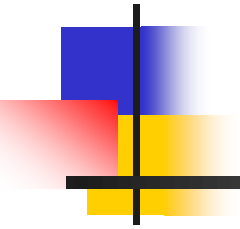
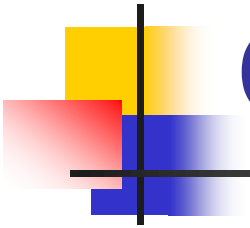


Programmation Parallèle

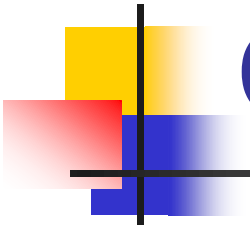
Julien Jaeger

julien.jaeger@cea.fr



Contexte

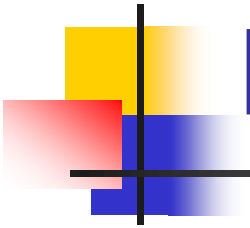
- Evolution des architectures de processeurs ?
  - Augmentation de la fréquence
    - Ex : 1GHz → 1 milliard de changement d'horloge par seconde
  - Mais : limite physique
    - Consommation électrique
    - Dissipation de la chaleur
    - taille de gravure vs. taille d'un électron
- Solution :
  - Le parallélisme est la solution pour augmenter la puissance des processeurs
  - Plusieurs pistes…



Contexte

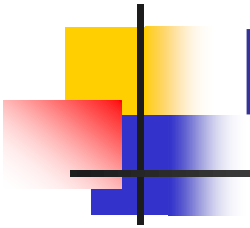
- Parallélisme
  - Déjà existant au sein d'un processeur (pipeline, traitement de plusieurs instructions, exécution Out-Of-Order, …)
  - Multiplication des unités de traitements
    - Augmentation du nombre de coeurs
    - Duplication des unités vectorielles

- De nombreux domaines utilisent des ordinateurs "massivement parallèle" (calcul haute performance) :
  - simulation numérique (Industries aéronautique, automobile, nucléaire, Météorologie…) ;
  - infographie : films d'animation ;
  - traitement d'image (Photoshop) ;



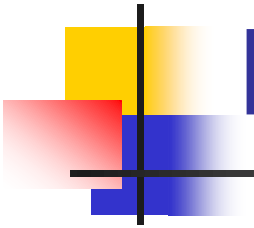
But de ce cours

- Comprendre le parallélisme
  - Description d'une architecture de processeur/nœud de calcul
  - Découverte des types de parallélisme
- Apprendre à exploiter le parallélisme d'un code
  - Trouver le parallélisme
  - Connaître les modèles de programmation
- Résumé
  - "Comprendre le parallélisme et savoir programmer des applications parallèles est un atout majeur"



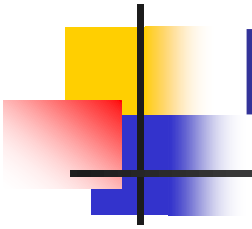
Déroulement du module

- Prérequis
  - Système d'exploitation : Linux
  - Langage de programmation : C
    - Maîtrise arithmétique pointeur demandée
- Travail en salle machine
  - Programmation en parallèle
- Evaluation des connaissances
  - Partiels & TPs



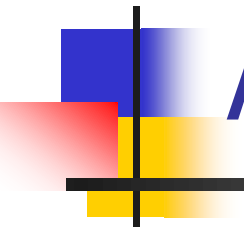
Plan

- Introduction
  - Architectures et programmation
- Programmation mémoire distribuée
  - Modèle MPI *(Message-Passing Interface)*
- Programmation mémoire partagée
  - Modèle *thread*
  - Modèle OpenMP
- Vers des modèles hybrides et hétérogènes



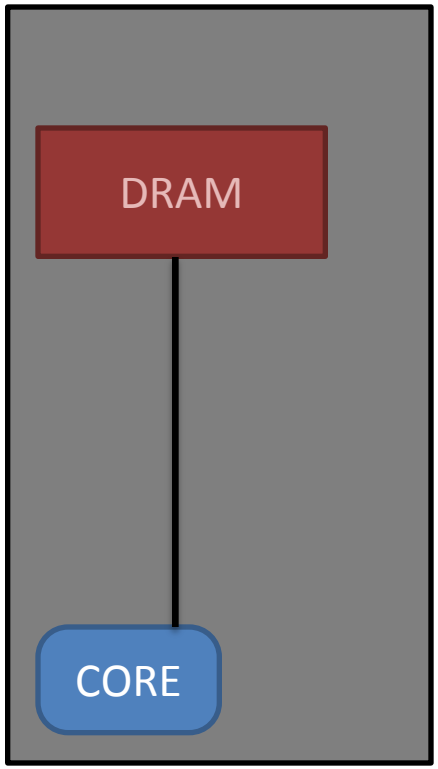
Plan du cours 1

- Architecture des machines parallèles
  - Système à mémoire partagée
  - Système à mémoire distribuée
  - Supercalculateurs
- Introduction à la programmation parallèle
  - Notions et définitions
  - Types de parallélisme
  - Modèles de programmation



