f}}$$

Hélas, souvent f est compris entre 10 et 20%

Maximal Possible Speedup



Maximal Possible Efficiency



Loi d'Amdahl

La loi d'Amdahl peut être :

optimiste : La loi d'Amdahl ne tient pas compte du surcoût dû à la parallélisation, typiquement les communications

pessimiste : La loi d'Amdahl ne tien pas compte :

- de la possibilité d'amélioration de l'algorithme
- d'amélioration des accès mémoire
- la fraction de code séquentiel diminue souvent avec la taille du probème(Scaling)

Sommaire

Scaling et Scalabilité

Scaling : analyses effectuées sur l'implémentation d'une application // lorsqu'on augmente le nombre de processeurs

Scalabilité : capacité d'un algorithme // à augmenter ses performances avec le nombre de processeurs sur une architecture donnée

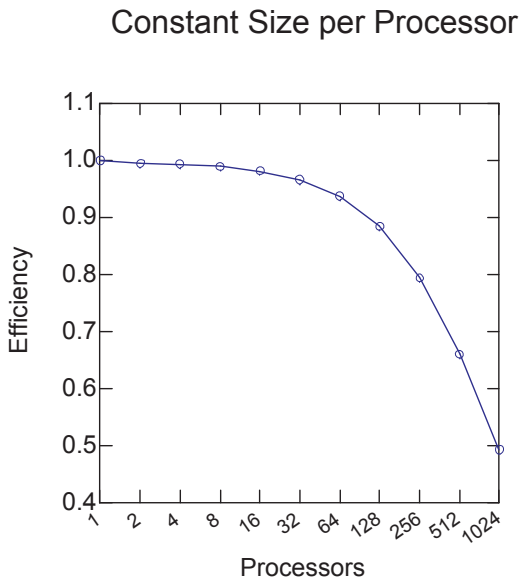
Ce qui pénalise la scalabilité :

- L'overhead dû aux communications
- La répartition plus ou moins équilibrée des tâches sur les processus (Load-Balancing)
- La plus ou moins grande cohérence entre l'algorithme et l'architecture du calculateur

Fixed Size per Processor

Fixing the amount of data per processor usually gives highest efficiency possible, hence it is commonly cited.

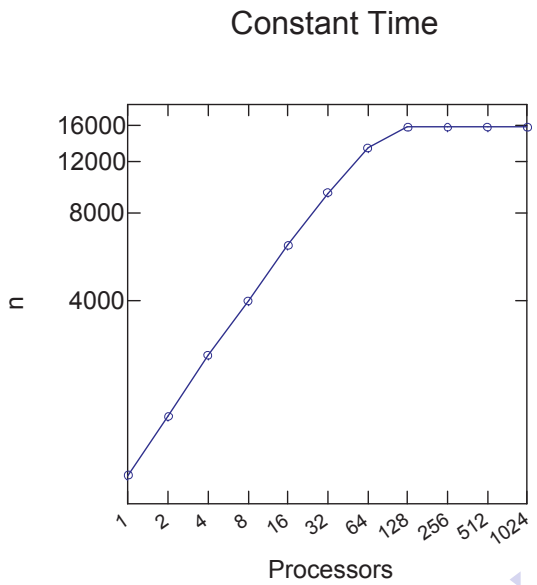
Suppose each processor can hold 1000 elements.



Fixed Time

Fix time, find largest problem solvable. Commonly used in evaluating database servers, *transactions per second*. [Gustafson 1988] considered this for general computing.

