

# CHPS0711

## Programmation parallèle

### Cours 1

# Introduction au parallélisme

## Architectures

## Algorithmique

## Programmation



UNIVERSITÉ  
DE REIMS  
CHAMPAGNE-ARDENNE

Pierre Delisle  
Université de Reims Champagne-Ardenne  
Département de Math. Méca. et Informatique  
Octobre 2018

# Pierre Delisle

- Maître de Conférences
- Département MMI URCA
- Mail
  - pierre.delisle@univ-reims.fr
- Site Web
  - <http://cosy.univ-reims.fr/~pdelisle>



# Introduction au parallélisme

Qu'est-ce que le parallélisme ?  
À la recherche du parallélisme...  
Architectures parallèles  
Langages de programmation

# Les limites du mono-processeur

- L'augmentation de la vitesse d'un processeur devient de plus en plus coûteuse et les gains sont de plus en plus limités
- Pourtant, les besoins ne diminuent pas !
- Une solution
  - dupliquer les unités de calcul
- Difficultés
  - Dégager la concurrence des codes d'application
  - Gérer les communications/accès mémoire

# Qu'est-ce que le parallélisme ?

- Exécution d'un algorithme en utilisant plusieurs processeurs plutôt qu'un seul
- Division d'un algorithme en tâches pouvant être exécutées en même temps sur des processeurs différents
- Le but : réduire le temps de résolution d'un problème en utilisant un ordinateur parallèle
- 3 niveaux d'abstraction
  - Architectures, Algorithmes, Programmation

# Architectures parallèles

- Ordinateur parallèle : ordinateur comportant plusieurs processeurs et supportant la programmation parallèle
- Plusieurs types d'architectures
  - Distribuées
  - Centralisées
- Modèles d'architecture : simplification du fonctionnement des ordinateurs parallèles
  - Taxonomie de Flynn

# Taxonomie de Flynn

- SISD : Single Instruction Single Data
  - Ordinateurs séquentiels
- SIMD : Single Instruction Multiple Data
  - Tableaux de processeurs (arrays)
  - Ordinateurs vectoriels en pipeline
- MISD : Multiple Instruction Single Data
  - Tableaux systoliques (systolic array)
- MIMD : Multiple Instruction Multiple Data
  - Multiprocesseurs et Multi-Ordinateurs

# Algorithmique parallèle

- Approches de résolution de problèmes dans un contexte d'exécution parallèle
- Modèles algorithmiques : contextes d'exécution parallèle simplifiés pour faciliter la conception
  - PRAM
- Analyse théorique de la performance



# Programmation parallèle

- Programmation dans un langage permettant d'exprimer le parallélisme dans une application réelle
- Différents niveaux d'abstraction possibles
- La parallélisation automatique serait la solution idéale, mais c'est très difficile
- La façon de programmer n'est pas indépendante de la machine utilisée

# Bref historique du parallélisme

- ...-1970 : Accessible presque uniquement aux gouvernements (défense), très coûteux
- 1970 – 1980 : Grosses entreprises (pétrole, automobile)
- 1980 – 1990 : premières machines parallèles « abordables » (microprocesseurs, VLSI)
  - Ordinateurs parallèles très performants, mais très peu flexibles
  - PROBLÈMES : programmation, portabilité

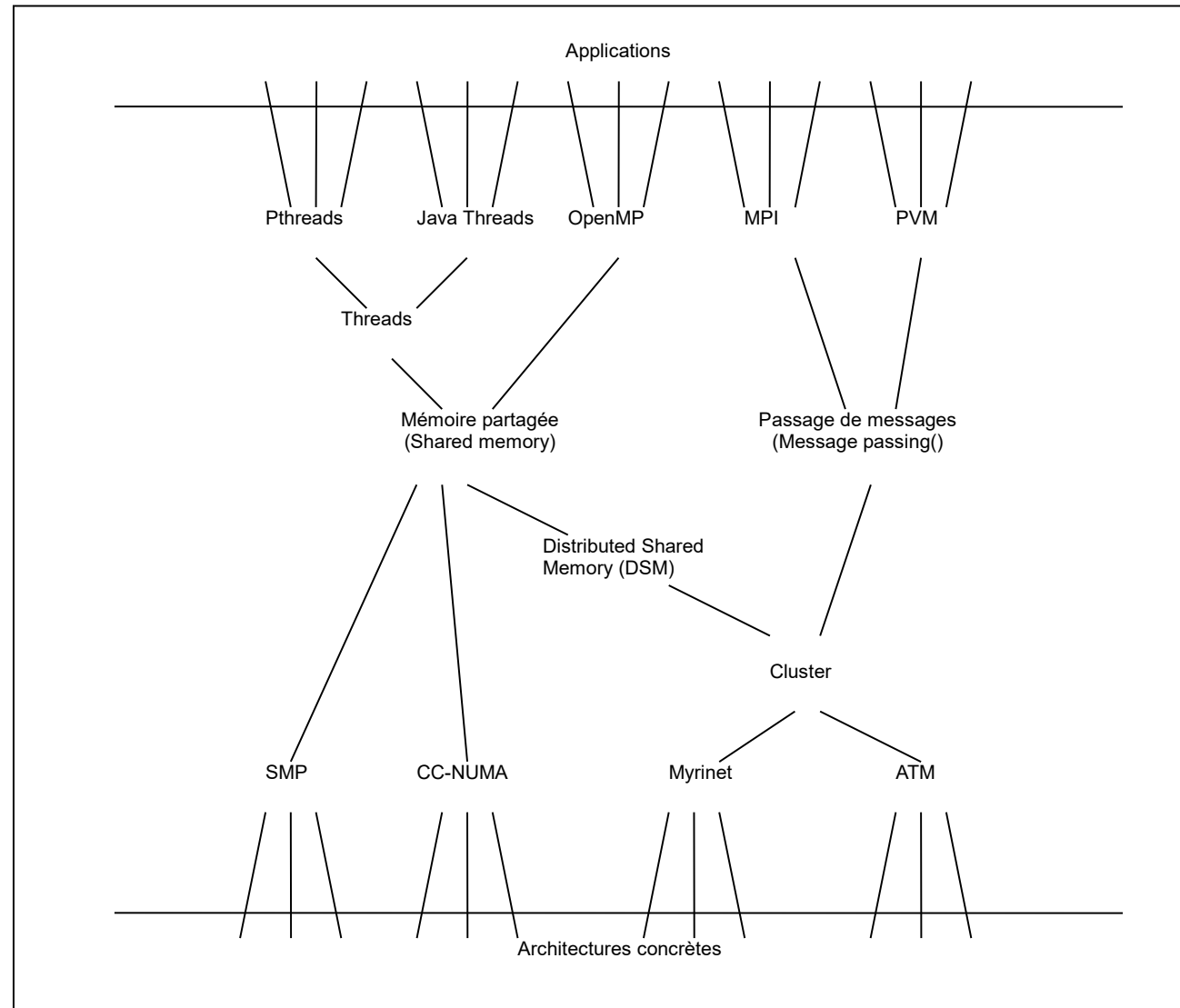
# Bref historique du parallélisme (suite)

