Chapitre III : Fondements de la programmation parallèle

ISSAT Sousse

2023-2024

Sommaire

Introduction

- Dans la décennie précédente, le monde a vécu une des plus excitantes périodes dans le développement des ordinateurs : les améliorations des performances des ordinateurs ont été spectaculaires.
- De nouveaux **paradigmes de programmation**, **langages**, **techniques d'ordonnancement et de partitionnement**, et **algorithmes** sont nécessaires pour exploiter efficacement la puissance de ces machines sophistiquées et pour gagner de la puissance soit au niveau algorithmique, soit au niveau architectural.

Sommaire

formes du parallélisme

Deux principales formes qui peuvent être caractérisées :

- Implicite
- Explicite

L'approche mixte(Implicite/Explicite)une autre forme.

Approche Implicite

- Cette approche se traduit par les langages parallèles et les compilateurs paralléliseurs.
- Prendre un langage existant et laisser le compilateur faire tout le travail.
- Approche idéale du point de vue de l'utilisateur :
  - Codes existants peuvent tourner sans rien changer ni optimiser.
  - Economie dans les coûts de développement.

Approche Implicite

- Le plein potentiel parallèle d'un problème ne peut pas toujours être exploité.
- Pour l'écriture d'un nouveau programme, il est conseillé que le programmeur ait une certaine connaissance sur les possibilités de détection du parallélisme du compilateur afin d'écrire du code qui s'exécute rapidement.
- Déviation du but original de détection automatique du parallélisme.

Approche Explicite

- Cette approche est justifiée par le fait que l'utilisateur est souvent le meilleur juge de la façon avec laquelle le parallélisme peut être exploité efficacement pour une application particulière.
- Avantages :
  - Les utilisateurs sont déjà habitués aux langages de base.
  - Pour que leurs programmes s'exécutent efficacement, seulement un ensemble limité de fonctions supplémentaires a besoin d'être appliqué.

Approche Explicite

- Problèmes :
  - Le programmeur est responsable d'une grande partie de l'effort de parallélisation.
  - Manque de portabilité : ces langages sont destinés à des classes spécifiques de machines.
  - Manque de standardisation : beaucoup d'extensions ont été développées avec des fonctionnalités similaires mais des apparences différentes.

Sources du parallélisme

- Exploiter toute la puissance de calcul d'une machine parallèle est souvent difficile :
  - Les algorithmes et les applications ne possèdent pas tous le même parallélisme potentiel.
  - Mettre en évidence le potentiel de parallélisme d'un algorithme et l'exploiter au mieux des capacités de la machine reste un problème difficile.

Sources du parallélisme

- Problème du choix de l'algorithme pour résoudre un problème donné :
  - Plusieurs algorithmes séquentiels possibles.
  - Le parallélisme extrait du meilleur algorithme séquentiel n'aboutit pas forcément au meilleur algorithme parallèle.
  - Chaque algorithme se prête plus ou moins à une parallélisation.
- On identifie trois sources de parallélisme :
  - Parallélisme de données,
  - Parallélisme de tâches (ou de contrôle),
  - Parallélisme de flux.

Sources du parallélisme

- Parallélisme de données
  - Exploite comme source de parallélisme la régularité des données
  - Applique en parallèle un même calcul à des données différentes
- Parallélisme de tâches
  - Consiste à appliquer des calculs différents sur des données différentes en même temps afin de générer du parallélisme
- Parallélisme de flux
  - Correspond à la technique du travail à la chaîne
  - Chaque donnée subit une séquence de traitement réalisée en mode pipeline en exploitant une régularité des données
- Lorsqu'on combine plus qu'un mode, on parle de parallélisme mixte.