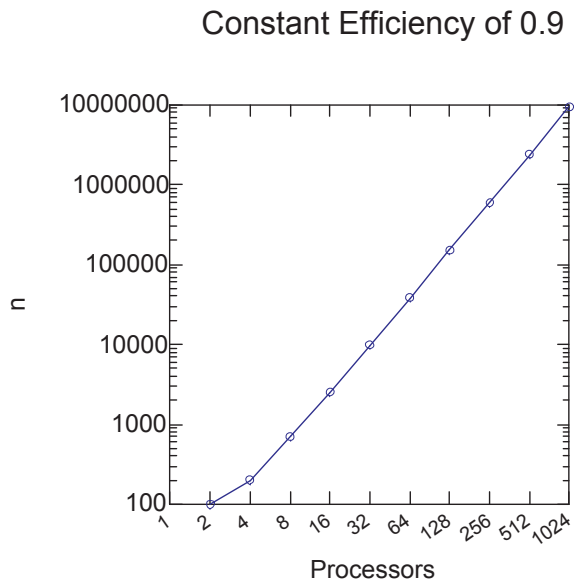
Fix time to be *SerTime(1000)*.



Fixed Efficiency

Fix efficiency, find smallest problem needed to achieve that efficiency (isoefficiency analysis).

For example, for 90% efficiency:



Load Balancing

- Distribuer le travail de façon équitable sur tous les processus
- Minimiser le temps global d'attente de tous les processus
- Le "Load Balancing" de l'application est important, le temps du processus le plus lent détermine le temps de l'application

Comment obtenir un bon équilibrage des tâches

- Chaque processus reçoit une part égale du travail
  - Pour des opérations sur des tableaux/matrices où chaque tâche effectue le même travail
  - Pour une boucle d'itérations où chaque itération fait la même quantité de travail
  - Si on utilise des machines ayant des caractéristiques de performances différentes, en tenir compte pour la distribution des tâches
- Distribution dynamique du travail
  - Certaines classes de problèmes n'équilibrent pas les tâches, ême si les données sont distribuées de façon équilibrée
    - Matrices creuses
    - Méthode à maillage adaptatif
    - N-Body problèmes
  - Lorsque la quantité de travail est inprévisible, il est souhaitable d'utiliser une approche de scheduling, qui fournit du travail chaque processus qui termine sa tâche

Sommaire

Analyse de performances d'une application parallèle

- Développer de grosses applications scientifiques qui font une utilisation optimale des ressources de calculs à disposition présentent quelques difficultés
- les ressources peuvent soit être inefficacement utilisées soit sous utilisées
- Les facteurs qui déterminent l'efficacité d'une application sont complexes, interdépendants et parfois cachés aux utilisateurs
- Aussi, les outils d'analyse de performances sont essentiels pour optimiser une application séquentielle ou parallèle

Analyse de performances : Applications séquentielles ou parallèles

- Objectifs : réduire le temps d'exécution d'un programme
- Approche itérative :
  - Timing, profiling
  - Analyse des parties consommatrices du code :compteurs harware,..
  - Optimiser/paralléliser les parties consommatrices, éliminer les goulets d'étranglement
  - Évaluer les performances, améliorer l'application/paralléliser
- Insuffisances des outils classiques d'analyse de performances (timing, compteurs Hardware, profiling,...) :
  - ils mesurent les temps totaux consommés et n'indiquent pas quand ce temps est consommé $\Longrightarrow$ outils d'analyse de traces
  - ils ne tiennent pas compte :
    - des communications
    - du temps d'attente pour l'envoi ou la réception des messages
    - du load-balancing de l'application

Analyse de performances : intéractions entre système et application

- Facteurs liés au système
  - tous les facteurs associés à un seul CPU
  - le réseau de communication entre les processeurs
- Facteurs liés à l'application
  - tous les facteurs associés à une application séquentielle
  - la bibliothèque "Message Passing", OpenMP
  - le type de communications
  - la granularité des tâches
  - Load balancing

Sommaire

Temps d'exécution d'un programme parallèle

- MPI/OpenMP fournissent des fonctions standards qui mesurent le "wallclock time" entre deux points du programme :
  MPI_WTIME(MPI) OMP_GET_WTIME(OpenMP)
- Ces fonctions sont intrusives dans le code
- Pour évaluer le speedup, on utilise le "wallclock time", le temps CPU ne tient pas compte de l'overhead pour la parallélisation
- les profileurs (gprof) sont insuffisants pour les codes parallèles
  - ils mesurent les temps CPU et non le ""wallclock time"
  - ils font la somme sur toutes les invocations de chaque routine
  - ils ne peuvent mettre en évidence le déséquilibre des tâches

