

1

Introduction A propos du parallélisme



Coulaud - PG 305 - V1.1

Oct. 2017 - 3

Plan

1. Introduction
2. Les grands types d'architectures
3. Les lois d'Amdahl et de Gustafson
4. Les modèles de programmation



Coulaud - PG 305 - V1.1

Oct. 2017 - 4

Définition du parallélisme

Parallélisme : désigne le fait, pour un système, d'être capable d'effectuer plusieurs unités de calcul simultanément pour accélérer l'exécution d'une application

Différents niveaux :

- | | |
|-------------------------------------|-----------------------|
| 1. Plusieurs unités de calcul 1 à 4 | Compilateur |
| 2. Cœurs | Compilateur + langage |
| 3. Machines : grappe de PCs | Langage |
| 4. Grappe de grappes | Langage |
- 



Le parallélisme pour qui, pourquoi ?

Simulations numériques

- simulent des phénomènes physiques :
aéronautique, météorologie, chimie, géophysique, ...
- traitent
de gros volumes de données (Tera voir Peta octets)
durent longtemps (plusieurs semaines)

Impossible de tourner en séquentiel car

- limitation mémoire,
- temps de calculs.

→ Il faut diviser le travail pour accélérer les calculs

Objectif : résoudre des problèmes de plus en plus gros mais avec des restitutions de plus en plus rapides !!



Taxinomie de Flynn

Classification des ordinateurs

	1 flot d'instruction	> 1 flot d'Instruction
Donnée unique	SISD Von Neumann	MISD pipeline
Plusieurs données	SIMD GPU	MIMD machine moderne

S : Single, M : Multiple

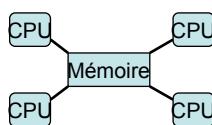


Architecture à mémoire partagée (1)

Tous les processeurs ont accès à une mémoire globale et partagent le même espace d'adressage

Le système exécute une seule copie de l'OS

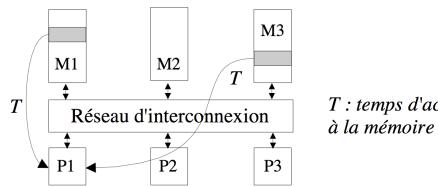
Les processeurs communiquent par des lectures/écritures dans la mémoire globale



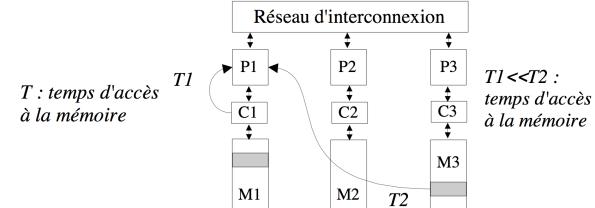
Architecture à mémoire partagée (2)

Classification en fonction de l'accès à la mémoire

Uniform Memory Access



NUMA et ccNUMA



Cache Coherent - NUMA

@ Les architectures parallèles - MPI Aurélia Marchand



Architecture à mémoire partagée (3)

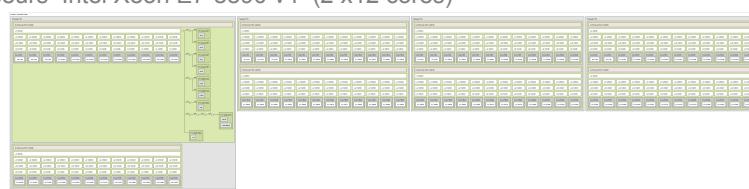
Caractéristiques

2 Deca-core Ivy-Bridge



Caractéristiques

4 processeurs Intel Xeon E7-8890 v4 (2 x12 cores)



Architecture à mémoire partagée (4)

Avantages

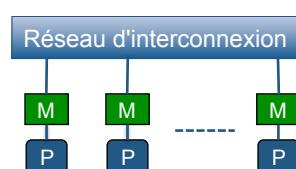
- Simplicité d'utilisation ;
- Accès rapide à la mémoire ;
- Parallélisme de haut niveau :
Directives, parallélisation automatique.

Inconvénients

- Nombre de processeurs limité ;
- Limitation de la taille de la mémoire ;
- Goulot d'étranglement à la mémoire ;
- Affinité mémoire pour de la performance (cc-numa).