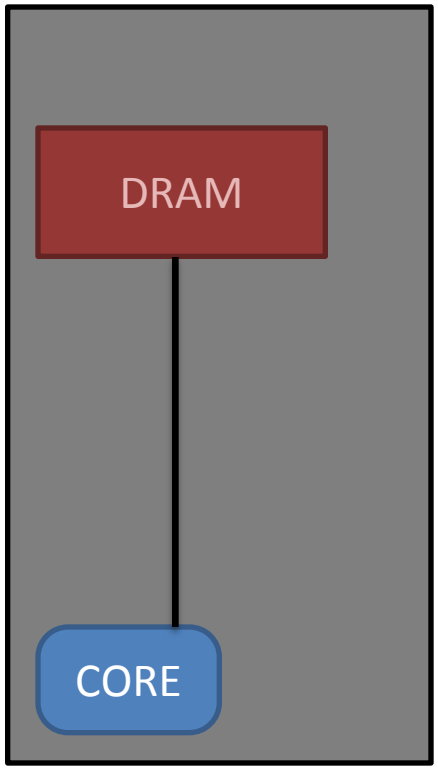
Architectures des calculateurs

Au commencement…



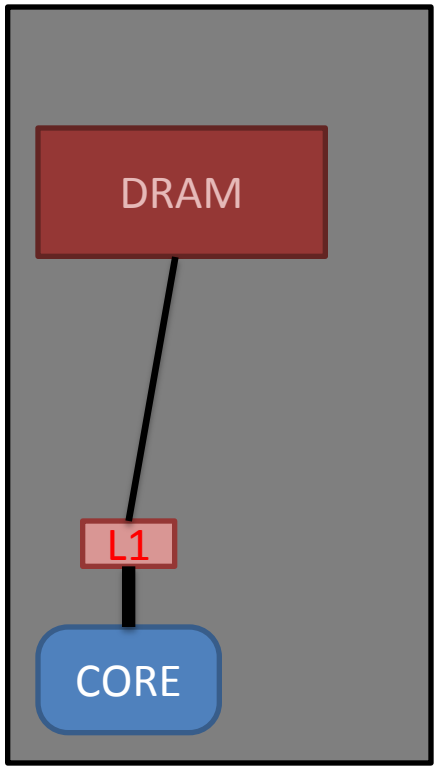
- Il y a un cœur de calcul (ALU) réalisant les opérations…
  - Arithmétiques
  - Logiques
- …et une mémoire pour stocker les données

Problème – memory wall



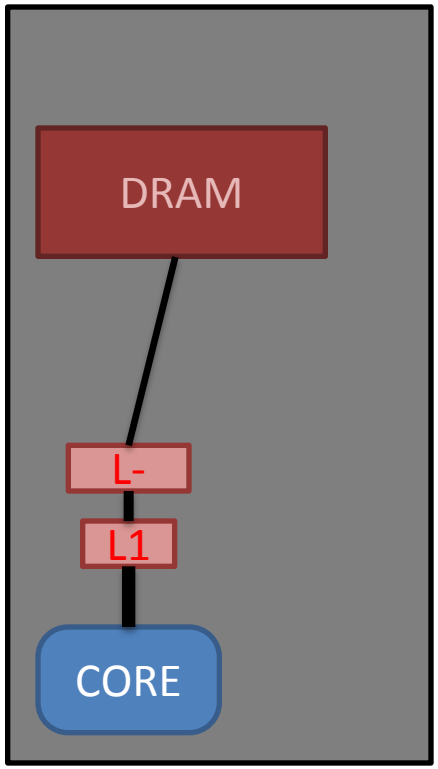
- Perfomance du calcul augmente plus rapidement que celle de la mémoire
- Il faut trouver un moyen de «nourrir» les unités de calcul
  - Sinon, il n'est pas possible d'obtenir la performance max

Alors arriva le cache



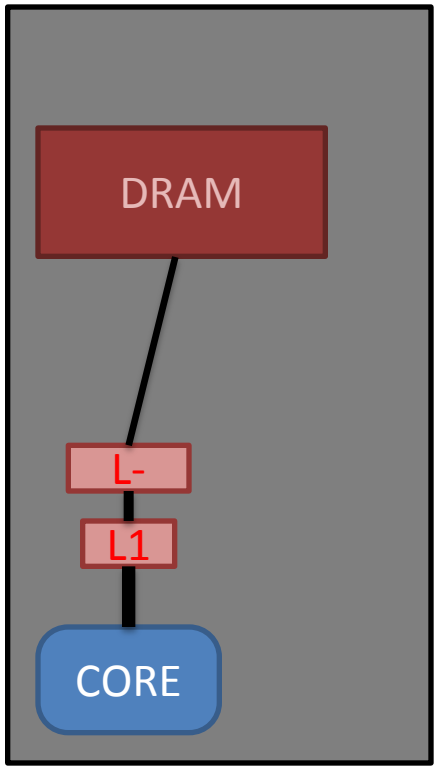
- Ajout d'un cache de données
- Une mémoire beaucoup plus performante et plus proche du cœur
  - Meilleure latence
  - Meilleure bande passante
- Sorte de «tampon» mémoire à côté du coeur

HPC vu que c'était bien!



- Un seul niveau de cache pas suffisant
  - Difficile de nourrir le cache
  - Taille trop petite pour garder les données utilisées
- Ajout d'autres niveaux de caches
  - Dépend de l'architecture (2 ou 3 généralement)