- 1990 - 2000 : Machines basées sur le regroupement de PC répandus commercialement (Beowulf)
- 2000 - ... : SMP, clusters, CC-NUMA, Grilles de calcul
- Depuis quelques années
  - Processeurs multi-core
  - Processeurs graphiques
  - Cloud computing

# Le parallélisme aujourd'hui

- Constats
  - Le parallélisme devient de plus en plus accessible
  - Les environnements de développement deviennent de plus en plus conviviaux
- Cependant...
  - Il reste du travail à faire
    - Besoin d'intelligence !
    - Besoin de solutions logicielles performantes et portables

# Vue d'ensemble du calcul parallèle du début des années 2000



# Enjeux importants du parallélisme

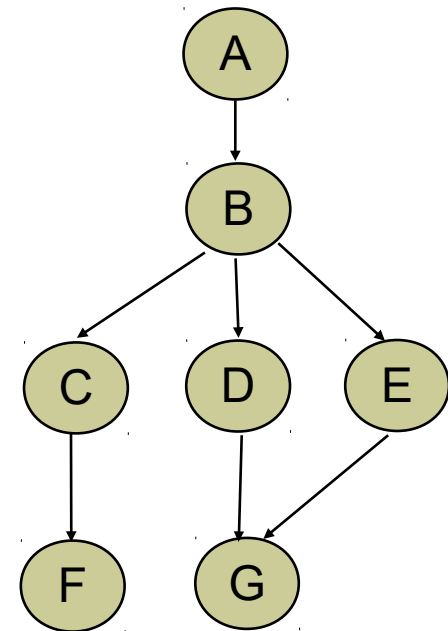
- Situation idéale : transparence
  - Le système dégage le parallélisme à partir d'un programme séquentiel
  - Idéal, mais très difficile
- Situation à éviter : multiplicité inutile
  - Coûts de conception/développement importants
- Compromis nécessaire entre la facilité de conception/développement et la performance
- Favoriser la portabilité des applications

# À la recherche du parallélisme...

Graphe de dépendance  
Parallélisme de données  
Parallélisme de tâches  
Parallélisme en pipeline

# Graphe de dépendance

- Permet de représenter visuellement le parallélisme d'un algorithme (exemple)
- Sommet = tâche
- Arc = Relation de dépendance



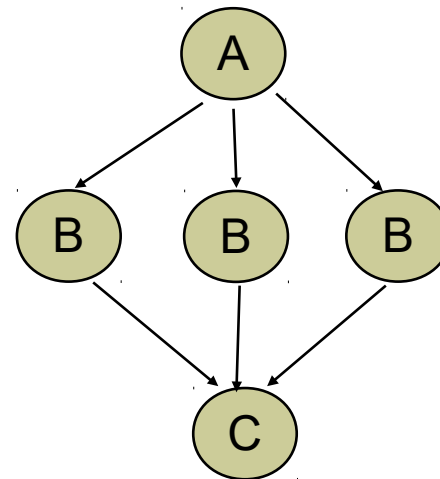


# Parallélisme de données

- Présent lorsque des tâches indépendantes appliquent la même opération sur différents éléments d'un ensemble de données

Pour  $i = 1$  à 100  
Lire ( $b[i]$ ,  $c[i]$ )  
Pour  $i = 1$  à 100  
 $a[i] = b[i] + c[i]$   
Pour  $i = 1$  à 100  
Écrire ( $b[i]$ ,  $c[i]$ )

A  
B  
C



# Parallélisme de tâches

- Présent lorsque des tâches indépendantes appliquent des opérations différentes sur des données (différentes ou non)

$$a = 2$$

$$b = 3$$

$$c = 3$$

$$m = (a + c) / 2$$

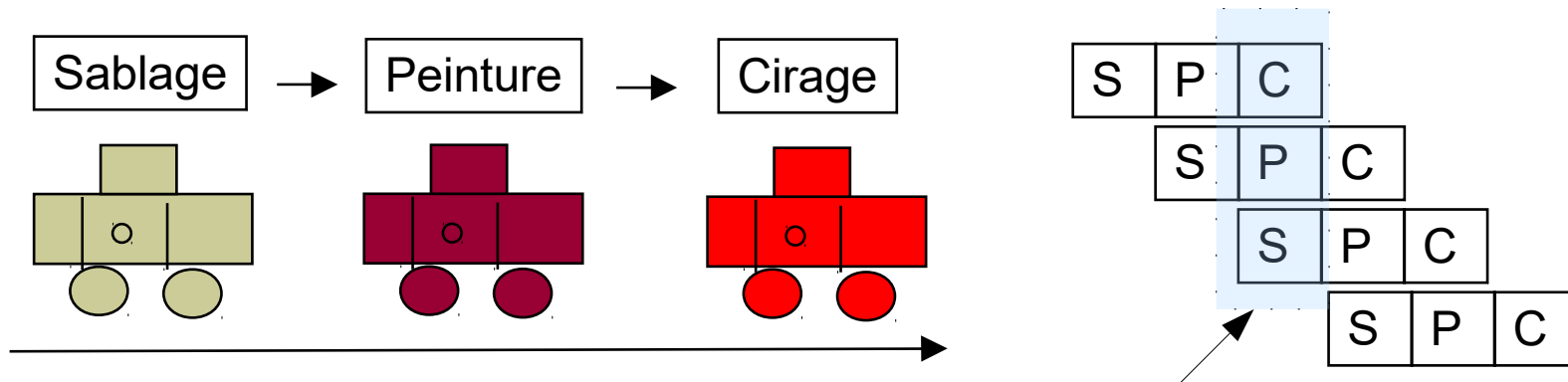
$$s = (a^2 + b^2) / 2$$

$$v = s - m^2$$

Graphe de  
dépendance ?