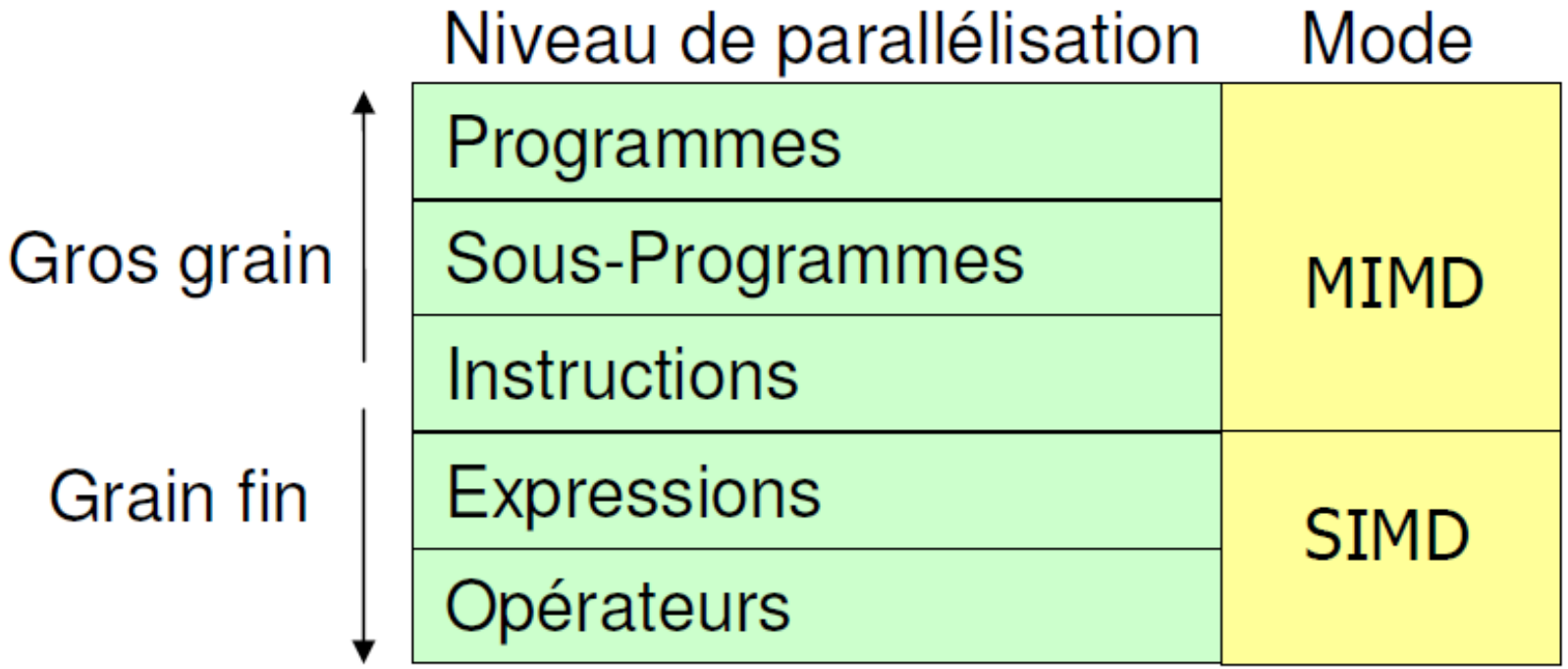
Grain (ou granularité) et degré de parallélisme

- La granularité et degré de parallélisme sont deux paramètres importants à étudier lorsqu'on parallélise une application.
- Ils ont une grande influence sur l'efficacité de la parallélisation.

Grain de parallélisme

- Taille moyenne des tâches élémentaires qui ont guidé la parallélisation (en nombre d'instructions, temps d'exécution, . . .).
- Le choix du grain de parallélisme est fortement lié à l'architecture sous- jacente.
- Ce paramètre a généralement une grande influence sur le paramètre du degré de parallélisme.

Grain de parallélisme



degré de parallélisme

- C'est la mesure du nombre de sous-tâches exécutées simultanément dans un programme parallèle.
  - indication du nombre de processeurs que l'on pourra utiliser.
- Il peut fortement varier dans les différentes parties du programme
  - on considère souvent le degré maximal, moyen et minimal
- Ces degrés peuvent être évalués statiquement à la compilation, ou dynamiquement lors de l'exécution.
- Remarque : Bien que l'exécution d'un programme avec un nombre de processeurs égal au degré maximal de parallélisme soit, en principe, la plus rapide, cela ne signifie pas forcément que c'est le nombre optimal de processeurs !

Paradigmes de programmation parallèle

- La conception des applications parallèles se base sur l'adoption de paradigmes de programmation bien définis.
- Le choix du paradigme adéquat est déterminé par les ressources de calcul parallèles disponibles et par le type du parallélisme inhérent dans le problème.
- Quelques paradigmes reconnus.
- Un programme = combinaison de plusieurs paradigmes.