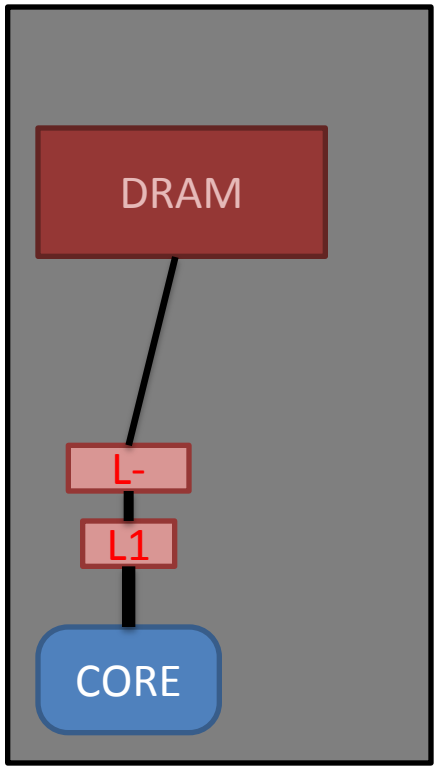
Scotty, we need more (compute) power



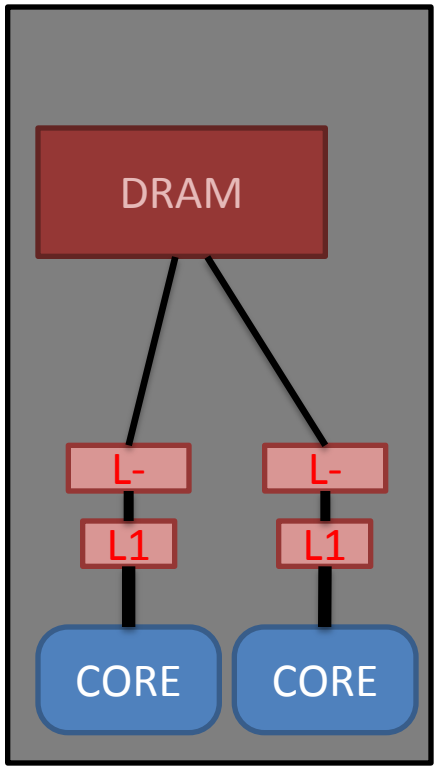
- Chaque génération de supercalculateur à pour but de fournir plus de puissance de calcul
- Généralement atteint avec l'augmentation de la fréquence

Problème – heating wall

- L'augmentation de la fréquence s'obtient avec l'augmentation du nombre de transistors
  - Et leur diminution en taille
- -> Densité toujours plus grande de transistors
  - Impossible de dissiper la chaleur

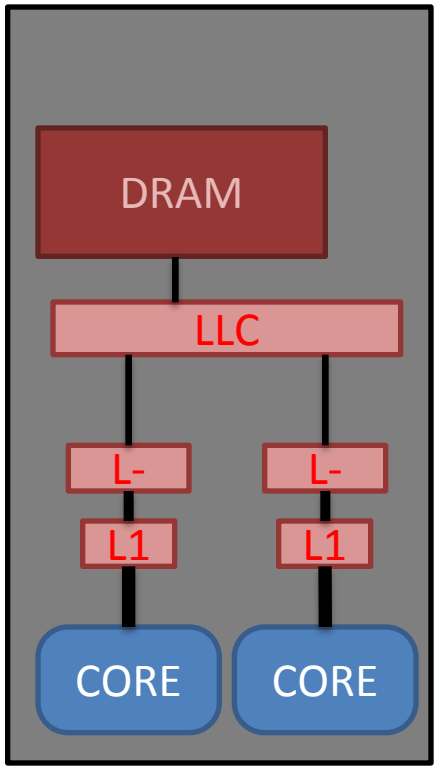


La multiplication des cœurs



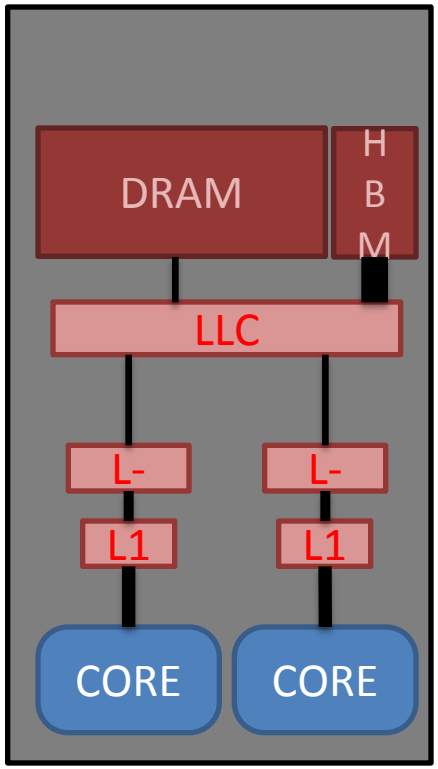
- Plutôt que d'augmenter le nombre de transistors par cœurs…
- … Augmentons le nombre de cœurs de calcul par processeur!
- Permet d'augmenter la quantité de calcul par cœur en limitant la densité

Les caches, c'est (très) bien!



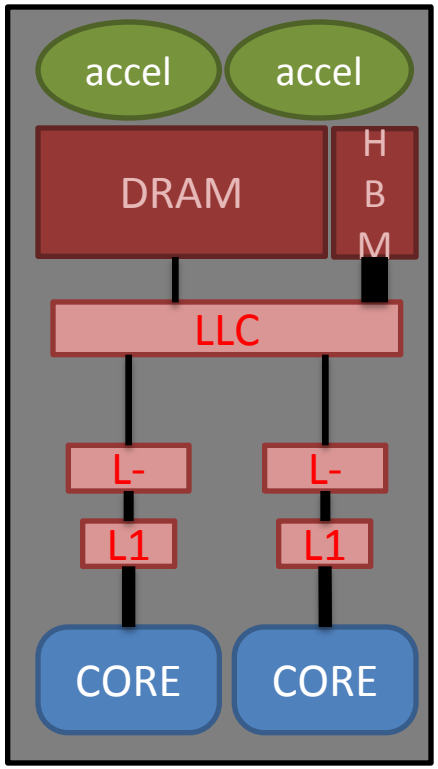
- Le partage des données entre les cœurs ne peut se faire qu'à travers la mémoire globale.
  - Peu efficace si les cœurs doivent utiliser les mêmes données
- Cache de dernier niveau (LLC) partagé entre les cœurs
  - Il est possible d'avoir aussi des caches intermédiaires partagés