# Parallélisme en pipeline

- Diviser un algorithme strictement séquentiel en phases
- Résoudre plusieurs instances de problèmes à la chaîne
- La sortie (output) d'une phase représente l'entrée (input) de la phase suivante



# Architectures parallèles

Multi-ordinateurs  
Multiprocesseurs

# Introduction

- 1960-1990 : beaucoup de développement dans les architectures, mais trop de variété
  - Processeurs : fabrication propriétaire vs. Générique
  - Petit nombre de processeurs super-puissants vs. grand nombre de processeurs « ordinaires »

# Introduction

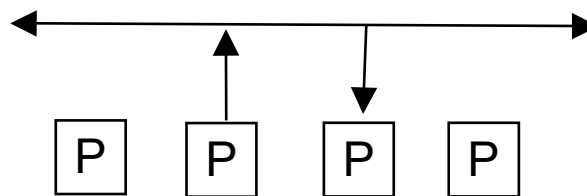
- Aujourd'hui :
  - Utilisation de CPU génériques, croissance plus rapide que les technologies propriétaires
- Questions
  - Comment relier ces processeurs les uns aux autres?
  - Comment relier ces processeurs à la mémoire ?

# Réseau d'interconnexion

- Sert à :
  - Relier les processeurs à une mémoire partagée
  - Relier les processeurs les uns aux autres
- Le médium du réseau peut être :
  - Partagé
  - Commuté
- Problématiques :
  - Vitesse de communication vs. vitesse des processeurs

# Médium partagé

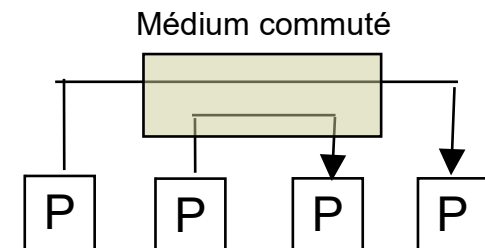
- Les messages sont envoyés à tous, un à la fois
- Tous les processeurs reçoivent le message mais seuls ceux identifiés comme destinataires le conservent
- Exemple : Ethernet
- Beaucoup de communications = collisions





# Médium commuté

- Permet des communications point-à-point entre paires de processeurs
- Chaque processeur a un lien avec le médium
- Principaux avantages sur le médium partagé
  - Permet des communications simultanées
  - Plus extensible à de grand nombre de processeurs
- Principaux inconvénients :
  - Complexité
  - Coût



# Topologie du réseau d'interconnexion

- Directe : 1 processeur = 1 commutateur
- Indirecte : Nb. procs < Nb. Commutateurs
- Critères de performance du réseau :
  - Diamètre : Distance maximum entre 2 nœuds
  - Largeur bisectionnelle : Nombre min. de liens devant être enlevés pour sectionner le réseau en deux moitiés
  - Degré : Nb liens directs d'un nœud vers d'autres nœuds
  - Longueur des liens entre les noeuds

# Topologie du réseau d'interconnexion

- Plusieurs types de topologies
  - Grille
  - Arbre binaire
  - Arbre hyperbolique
  - Papillon
  - Hypercube
  - Multi-étage
- Aucune topologie n'est supérieure sur tous les critères

# Le besoin d'architectures flexibles

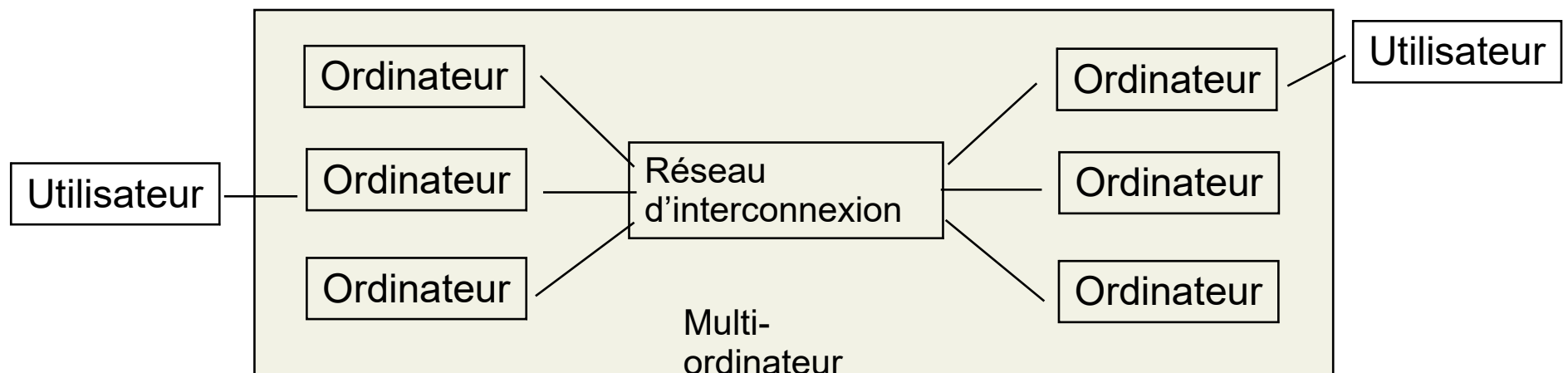
- Historiquement, les ordinateurs parallèles manquaient de flexibilité
  - Ordinateurs vectoriels limités principalement au parallélisme de données
  - Difficulté de programmation, nécessité de tenir compte du réseau d'interconnexion
- 2 types d'ordinateurs mieux adaptés aux besoins actuels se sont développés depuis les années 90
  - Multiprocesseurs
  - Multi-ordinateurs

# Multi-ordinateur

- La mémoire est physiquement distribuée entre les processeurs
- Logiquement, espaces d'adressage disjoints
- Chaque processeur n'a accès qu'à sa mémoire locale
- Interactions entre processeurs effectuées par passage de messages
- Clusters, réseaux de stations, ...