Le paradigme ≪ Parallélisme de phases ≫

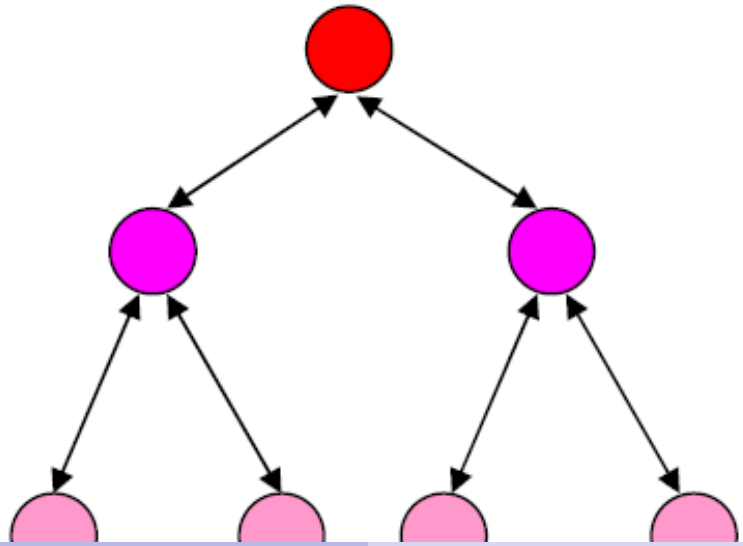
- Programme = suite d'étapes
- Étape = 2 phases : calcul + interaction
  - Calcul : de multiples processus réalisent chacun un calcul indépendant
  - Interaction entre les processus :
    - Synchronisation
    - Communication bloquante, etc.
- Inconvénients :
  - Pas de recouvrement entre l'interaction et le calcul.
  - Nécessité et difficulté de maintenir l'équilibrage de charge entre les processus.

Le paradigme ≪ Diviser pour Paralléliser ≫

- Découverte dynamique d'un arbre de calculs.
- Un processus parent divise le travail entre ses fils.
- Les processus fils combinent leurs résultats vers leur père.
- Division et regroupement récursifs.

Diviser pour paralléliser (Schéma algorithmique)

- Réduction du problème à des instances indépendantes plus petites.
- Résolution parallèle récursive des sous-instances, en appliquant récursivement la même technique à chacune des sous-instances.
- Construction de la solution du problème initial à partir des solutions de chacune des sous-instances.



Le paradigme ≪Maître/esclaves≫

- Un processus maître = coordinateur
  - Exécute le code séquentiel
  - Initie des processus esclaves
  - Transmet du travail à ces processus esclaves
  - Attend les résultats des esclaves
  - Itération de l'assignation de travail
- Des processus esclaves
  - Attendent du travail du maître
  - Exécutent le travail
  - Retournent le résultat au maître
- Paradigme simple
- Maître = goulot d'étranglement
- Paradigme non-extensible

Sommaire

Evaluation des performances du parallélisme

- L'étude théorique des performances des algorithmes parallèles permet de connaître par exemple :
  - L'efficacité d'un programme,
  - Le gain issu du parallélisme,
  - Son comportement futur dans le cas de l'augmentation du nombre de processeurs,
  - Etc.
- Il existe de nombreuses métriques de performances. Les plus utilisées sont les suivantes :
  - l'accélération
  - L'efficacité

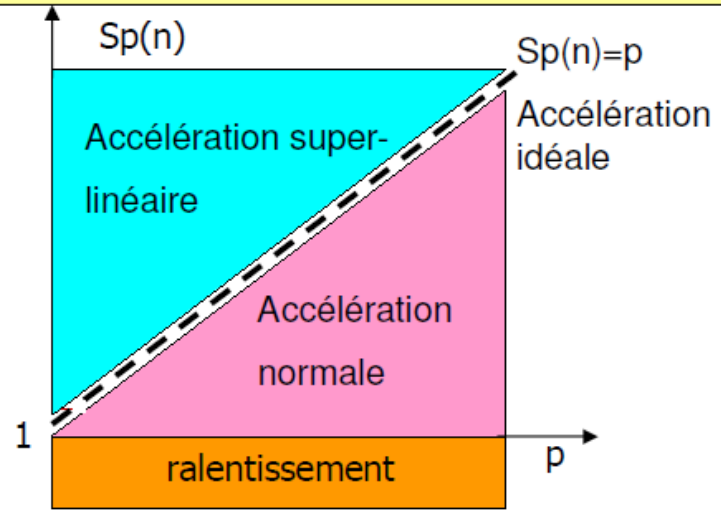
Terminologie

- Soit à résoudre une instance d'un problème de taille n
- On note par :
  - T1(n) le temps d'exécution séquentiel sur un seul processeur
  - Tp(n) le temps d'exécution parallèle sur p processeurs

Accélération

- On définit le facteur d'accélération, qui permet de mesurer le gain en temps dû à la parallélisation (ou le taux d'utilisation des processeurs), par le rapport :