Toujours plus de données



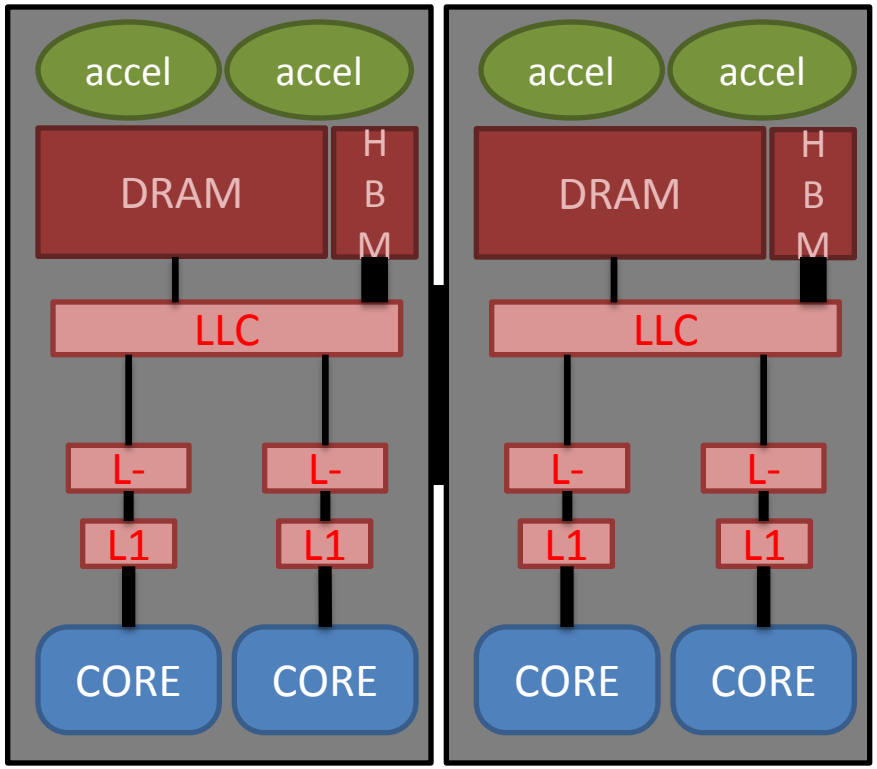
- L'augmentation toujours continuelle de la puissance de calcul entraîne une demande toujours croissante en terme de transferts de données
- Ajout, en plus de la mémoire classique, d'une mémoire à forte bande passante

Toujours plus de calcul



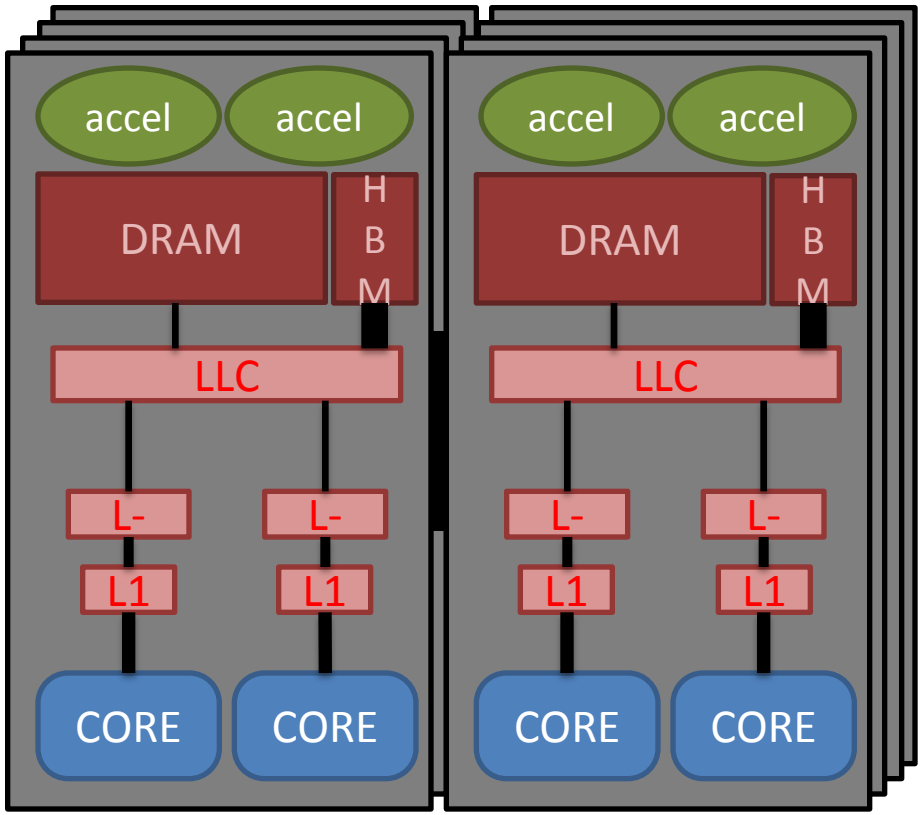
- Augmentation du nombre de cœurs demande plus de puissance électrique
  - Besoin de fournir de la puissance de calcul à moindre puissance
- Des accélérateurs sont rattachés aux nœuds de calcul