# Multi-ordinateur

- La machine parallèle est composée de plusieurs ordinateurs possédant
  - Un système d'exploitation
  - Des programmes
  - Un point d'entrée



# Clusters vs. Réseaux de stations

- Réseau de stations (NOW) :
  - Ensemble hétérogène d'ordinateurs
  - Souvent réparti géographiquement
  - Les nœuds sont souvent déjà utilisés localement
  - Calcul parallèle = Utilisation secondaire
  - Connectés par un réseau générique (Ethernet)
- Cluster
  - Ensemble homogène d'ordinateurs
  - Réunis géographiquement
  - Nœuds dédiés au calcul parallèle
  - Ressource unifiée de calcul parallèle
  - Réseau haut débit (Fast/Gigabit Ethernet, Myrinet)

# Clusters vs. Réseaux de stations

- Comparaison des réseaux

	Latence (micro-sec)	Largeur de bande	Coût par nœud (\$)
Ethernet	150?	10	?
Fast Ethernet	100	100	100
Gigabit Ethernet	100	1000	1000
Myrinet	7	1920	2000

- Avantage des NOW : Peu coûteux et facile à construire avec des composants génériques

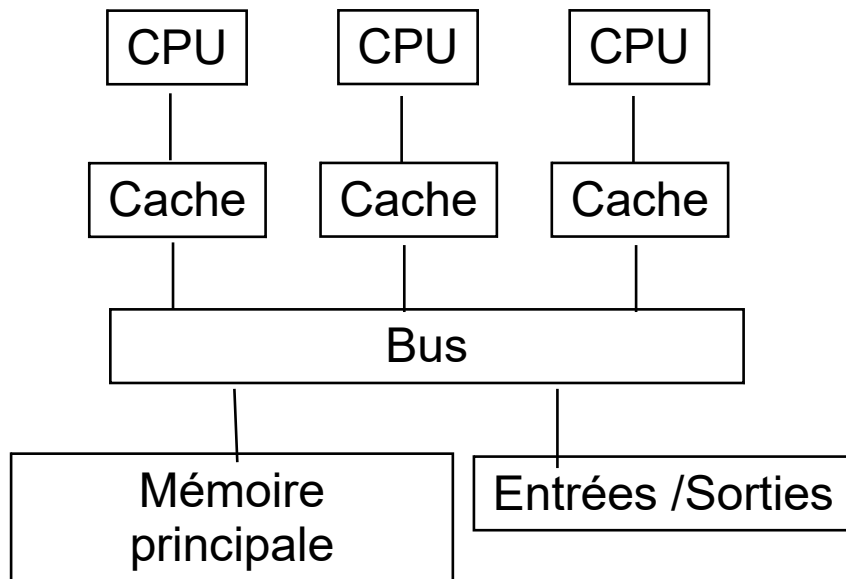


# Multiprocesseur

- Ordinateur comprenant plusieurs processeurs
- Mémoire partagée
- Les caches réduisent la charge sur le bus et/ou la mémoire
- Les processeurs communiquent par des lectures/écritures sur des variables en mémoire partagée (physiquement ou virtuellement)
- SMP, Multi-core, ...

