Programmation Répartie

Présentation

Par semaine

- 1h cours
- 1h TD
- 1h30 TP

Contenu

- Programmation parallèle
- Client / serveur

Prérequis

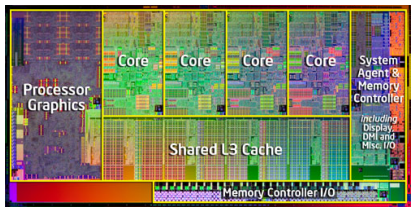
- Programmation Java/C
- Principes des systèmes d'exploitation (Semestre 3)

Introduction au calcul parallèle

Pourquoi un cours sur le parallélisme ?

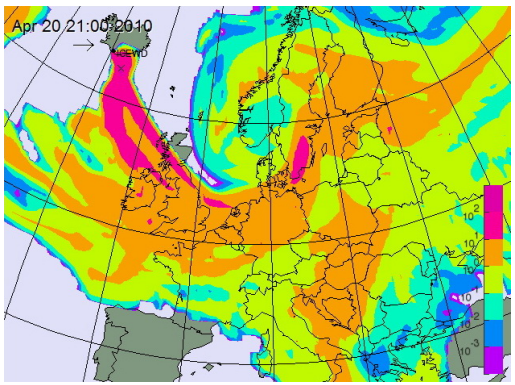
Présent matériellement partout

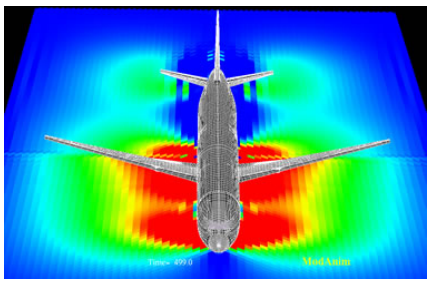
- Calculateur
- Ordinateur personnel
- Smartphone

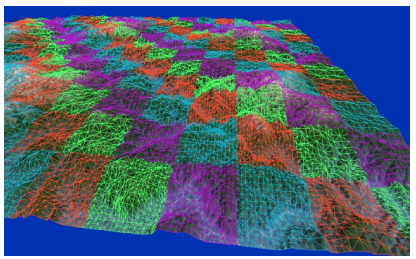


Utile pour les applications :

- Interface graphique
- Traiter des problèmes de très grande taille
  - Modélisation de phénomène physique
  - Traitement requêtes
  - Traitement signaux







Exemples application parallèle

- Science : Évolution climatiques, Tsunami, tremblement de terre, courant océaniques
- Industrie : Crash d'avion, dynamique des fluides
- Génétique : décodage du génome
- Astronomie : traiter données télescope
- Défense : simulation explosion nucléaire
- Animation : Gérer les lumières, les textures sur des modèles 3D.

L'augmentation des performance par l'augmentation de fréquences des processeurs est limitée par les limites physiques

- Consommation électrique
- Dissipation de la chaleur
- Limite gravure

Pour augmenter les performances, le seul moyen est d'augmenter le nombre d'unité travaillant en parallèle.

Pourquoi un cours sur le parallélisme

- Présent matériellement partout
- Nécessaire pour traiter les problèmes de grande taille
- Obtenir de meilleurs résultats

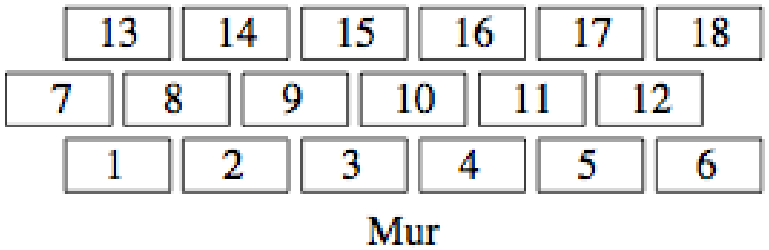
Qu'est-ce que le calcul parallèle ?

- Diviser une application en sous tâches
- Exécuter ces sous tâches en parallèle

Comment écrire un algorithme parallèle ?