Multiplication des sockets



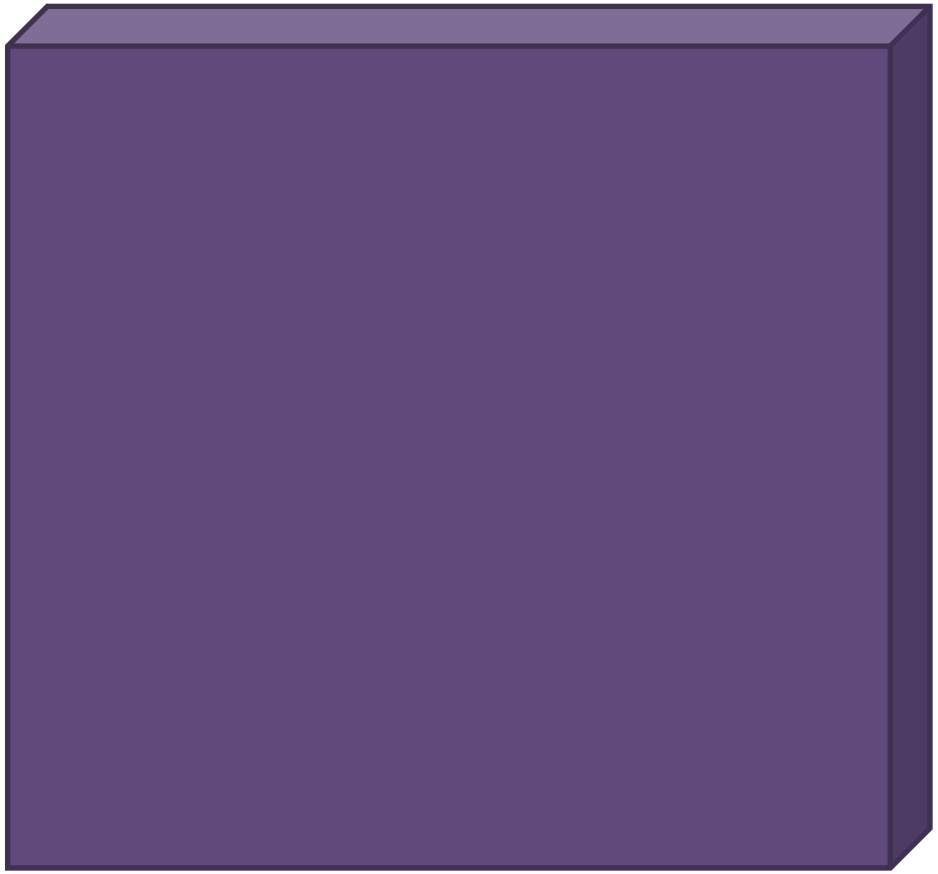
- Difficile d'étendre indéfiniment un tel ensemble d'éléments
- Par contre, possible de multiplier ces ensemble et de les relier

Multiplication des nœuds



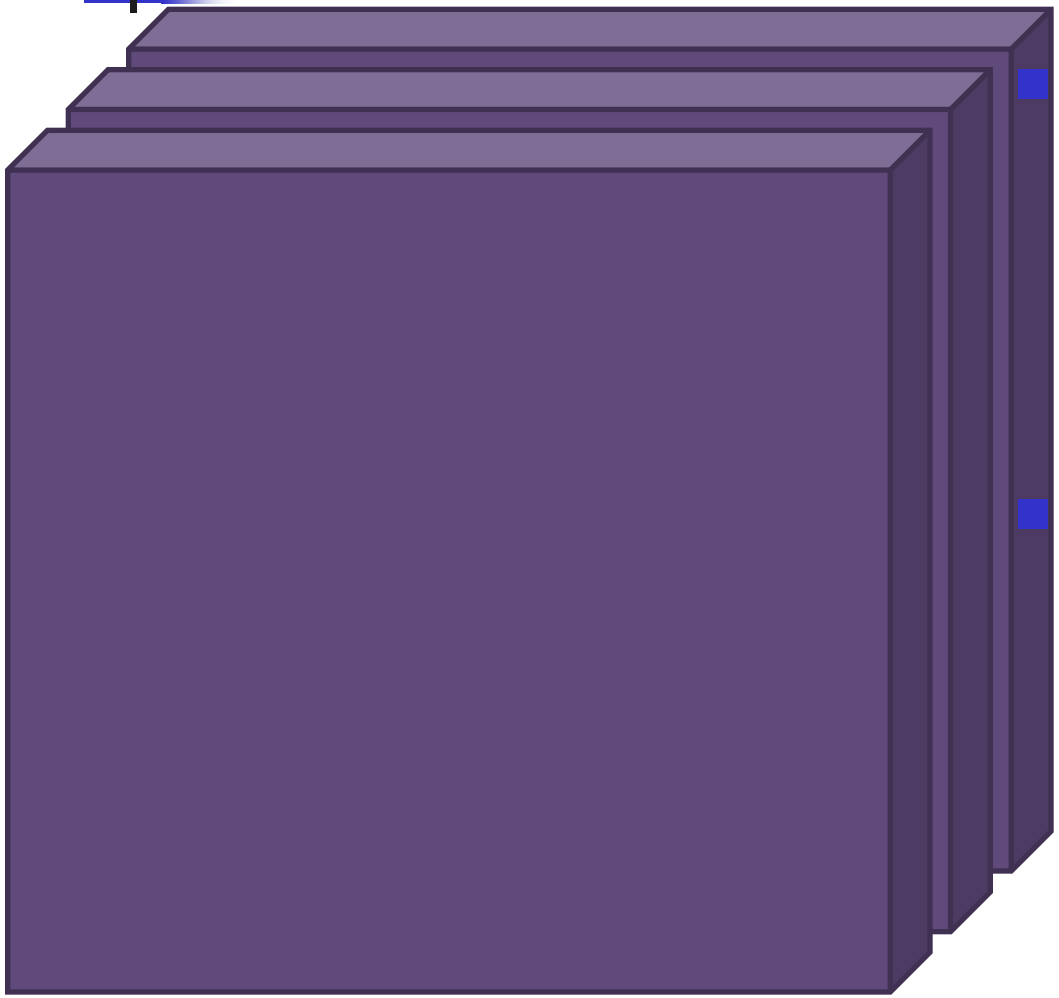
- De même, difficile de multiplier indéfiniment le nombre de sockets
- Par contre, possible de multiplier ces nœuds de calcul

Supercalculateurs



- Un ensemble de nœuds est rassemblé dans une armoire.