# 2 types de multiprocesseur

## Centralisé

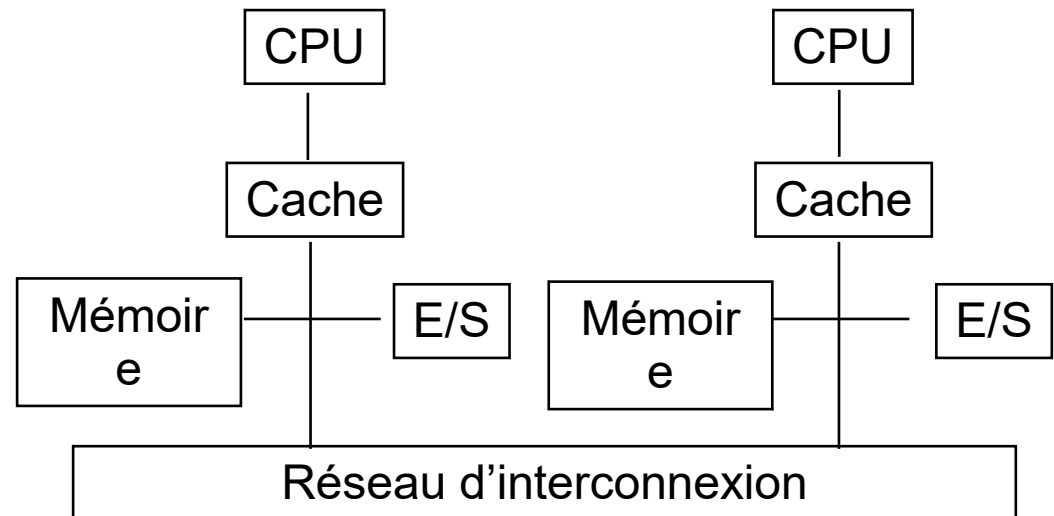


## Partage physique

- 2 problèmes

- Cohérence de cache, synchronisation

## Distribué



## Partage Virtuel

# Cohérence de cache

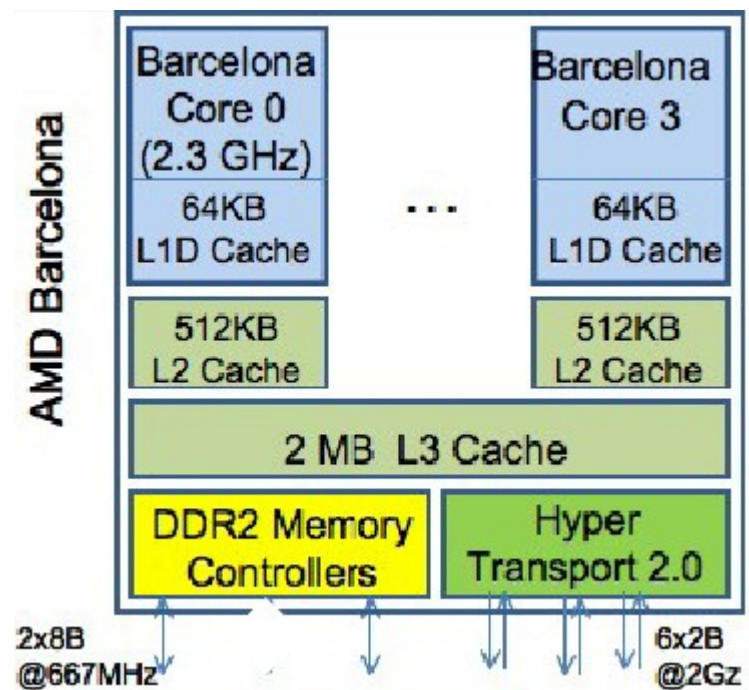
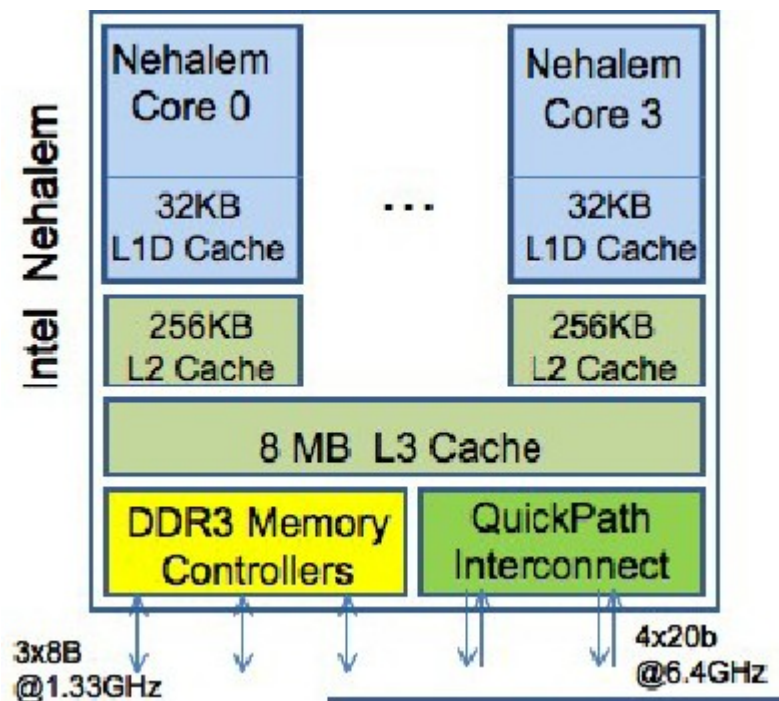
- La duplication de certaines données dans les caches permet de minimiser les accès mémoire
- Que fait-on lorsqu'une écriture est effectuée?
- Protocole de cohérence de cache : ensemble de règles assurant que les processeurs possèdent la même valeur d'un emplacement mémoire
- Espionnage (Snooping) / write invalidate

# Architecture multi-core

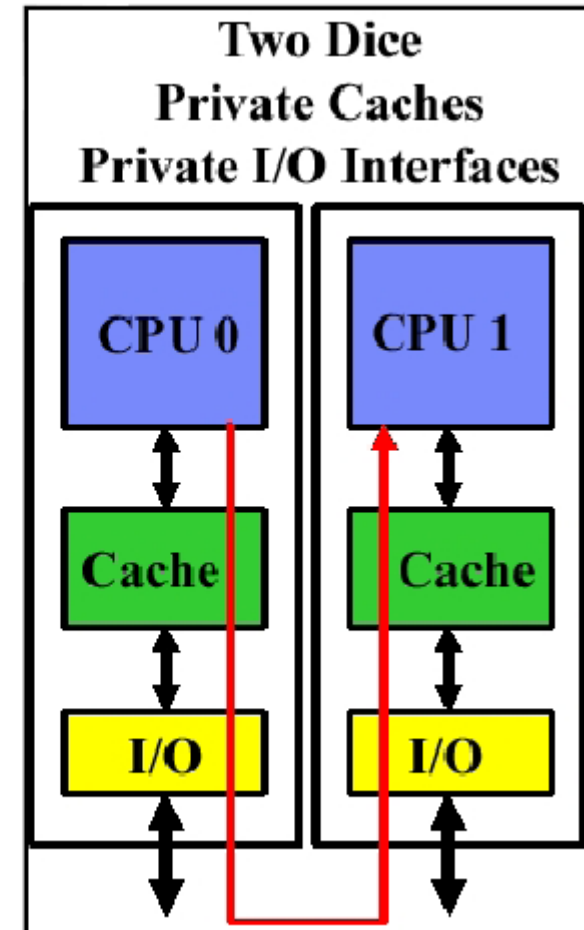
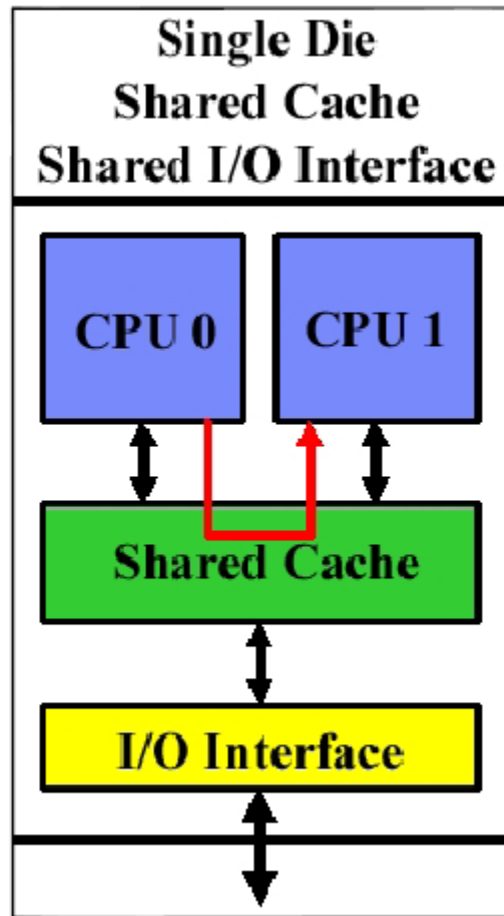
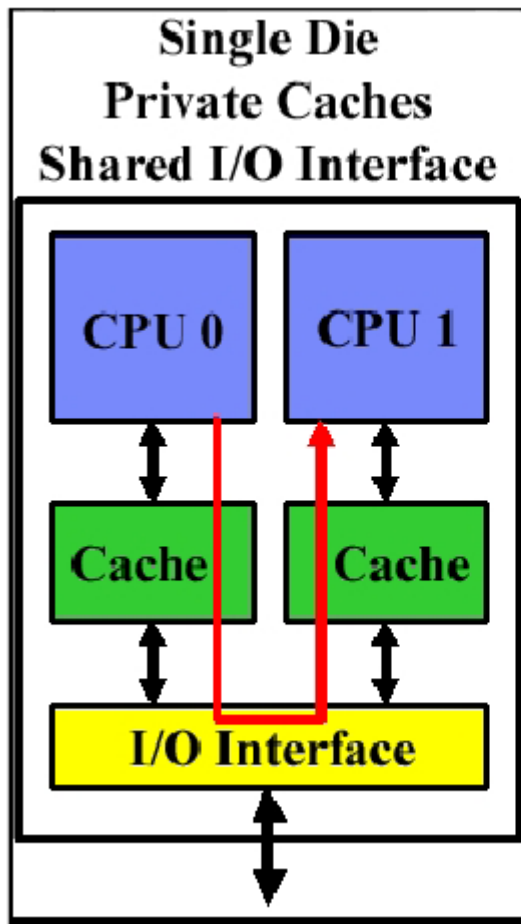
- Processeur → au moins 2 unités de calcul
- Augmentation de la puissance de calcul sans augmentation de la fréquence d'horloge
  - Réduction de la dissipation thermique
- Augmentation de la densité
  - Comme les coeurs sont sur le même support, la connectique reliant à la carte mère ne change pas