Outils d'analyse de traces pour MPI

- Intel Trace Analyzer & Collector : outils d'analyse de performances de programmes parallèle hybride sur plate-formes Intel.
- Cet outil est construit sur Vampir et Vampirtrace actuellement supporté et développé par l'Université de Dresde (http ://vampi.eu)
- Vampir et Vampirtrace sont implémentés sur la plupart des plates-formes HPC
- Ces deux outils fournissent des traces post-mortem de l'exécution d'un programme parallèle
- L'évaluation libre de ces produits est possible pendant une courte période

Trace Analyzer & Collector / Vampir

- Trace Analyser et Vampir aide :
  - à comprendre le comportement de l'application
  - à évaluer le load-balancing
  - à analyser les performances de sous-programmes/blocs d'instructions
  - à analyser les communications et leurs performances
  - à identifier les engorgements des communications ou les "deadlock"
- Trace Collector et Vampirtrace :
  - Génère la trace des événement des appels MPI et de l'application
  - fournit une interface permettant une analyse plus approfondie de l'application

OpenMP : Tracing tools

- Intel Thread Profiler : outil commercial d'analyse de performances de programmes OpenMP sur plates-formes Intel
- Intel Trace Analyzer & Collector avec Thread Profiler permettent d'analyser des codes hybrides
- Vampir : analyse des threads parallèles dans des applications hybrides
- les outils d'analyse de traces OpenMP mesurent :
  - "wallclock time"
  - mesurent le temps :
    - d'attente aux barrières de synchronisation
    - d'attente pour et à l'intérieur d'un **locks**
    - **d'attente pour et à l'intérieur d'une section critique**
  - Regroupent des données d'analyse de performances
    - On a per-thread basis
    - On a per-region basis
  - Identies les goulets d'étranglement, le temps passé en séquentiel et évalue le load-balancing

Alternatives aux outils de visualisation de traces

- Jumpshot,Upshot et Nupshot : outils du domaine public développés au laboratoire national d'Argonne pour mpich
- XMPI : interface graphique simple, domaine public
  http ://www.lam-mpi.org/software/mpi
- TAU (Tuning and Analysis Utilities)
  - Développé à l'université d'Oregon
  - Analyse de performances de programmes parallèles en mémoire partagée et en mémoire distribuée
  - Opensource
  - http ://www.cs.uoregon.edu/research/tau/home.php

Sommaire

Déboguage d'un programme parallèle

Sur plusieurs points différent du déboguage d'un programme séquentiel

- La complexité est accrue : communications, synchronisation,...
- La technique traditionnelle utilisant les points d'arrêt n'est pas toujours utilisable avec un programme parallèle lorsqu'il n'a pas un comportement reproductible, déterministe
- Deux exécutions, avec des données identiques peuvent aboutir à des contrôles de flux différents dans le cas, par exemple de"race condition"
- les débogueurs classiques ont une exécution déterministe de l'application et n'ont pas toujours les moyens de mettre en évidence ce type d'erreur

Débogueurs parallèles

Totalview est actuellement le plus performant dans le cas d'une application parallèle, une alternative est allinea DDT(http ://www.allinea.com).
Totalview :

- Supporté par la plupart des plates-formes de calcul et des compilateurs
- Débogueur graphique pour C,C++, UPC, Fortran, HPF et code assembleur
- Multi-processus et multi-thread débogueur
- Supporte MPI,PVM et OpenMP paradigmes
- Adapté aux applications multi plates-formes
- Utilisation simple, intuitive grâce à son interface graphique

Totalview

- Commercial, diffusé par ETNUS
- Mature, version 8.6.x.x
- Intégre un outil de déboguage de la mémoire
- Caractéristique essentielle : capacité à gérer tous les processus parallèles de façon cohérente
- Totalview “home page” : http ://www.etnus.com
- Tutoriel : http ://www.llnl.gov/computing/tutorials/totalview