- Trouver ses sous tâches
- Trouver la bonne division en sous tâches (le grain)
- Avoir une division adaptée à la machine cible

Analogie du parallélisme



Un seul maçon pour construire un mur

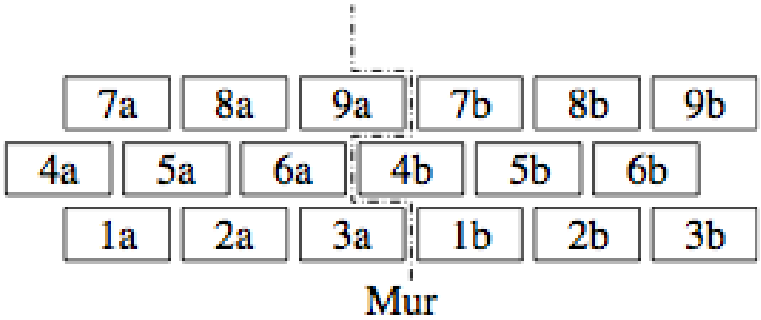
- Procède par rangées
- Lent

	7a	7b	8a	8b	9a	9b
4a	4b	5a	5b	6a	6b	
1a	1b	2a	2b	3a	3b	

Mur

Deux maçons pour construire un mur

- Posent une brique après l'autre
- Nécessite de se synchroniser pour ne pas se gêner pour les mettre en place



Deux maçons pour construire un mur : seconde version

- Chacun s'occupe d'une portion du mur
- Plus efficace car moins besoin de coordination (uniquement pour la jonction du mur)

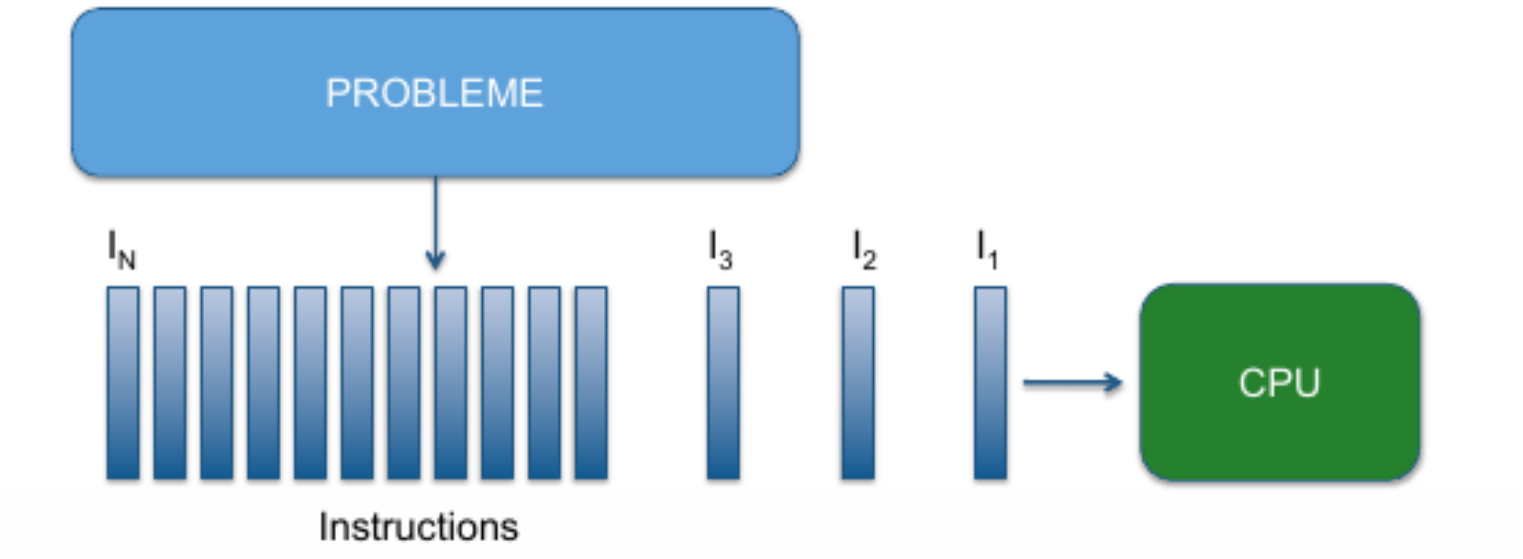
Déductions

- Le travail a plusieurs peut être plus efficace
- Nécessite du travail supplémentaire de synchronisation
- Plusieurs répartitions du travail existent (efficacités différentes)