# Exemples de processeurs multi-core

- Architectures Quad-Core



# Architecture des caches



# Partage des caches

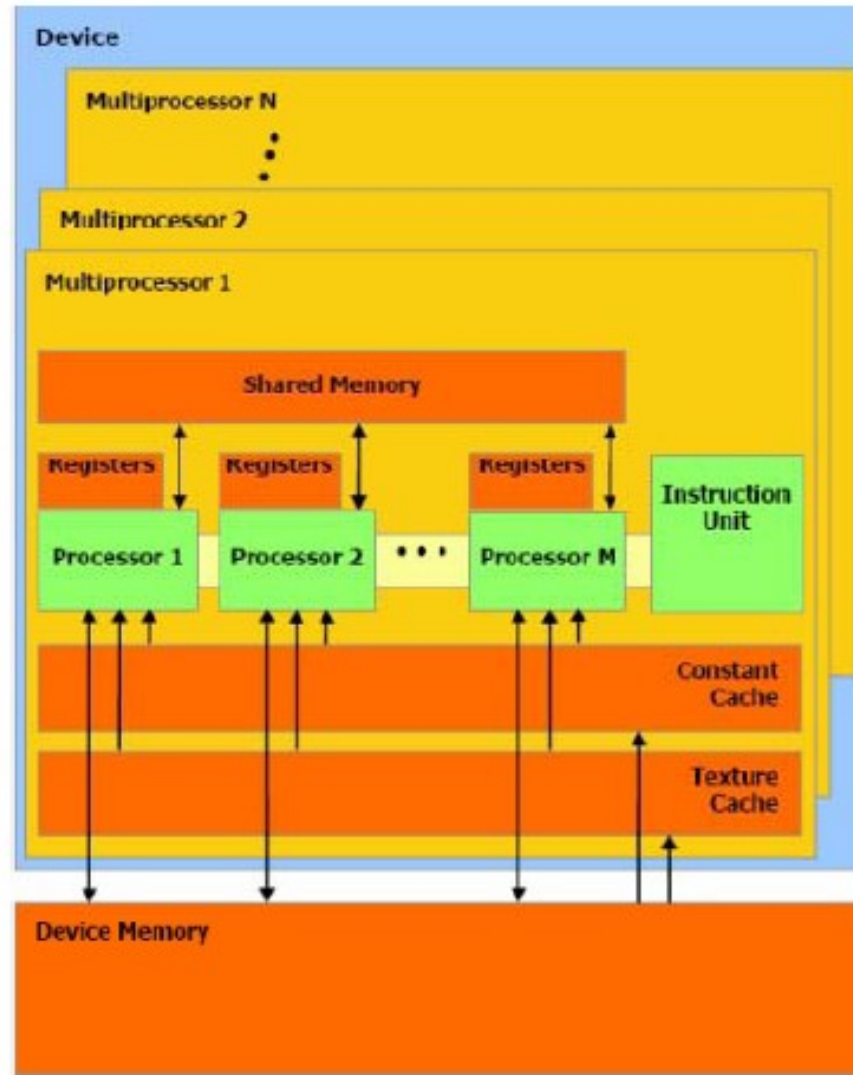
- Pas de partage de caches
  - Pas de contention entre les unités de calcul
  - Communication et migration plus coûteuses
    - Passage systématique par la mémoire centrale
- Partage du cache
  - Communication plus rapide entre unités de calcul
  - Migration plus facile → on ne passe pas par la mémoire centrale
  - Problème de contention au niveau des caches
  - Problème de cohérence de cache !