Sp(n) = T1(n) / Tp(n) =>	Sp(n)<1	: on ralentit ! (mauvaise parallélisation)
	1<Sp(n)≤p	: « normal ». Plus Sp est proche de p, meilleur est l'algorithme parallèle.
	Sp(n)>p	: accélération super-linéaire : analyser et justifier



Accélération Super-linéaire

- Ce n'est pas magique, et ce n'est pas normal
  - on doit analyser le phénomène et l'expliquer
  - corriger une erreur ou exploiter une optimisation
- Exemples d'explications :
  - On ne fait plus les bonnes opérations (résultat faux)
  - l'algorithme séquentiel n'est pas le même que l'algorithme utilisé pour la parallélisation, et qu'il est moins efficace.
  - Les données tiennent dans le cache total des p processeurs
  - On a modifié l'algorithme de départ et on converge plus vite (exp de l'algorithme génétique optimisé)
  - On cherche une solution dans un arbre et on stoppe le programme

Efficacité

- On définit l'efficacité, qui est équivalent à un rendement, d'un algorithme parallèle par le rapport :

$$Ep(n) = Sp(n) / p \quad \Rightarrow \quad 0\ (=0\%) < Ep(n) \leq 1\ (=100\%)$$

$$Ep(n) > 1 \Leftrightarrow \text{accélération super-linéaire}$$

- Elle permet de mesurer le taux moyen d'utilisation des processeurs.
- Plus l'efficacité est proche de 1, plus l'algorithme a de bonnes qualités parallèles.

Exemple (Multiplication matricielle)

Algorithme A

Temps en séquentiel = 10 mns
Nb procs = 10
Temps en // = 2 mns
Accélération = 10/2 = 5 (l'application va 5 fois plus vite)
Efficacité = 5/10 = 0.5

Algorithme B

Temps en séquentiel = 10 mns
Nb procs = 3
Temps en // = 4 mns
Accélération = 10/4 = 2.5 < 5
Efficacité = (5/2)/3 = 0.8 > 0.5

Remarque

- L'utilisateur s'intéresse surtout à l'accélération obtenue.
- L'acheteur de la machine s'intéresse beaucoup à l'efficacité.
- Le développeur s'intéresse aux deux.

Choix de la référence séquentielle

A quels programme et exécution séquentielle se comparer ?

- Même programme lancé sur un seul processeur ?
- Même algorithme implanté en séquentiel ?
- Meilleur algorithme séquentiel connu ?
- Compilation séquentielle avec le même compilateur ?
- Compilation avec le meilleur compilateur séquentiel ?
- Optimisations séquentielles autorisées par la parallélisation ?
- Optimisations séquentielles maximales ?
- Exécution sur un seul processeur de la machine parallèle ?
- Exécution sur la meilleure machine séquentielle ?

Choix de la référence séquentielle

- Tous les choix sont plausibles :
- Chaque choix de référence séquentielle correspond à :
  - Un point de vue différent,
  - Une préoccupation différente,
  - Un objectif d'analyse différent
- Exemple de choix :
  - Utilisateur final : SON programme séquentiel sur SA machine séquentielle
  - Paralléliseur : même algorithme sur un processeur de la machine

Sources de perte de performances

- Sous-optimisation séquentielle ⟷ Aspects séquentiels
- Fraction séquentielle
- Surcoût des opérations de gestion du parallélisme
- **Surcoût dû aux communications**
- Déséquilibre de charge ⟷ Algorithmique et programmation parallèle
- ES séquentielles/séquentialisées
- Sous-optimisation des outils et langages parallèles ⟷ Environnement de développement

Remarque

- L'approche précédente est raisonnable pour les machines parallèles à mémoire partagée. Cependant, sur des machines parallèles à mémoire distribuée, elle est insuffisante.
- Une autre définition du facteur d'accélération a été proposée par Gustafson qui est basée sur la loi d'Amdahl.

Scalabilité

- Une architecture parallèle est dite scalable si elle peut être étendue (resp. réduite) à un système plus large (resp. petit) avec une augmentation (diminution) linéaire en sa performance (coût).
- La scalabilité est utilisée comme une mesure de la capacité du système à fournir des performances augmentées lorsque sa taille est augmentée, par exemple.
- Autrement dit, la scalabilité est une réflexion de la capacité du système à utiliser efficacement l'augmentation des ressources de calcul.

Scalabilité

- En pratique, la scalabilité d'un système peut être exprimée en différentes formes, incluant la vitesse, l'efficacité, la taille , les applications, la génération et l'hétérogénéité.
- Dans un système fortement (resp. faiblement) scalable, la taille du problème nécessite qu'elle augmente linéairement (resp. exponentiellement) en fonction de p pour maintenir une efficacité fixe.