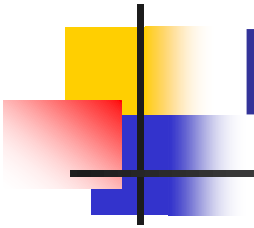
Supercalculateurs



Un ensemble de nœuds est rassemblé dans une armoire.

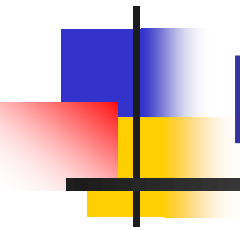
Un supercalculateur est composé d'armoires reliées entre elles

Exemple: Tera-1000

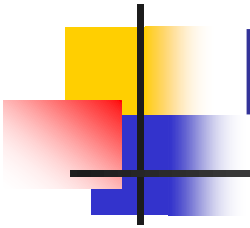


Résumé

- Architectures parallèles caractérisées par la topologie mémoire
  - Partagée, distribuée, partagée/distribuée;
- Aujourd'hui, le parallélisme est présent au sein même du processeur
  - Superscalaires
  - Multicoeurs
- Lors de l'écriture d'une application, il faut désormais penser parallélisme

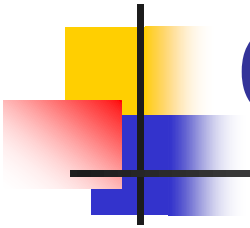


Introduction à la programmation parallèle



Définitions

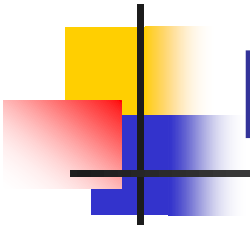
- Tâche
  - Travail à faire
- Thread (ou flot d'exécution)
  - Implémentation d'une tâche : suite logique séquentielle d'actions résultat de l'exécution d'un programme
- Processus
  - Instance d'un programme. Un processus est constitué d'un ou plusieurs threads qui partagent un espace d'adressage commun. Si un processus comporte plusieurs threads, il est dit multithread
- Calcul parallèle
  - Le calcul parallèle consiste en le découpage d'un programme en plusieurs tâches qui peuvent être exécutées en même temps dans le but d'améliorer le temps global d'exécution du programme



Qu'est-ce que le parallélisme ?

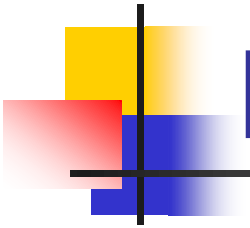
- Vieille idée pour résoudre plus vite un problème long et coûteux en temps calcul;
- Une solution : utiliser plusieurs unités de traitement (i.e. processeurs);
- Difficulté : organisation des tâches parallèles (algorithmique parallèle) :
  - résoudre correctement le problème initial : relation de dépendances entre tâches;
  - les unités de traitement doivent avoir constamment du travail (utile) à effectuer : distribution et équilibrage (dynamique) de la charge;

Programmation séquentielle et parallèle



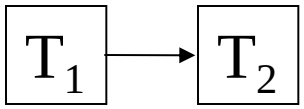
- Programmation séquentielle :
  - Suite ORDONNÉE d'instructions à exécuter pour résoudre le problème initial;
  - Sémantique séquentielle
    - Toute instruction ne peut commencer que lorsque la précédente est terminée et son résultat disponible ;
  - ORDRE TOTAL dans l'exécution des différentes instructions;

Programmation séquentielle et parallèle

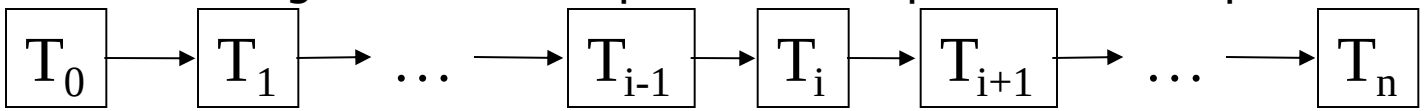


- Programmation parallèle :
  - Plusieurs flots d'exécution (instructions + données);
  - Plusieurs instructions exécutées simultanément;
  - Plusieurs processeurs (ou coeurs) ;
  - Met en évidence les **dépendances réelles** entre instructions :
    - la tâche T2 dépend de la tâche T1 **si et seulement si** T2 a besoin du résultat de T1 (pour que le calcul soit juste)
    - si T2 ne dépend pas de T1 et T1 ne dépend pas de T2, alors T1 et T2 sont des tâches **indépendantes**
    - => Deux **tâches indépendantes** peuvent être exécutées dans un ordre quelconque, voire simultanément (i.e. en **parallèle**)