# Émergence d'une nouvelle architecture : les GPU

- GPU : Graphic Processor Unit
- Architecture massivement parallèle
- Traditionnellement utilisé pour le calcul graphique
- On commence à les utiliser pour le calcul « générique »
  - API CUDA → Nvidia
  - API ATI Stream → ATI
  - API générique → OpenCL



# Architecture GeForce 8800



# Synchronisation

- Dans certains cas d'exécution parallèle, des règles de précedence doivent être respectées
- Dans un contexte de mémoire partagée, des mécanismes permettent d'assurer un certain ordre dans les opérations
  - Exclusion mutuelle : seulement un processeur doit effectuer une partie de programme
  - Section critique : partie de programme ne pouvant être effectuée que par un processeur à la fois
  - Barrière de synchronisation : les processeurs sont tous arrêtés à un certain point du programme

# Taille des ordinateurs parallèles

- « Petits » ordinateurs parallèles
  - $2 < \text{Nombre de processeurs} < 64$
  - Typiquement de type Multiprocesseurs (SMP et multi-core)
  - Vue globale de la mémoire, cohérence de cache
- « Grands ordinateurs parallèles »
  - $64 < \text{Nombre de processeurs} < \text{plusieurs centaines}$
  - Typiquement de type Multi-ordinateur
  - Souvent des clusters de SMP

# Romeo I (2002)

- SMP 24 X UltraSparc III 900 Mhz
- 24 Go RAM
- 210 Go d'espace disque



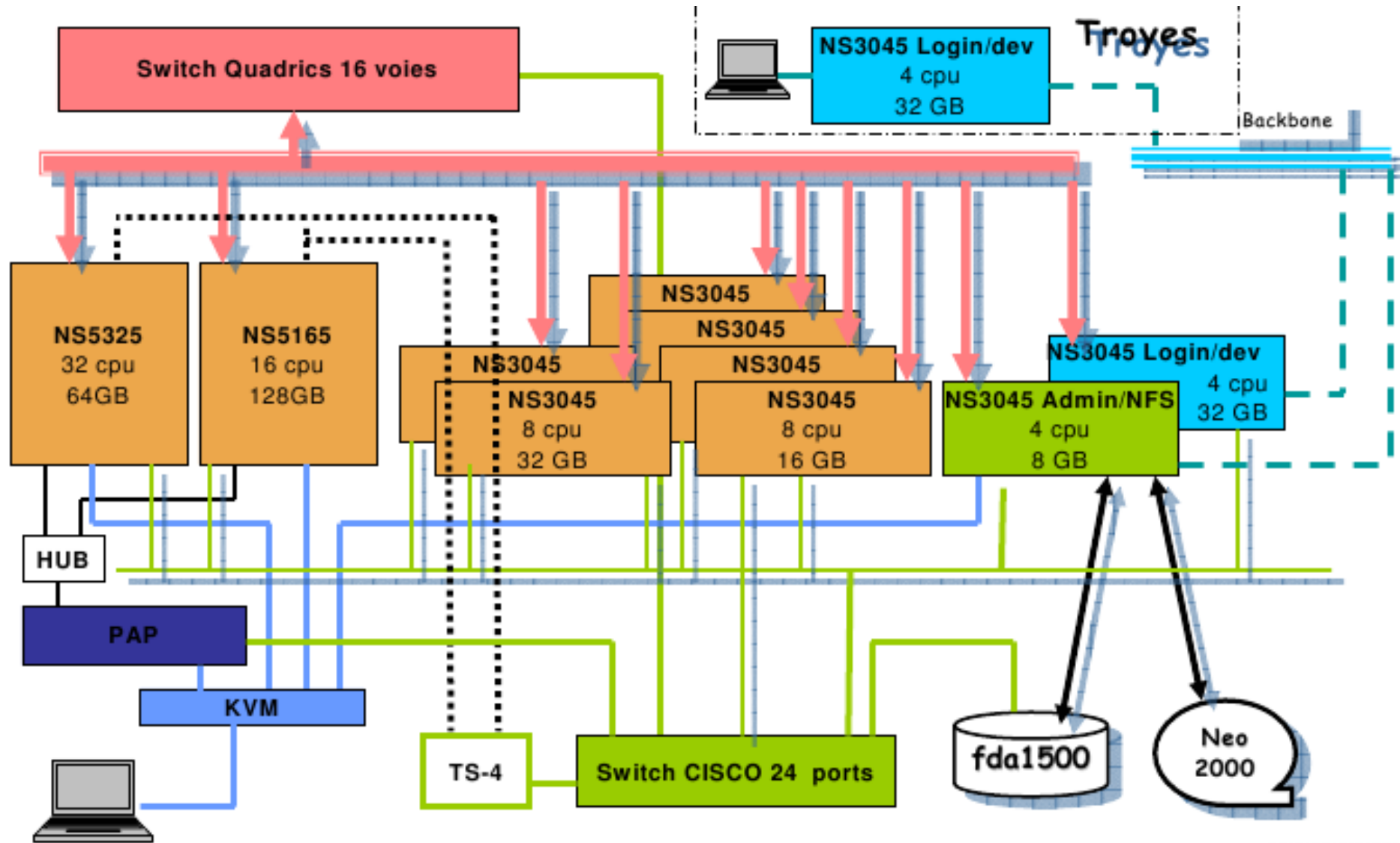
# Romeo II (2007)

- Centre de calcul régional Champagne-Ardenne
- 8 noeuds de calcul
  - Noeuds SMP de 4 à 16 processeurs
  - Procs 1.6 Ghz Dual-core (8 à 32 coeurs par noeud)
  - Entre 16 Go et 128 Go RAM par noeud
- 96 coeurs et 400 Go de mémoire au total
- 20 disques de 146 Go chacun (2.9 To)
- Coût ?
  - Matériel, locaux, climatisation, support, ..., ..., ...

# Romeo II (2007)



# Architecture Romeo II



# Clovis – Romeo III (2010)

- 41 noeuds de calcul
  - 39 noeuds : 12 coeurs – 24 Go RAM
  - 2 noeuds : 32 coeurs – 64 Go RAM
  - Processeurs 2.6 Ghz Quad-core
- 532 coeurs et 1,064 To de mémoire au total
- 10 To de disque
- 1 noeud dédié GPGPU (2 cartes Nvidia C2050)