Généralisation

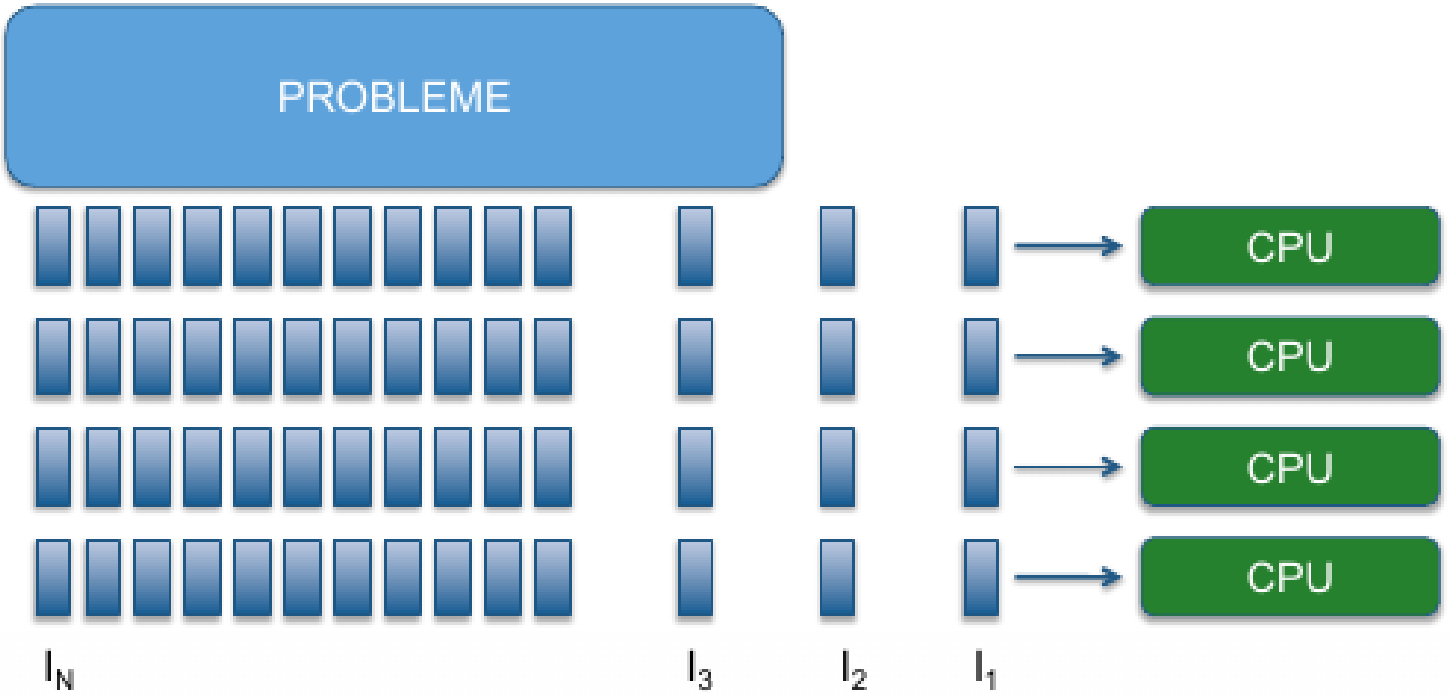
- Décomposer une tache en sous-tâches suffisamment indépendantes
- Surcoût du parallélisme (celui-ci doit être inférieur au gain apporté par la décomposition)
- Plusieurs algorithmes parallèles peuvent répondre au problème avec des efficacités différentes