Architecture à mémoire distribuée (1)



Caractéristiques :

- Interconnexion (réseau local rapide) de systèmes indépendants (nœuds) ;
- Mémoire locale à chaque processeur ;
- Chaque processeur exécute des instructions identiques ou non, sur des données identiques ou non.



Architecture à mémoire distribuée (2)

Majorité des grands calculateurs du Top500

Deux catégories

1. Assemblage de nœuds classiques
Mémoire importante, processeurs rapides
Power7, intel, Sparc, AMD → cluster
2. Beaucoup de petits cœurs avec peu de mémoire
BlueGene L/P/Q, ...

Interconnexion rapide

1. Infiniband
2. Réseaux propriétaires SGI, CRAY, ...



Architecture à mémoire distribuée (3)

Avantages

- Grande puissance de calcul
Pas de limite en nombre de processeurs ni de taille mémoire ;
- Machine pas chère ;
- Portabilité : même nœud que sur votre bureau.

Inconvénients

- Échanger les données de manière explicite;
- Performance difficile à obtenir
latence réseau >> latence mémoire
- Puissance électrique et refroidissement ☺ ;
- Administration plus compliquée.



Loi d'Amdahl

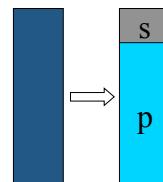
Prédit l'accélération théorique maximale d'un programme en fonction :

- du nombre de processeurs N
- de la proportion d'activité parallélisable pour un problème donné et une taille de problème fixée.

T_s = temps séquentiel du programme

T_p = temps d'exécution parallèle

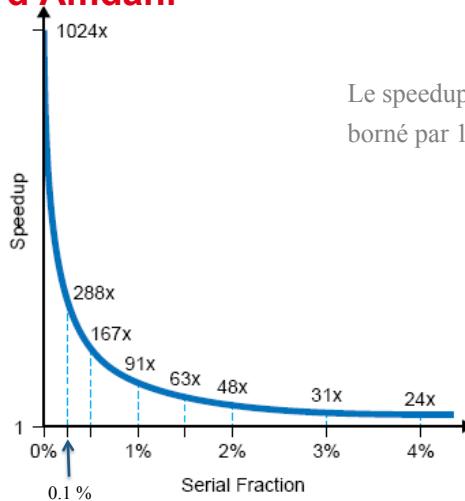
$$T_p = s + p$$



$$\text{speedup} = \frac{T_s}{T_p} = \frac{s + p}{s + p/N} = \frac{1}{a + (1-a)/N} < \frac{1}{a} \quad (N \rightarrow \infty)$$



Loi d'Amdahl



Le speedup (ou accélération) est borné par $1/a$.

FIGURE 1. Speedup under Amdahl's Law



Loi d'Amdahl

Mauvaises nouvelles pour les applications parallèles

Deux points intéressants :

- Il est impératif de limiter la partie séquentielle
- Un ordinateur parallèle devrait être un ordinateur séquentiel rapide pour résoudre rapidement la partie séquentielle.

Que se passe-t-il si on augmente la taille du problème initial ?



La loi de Gustafson

- Loi d'Amdahl = accélération à travail constant
- Dans un programme parallèle la quantité de données à traiter augmente donc la partie séquentielle diminue.

Hypothèse : T_p est constant $\rightarrow T_s = s + p N$

avec N le nombre de cœurs, a la fraction séquentielle du code de départ.

$$\text{speedup} = \frac{T_s}{T_p} = \frac{s + p * N}{s + p} = a + (1 - a) * N$$

Si la taille du problème augmente $p \rightarrow \infty$ alors $a = s / (s + p) \rightarrow 0$

$$\text{Speedup} \rightarrow N$$

Cette loi est plus optimiste que celle d'Amdahl.



Loi de Gustafson

La limite de la loi d'Amdahl peut être dépassée **si** la quantité de donnée à traiter augmente.

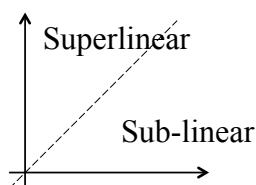
Pour un problème de taille constante par cœur et en supposant que la durée de la partie séquentielle n'augmente pas avec la taille globale du problème.

$$\text{speedup} \leq N + (1 - N)a$$

Si la taille du problème « scale » suffisamment alors n'importe quelle efficacité peut être atteinte pour n'importe quel nombre de processeurs.



Speedup super linéaire ?



Quelques fois des accélérations super linéaires sont observées !