# Clovis – Romeo III (2010)



# Romeo IV ?

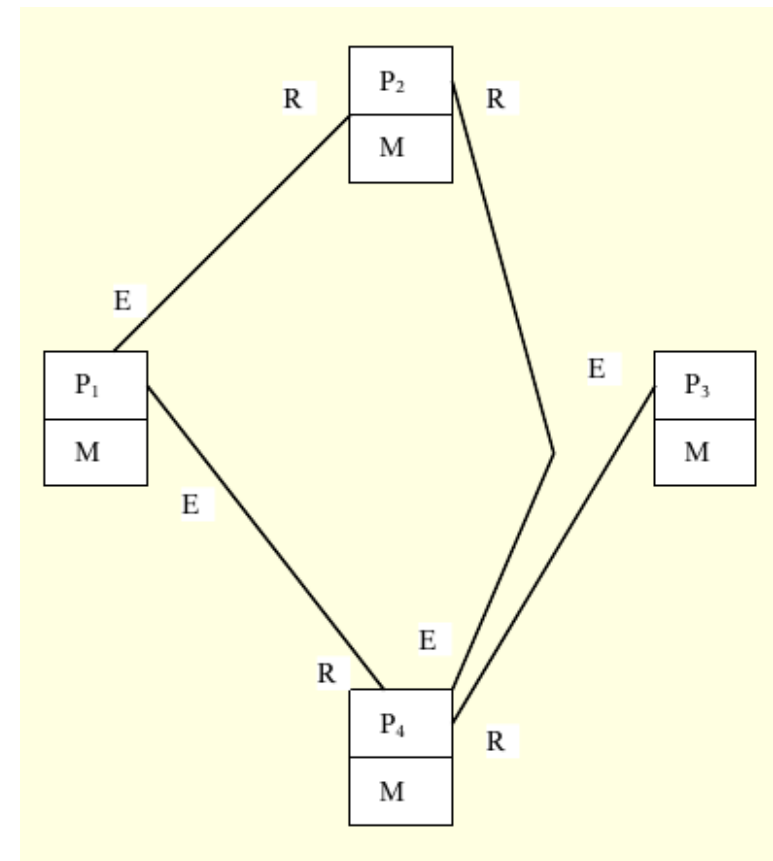
- Objectif 2013
  - Top 500 : <http://www.top500.org/>

# Programmation parallèle

Mémoire distribuée  
Mémoire partagée

# Programmation mémoire distribuée

- Modèle à passage de messages
  - Processeurs attribués à des processus distincts
  - Chaque processeur possède sa propre mémoire
  - Communications par envois et réception de messages
- Languages
  - PVM, MPI



# Hello World ! en C/MPI

```
#include <stdio.h>
#include <mpi.h>

int main (int argc, char *argv[]) {

    int id, taille;

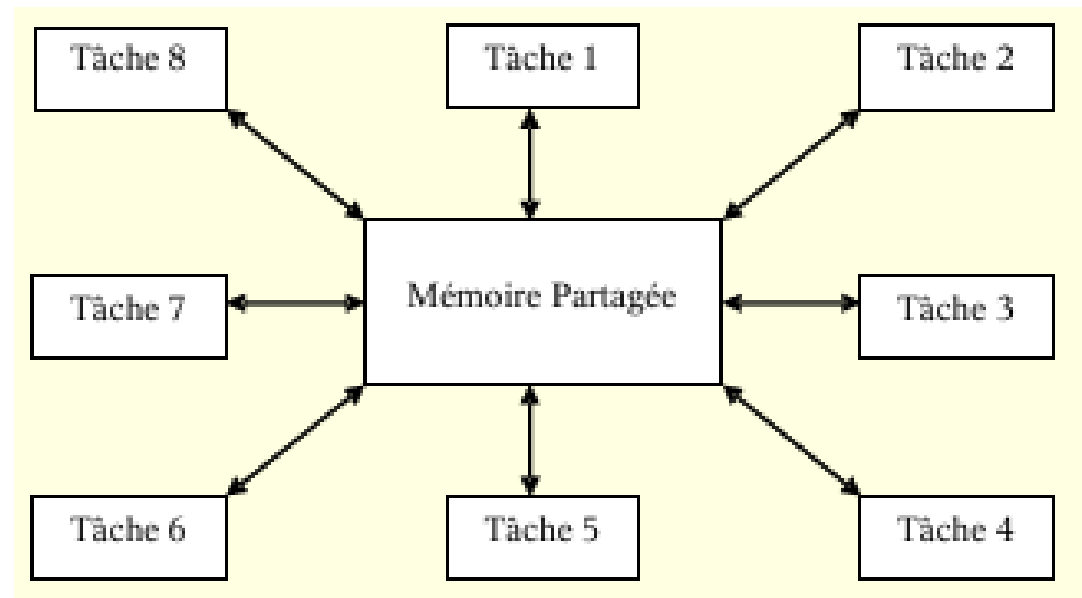
    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &id);
    MPI_Comm_size (MPI_COMM_WORLD, &taille);

    printf( "Hello world du processus %d de %d\n", id, taille );

    MPI_Finalize();
    return 0;
}
```

# Programmation mémoire partagée

- Modèle à mémoire partagée
  - Processus attribués à des processeurs distincts
  - Mémoire partagée (physiquement ou virtuellement)
  - Communications par lectures/écritures dans la mémoire partagée
- Langages
  - Pthreads
  - Java Threads
  - OpenMP



# Hello World ! En pthreads

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

void *helloThread(void *threadId) {
    int *tid;

    tid = (int *) threadId;
    printf("Hello World! du thread #%d !\n", *tid);
    pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
    pthread_t threads[5];
    int statut, i *t;

    for (i = 0; i < 5; i++) {
        t = (int *) malloc(sizeof(int));
        *t = i;
        printf("Dans le main: creation du thread %d\n", *t);
        statut = pthread_create(&threads[i], NULL, helloThread, (void *) t);
        if (statut != 0) {
            fprintf(stderr, "Probleme creation thread, code %d \n", statut);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

# Hello world ! en Java Threads

```
public class HelloThread extends Thread {  
  
    int id;  
  
    HelloThread (int id) {  
        this.id = id;  
    }  
  
    public void run () {  
        System.out.println ("Hello world du thread " + id + " !");  
    }  
  
    public static void main (String args [] ) {  
        for (int i = 0; i < 5; i++) {  
            (new HelloThread(i)).start();  
        }  
    }  
}
```



# Hello World ! En C/OpenMP

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char *argv[]) {
    int id;

    omp_set_num_threads(5);
    #pragma omp parallel private (id)
    {
        id = omp_get_thread_num();
        printf("Hello, world, du thread %d !\n", id);
    }
    return 0;
}
```

# Conclusion

- Les différents types d'ordinateurs ne se programment pas tous de la même façon
- Prochain cours : Conception d'algorithmes en général
- Cours suivants : Algorithmique et programmation sur Multiprocesseurs/Multi-ordinateurs