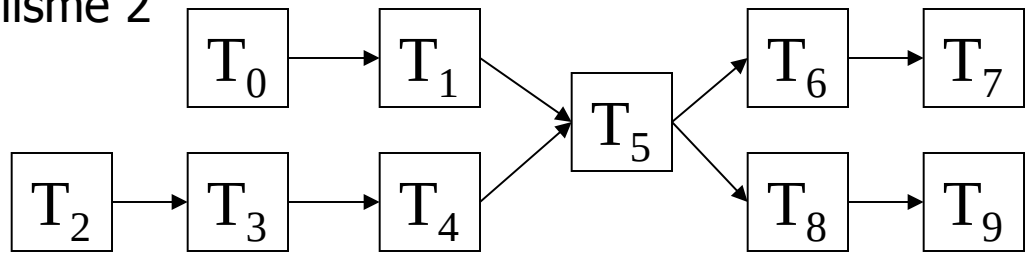
Graphe de dépendance

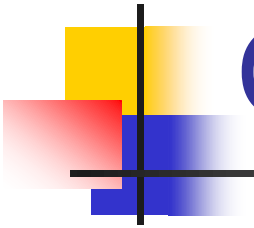
- Graphe de dépendance : met en évidence les relations de dépendances entre les tâches pour mener à bien une action;
- 

signifie T2 dépend de T1;
- Profondeur du graphe donne la dépendance ;
- Largeur du graphe donne l'indépendance (parallélisme) ;
- Programmation séquentielle : dépendance n et parallélisme 1



- Ex programmation parallèle avec 10 tâches : dépendance 6 et parallélisme 2



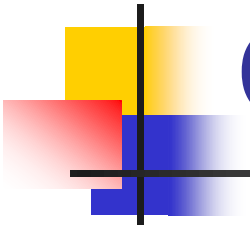


Concurrence

- Les exécutions des tâches parallèles sont :
  - **simultanées** :
  - ou **alternées** (i.e. une tâche est interrompue pour laisser place à une autre tâche, puis reprise ultérieurement) ;
  - ou bien les deux ;

- ➔ Problème : les tâches peuvent accéder à des données communes et les modifier (notion de **section critique**);

- ➔ Solution : il faut trouver des mécanismes pour assurer la cohérence des données (mécanismes de **verrous** et d'**exclusions mutuelles**);

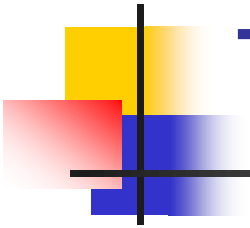


Communication

- **Communication et synchronisation :**
  - Pour assurer la cohérence d'un calcul, des tâches parallèles peuvent se donner des "rendez-vous" avant de poursuivre.

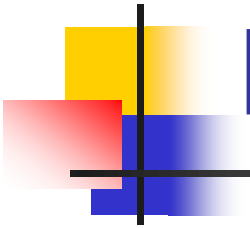
- Ces points de rendez-vous sont appelés points de **synchronisation** :
  - Si la synchronisation concerne toutes les tâches parallèles
    - ➔ **synchronisation globale ou collective**
  - Si les tâches ont des espaces d'adressages différents (ex sur machine à mémoire distribuée),
    - ➔**communications** (échange d'informations entre tâches "cloisonnées" (au sens de la mémoire))

- Communications
  - synchronisation globale => **communication globale ou collective**
  - synchronisation entre 2 tâches => **communication point à point**



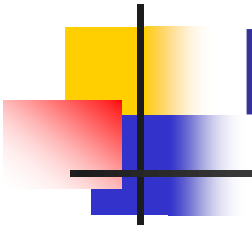
Types de parallélisme

- Quelles sont les sources de parallélisme dans une application ?
- 3 sources :
  - parallélisme de contrôle (tâches)
  - parallélisme de flux (pipeline)
  - parallélisme de données



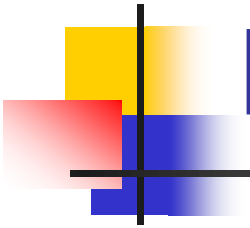
Parallélisme de contrôle

- Idée : *"Faire plusieurs choses en même temps"*
- Constatation naturelle :
  - **Une application est composée d'actions que l'on peut faire en même temps**
  - Exemple : l'exécution d'une recette de cuisine avec plusieurs cuisiniers
- Exploitation du parallélisme de contrôle consiste à gérer les dépendances entre les actions d'une application pour obtenir une allocation des ressources de calcul aussi optimale que possible
  - **Extraction de ce parallélisme à partir du graphe de dépendance : largeur du graphe**
- **En pratique, degré de parallélisme peu élevé et souvent complexe à mettre en place** (ex : ILP dans les processeurs)
- Correspond aux modèles de programmation parallèle suivants :
  - **MIMD (Multiple Instruction Multiple Data)**
  - **MPMD (Multiple Program Multiple Data)** : les processeurs exécutent des programmes différents avec leurs propres données



Parallélisme de flux

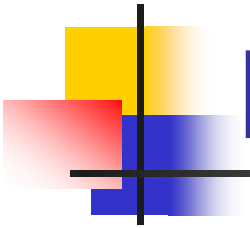
- Idée : *"Travailler à la chaîne"*
- **Principe : mode de fonctionnement en Pipe-line**
  - on dispose d'un flux de données, généralement similaire, sur lesquelles on doit effectuer une suite d'opérations en cascade
  - les ressources de calcul sont associées aux actions et chaînées de manière à ce que les résultats des actions effectuées au temps T soient passés au temps T+1 au processeur suivant
  - Exemple : machine vectorielle
- **Degré de parallélisme est fonction de la profondeur du pipe (nombre d'étages)**
- **Travail sur des données vectorielles** :
  - les vecteurs doivent être long pour minimiser le coût du chargement du pipe
- **Le flux de données doit être continu au maximum** (chaque arrêt/reprise du flux provoque un déchargement / chargement du pipe)