- Effets mémoire/cache
- Plus de processeurs généralement fournissent une meilleur rapport mémoire/cache
- Temps de calcul décroît à cause d'une meilleur taux de succès page/cache.



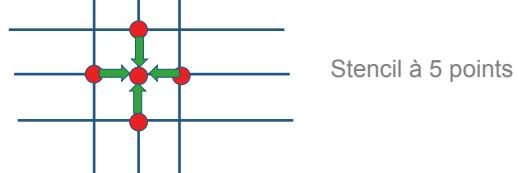
Les modèles de programmation

Équation de la chaleur discrète

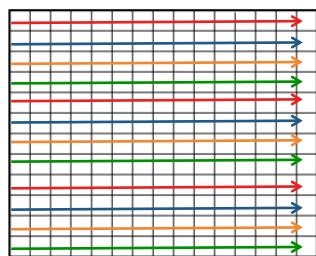
Grille uniforme NxN points

h le pas

```
for(j = 0; j < N; ++j)
    for(i = 0; i < N; ++i)
         $u_{i,j}^{n+1} := \frac{1}{h^2} \{ u_{i-1,j}^n + u_{i+1,j}^n - 4u_{i,j}^n + u_{i,j-1}^n + u_{i,j+1}^n \} + f_{i,j}$ 
```

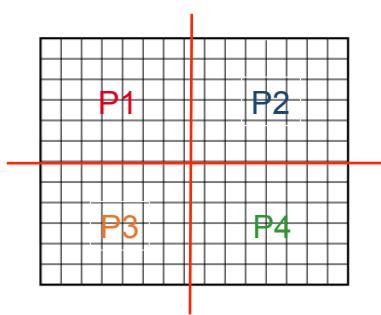


Les modèles de programmation



Parallélisme de la boucle interne
Faible volume de calcul

Parallélisme à grain fin



Décomposition de domaines.

Volume de calculs > volume de communications

Parallélisme à grain grossier



Parallélisme à grain fin

Parallélisme de boucle :

éclater le calcul d'une boucle sur différents processeurs

Parallélisme incrémental :

le programme séquentiel évolue vers un programme parallèle

Parallélisme de haut niveau

soit par compilation automatique soit par des directives

Difficile d' avoir de la performance sur beaucoup de processeurs

- Il n'y a pas que des boucles (I/O, conditions, ...)

- Attention à l'overhead $n > N_{critique}$



Les modèles de programmation Parallélisme à grain grossier

SPMD = Single Program Multiple Data

- Même code qui exécute des données différentes
- Exemple : décomposition de domaine, ...



Modèle qui marche pour la performance mais il faut échanger des données :

- utiliser la mémoire globale
- utiliser le réseau : échange de messages MPI, invocation à distance

Plus difficile à réaliser, on gère le parallélisme à la main



Différents paradigmes

Threads :

- Les threads partagent le même espace d'adressage
- Développement simplifié (tout est visible)

Remote Procedure Call :

- Protocole réseau qui permet de faire des invocations de méthodes à distance.
- Modèle client/serveur

Échange de messages :

- Ensemble de tâches qui utilisent une mémoire locale et qui échangent de données via le réseau.
- Envoi/Réception de messages est explicite



Différents paradigmes (suite)

Partitionned Global Address Space (PGAS) : permet la référence à la mémoire locale du processus et à la mémoire des processus voisins

Data parallélisme :

Un ensemble de tâches travaille sur une portion distincte d'une structure de données partagées (tableaux, ...).

Le paradigme choisi dépend de l'architecture cible et du choix du programmeur



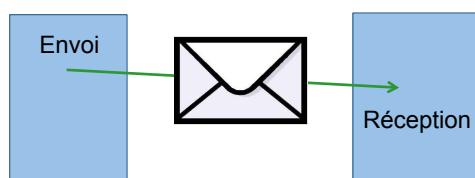
Différents paradigmes (suite)

Quelques remarques

1. Un outil n'est pas exclusif : OpenMP utilise des threads.
2. Les modèles hybrides sont possibles et nécessaires
MPI+threads, RPC+threads, ...
3. L'échange de messages est le plus utilisé
Historique, portable sur toutes les machines, ... mais pose de gros problème de performance sur les architectures actuelles.
Facile à comprendre.



Échange de messages



L'envoi et la réception doivent être explicites dans le code pour que la communication ait lieu. Différent du modèle RPC.