Équivalent sur machine



Programme séquentiel

- Unique unité de calcul
- Suite d'instructions qui s'exécutent les unes après les autres



Programme parallèle

- Unité de calcul multiples
- Programme divisé en sous parties suffisamment indépendantes
- Une suite d'instructions sur une unité de calcul est un programme séquentiel
- Deux instructions s'exécutant sur deux unités de calcul sont indépendantes l'une de l'autre

Qu'espère-t-on avec un calcul parallèle

Avoir une bonne **accélération** (ou speedup)

- $S = \frac{T_{seq}}{T_{par}}$
- Idéalement pour p unités de calcul, l'accélération idéale est de p
- Malheureusement ce n'est que rarement le cas
  - Partie séquentielle de l'algorithme
  - Surcoût lié à la synchronisation, transfert de données,...

Qu'est-ce qu'une machine parallèle ?

Qu'est-ce qu'une machine parallèle ?

Collection d'**éléments de calcul** capable de **communiquer et de coopérer** pour résoudre rapidement des **problèmes** (de grande taille)

- Quels sont ces éléments de calcul ?
- Comment peuvent-ils communiquer et coopérer ?
- Quels sont les problèmes parallélisables ?

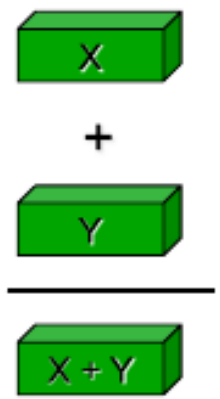
Classification des architectures

Classification de Flyn

Flot instr \ Flot données	Unique	Multiple
Unique	SISD	SIMD
Multiple	MISD	MIMD

Single Instruction Single Data

Processeurs séquentiels classiques

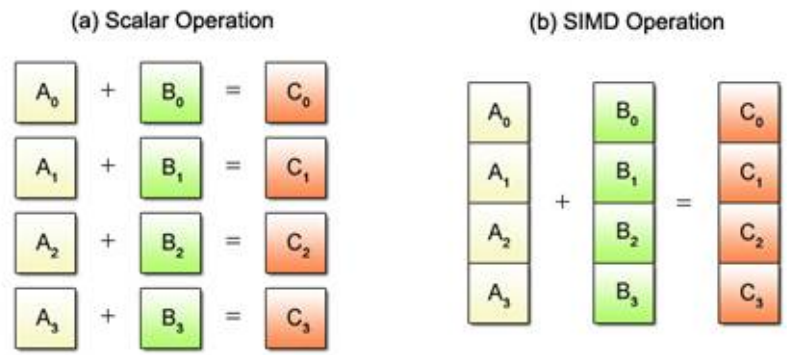


Mutliple Instructions Single Data

Pipeline : exécutent plusieurs instructions en meme temps sur la meme donnée.

Single Instruction Multiple Data

- Exécution d'une même instruction sur des données différentes
  - ex : instructions SSE sur les processeurs intel
  - calcul sur GPU : Cuda pour NVidia
  - architectures vectorielles : Cray



Multiple **I**nstructions **M**ultiple **D**ata

- Architectures multiprocesseurs.
- Chaque processeur a sa propre mémoire locale, communication entre les processeurs.
- - Mémoire **distribuée** : communication par envoi de message
  - Mémoire **partagée** : les processeurs partagent une mémoire commune

top 500

- Classement des ordinateurs les plus puissants
- Renouvelé tous les 6 mois
- `top500.org`

Conclusions

Constats

- Toutes les **machines** sont parallèles désormais
- Les besoins d'**applications** parallèles sont partout
  - Traiter les problèmes de **grande tailles**
  - Améliorer la **puissance**
- Plusieurs architectures parallèles différentes

Pour les programmeurs

- Trouver le parallélisme dans un problème
- Trouver la **bonne division** des tâches
- Adapter son algorithme à la machine cible
- Combiner les différentes formes de parallélisme