Parallélisme de données

- Idée : *"Répéter une action sur des données similaires"*
- **Principe : on partage les données et non plus les tâches**
  - exemple : corvée de pommes de terre à l'armée avec plusieurs appelés du contingent
- **Degré de parallélisme potentiellement élevé car fonction de la taille des données**
- Correspond au modèle de programmation parallèle **SPMD (Single Program Multiple Data)**
  - Tous les processeurs exécutent le même programme avec leurs données propres
  - Viable pour un grand nombre de données
  - Très efficace pour beaucoup d'algorithmes de calcul intensif (calcul scientifique, films d'animation) : par la suite, nous allons nous concentrer sur ce modèle de programmation

Paradigmes de la programmation parallèle

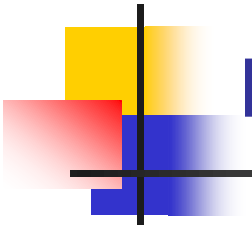


- Indépendamment des architectures matérielles des machines, deux modèles de programmation parallèles se dégagent :
  - modèle de programmation à mémoire distribuée
  - modèle de programmation à mémoire partagée
- En théorie, chaque modèle peut s'implémenter sur n'importe quel type d'architecture avec des effets collatéraux plus ou moins importants sur les performances

Modèle de programmation à mémoire distribuée

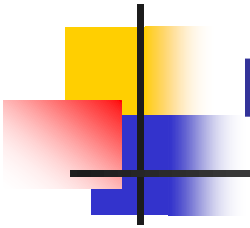
- Conditions :
  - **les tâches parallèles travaillent sur des mémoires distinctes, invisibles les unes des autres**
  - **les données (tableaux, etc…) sont éclatées (on parle de données distribuées) sur les différentes tâches parallèles**
- Conséquence : pour assurer la justesse du résultat final, des communications inter-tâches deviennent obligatoires
- On parle alors de **programmation par passage de messages (*Message Passing*)**
- **Adapté au modèle SPMD**

Modèle de programmation à mémoire distribuée



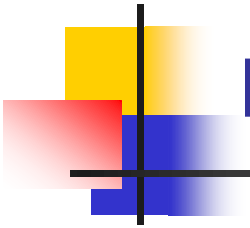
- Implémentation sur architecture à mémoire distribuée
  - facile car en reprend le principe :
    - sur une machine à mémoire distribuée, des processus qui ont leurs propres espaces d'adressage doivent envoyer des messages par le réseau pour échanger des infos
- Implémentation sur architecture à mémoire partagée
  - guère plus difficile :
    - par le biais de plusieurs processus en utilisant les segments de mémoire partagée pour les communications
    - par le biais d'un processus multithread en utilisant la mémoire de celui-ci pour échanger les informations entre threads

Modèle de programmation à mémoire distribuée



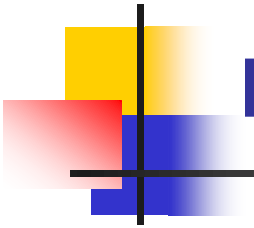
- Encapsulation des échanges de messages, implémentées par des bibliothèques, comme par exemple :
  - PVM (Parallel Virtual Machine) : une des premières bibliothèques portables d'échanges de messages
  - MPI (Message Passing Interface) : le standard à l'heure actuelle, très répandue, issue de la collaboration d'industriels et d'universitaires

Modèle de programmation à mémoire partagée



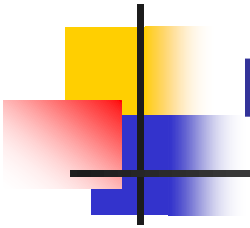
- Condition :
  - **les tâches parallèles ont une visibilité commune de la mémoire**
- Conséquence :
  - il faut gérer les accès concurrents à la mémoire (section critique - "ne pas se marcher sur les pieds")
- Malgré la simplicité apparente de programmation, les performances peuvent être rapidement dégradées car :
  - les sections critiques ne sont pas parallèles (par définition)
  - on ne se rend pas compte de la localité des données et de la hiérarchie de la mémoire (pour les nœuds NUMA)

Modèle de programmation à mémoire partagée

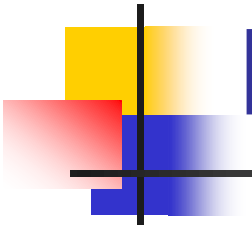


- Implémentation sur architecture à mémoire distribuée
  - difficile : comment "voir" la totalité de la mémoire ?
    - DSM (Distributed Shared Memory) : mécanisme logiciel permettant de donner l'illusion d'une mémoire unique à une mémoire physiquement distribuée
    - peut très vite couter cher car on ne se rend pas compte de l'accès distant aux données
- Implémentation sur architecture à mémoire partagée
  - naturelle car elle en reprend les principes :
    - dans un processus multithread, tous les threads ont accès à la mémoire du processus (qui peut adresser toute la mémoire du neoud)

Modèle de programmation à mémoire partagée



- API POSIX pthread :
  - manipulation normalisée (POSIX) de threads au sein d'un processus
  - adapté pour le modèle MPMD
- OpenMP :
  - manipulation de threads par directive de compilation
- A l'heure actuelle, des outils implémentant le modèle à mémoire partagée font part d'une grande réflexion afin que les applications puissent exploiter au mieux les processeurs multicores
  - TBB, Cilk++, ABB, …



Résumé

- La programmation parallèle utilise les dépendances entre les tâches. Elle fait apparaître de nouvelles notions (concurrence, communication), mais également de nouvelles difficultés (debugging, performances);

- Les 3 sources de parallélisme sont les parallélismes de contrôle, de flux, et de données (ce dernier offrant le degré potentiel de parallélisme le plus élevé);

- 2 modèles de programmation parallèle se dégagent : programmations à mémoire distribuée, et à mémoire partagée. Le modèle à mémoire distribuée SPMD (parallélisme de données) est le plus répandu dans le milieu du HPC.