IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TD1 – Des parallélismes

Modélisation et Programmation Concurrente – ELU 512

Objectifs

L'objectif de cette séance est de découvrir différentes formes de parallélismes et de concurrences.

À la fin de l'activité les étudiants devront être capables de :

- distinguer le parallélisme de flux, de contrôle et de donnée ;
- d'évaluer l'impact en terme de complexité de la parallélisation de quelques algorithmes simples.

Exercices

▷ **Exercice 1 (*Parallélisme de flux*) :**

On imagine une chaîne de production de voitures. On considère 5 postes de travail :

1. Arrivée du châssis
2. Ajout des roues
3. Ajout des sièges
4. Ajout du moteur
5. Ajout de la carrosserie

Question 1.1 : *Dessinez l'architecture de la chaîne de production.*

Question 1.2 : *Pour une tâche décomposable en m sous-tâches, chacune nécessitant* 1 *unité de temps, combien de temps faut-il à une* pipeline[1] *de m-étages pour produire n unités.*

Question 1.3 : *Avec une seule ressource de construction (sans* pipeline*), combien d'unités de temps aurait-il fallu ?*

Question 1.4 : *Critiquez l'argument suivant :*

1. Chaîne de traitements permettant le début d'un traitement même si la donnée précédente n'est pas encore traitée : `https://en.wikipedia.org/wiki/Pipeline_(computing)`. C'est typiquement ce que l'on peut voir sur une chaîne de production faisant transiter des produits de poste en poste.

Le gain d'efficacité d'un pipeline est proportionnel à son nombre d'étages. Par exemple, un additionneur à 5 étages, chacun d'une unité de temps, calcule 5 fois plus vite qu'un additionneur qui prendrait 5 unités de temps. Avec ce principe, on peut créer un additionneur à 10 étages en ajoutant 5 étages qui ne font que transmettre les données au pipeline aux 5 étages précédents. On gagne ainsi un facteur 10.

▷ **Exercice 2 (*Parallélisme de contrôle*) :**

On considère l'expression suivante

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Un calcul, un algorithme, consiste en un certains nombre d'opérations. On appelle Graphe de flux (*Data-Flow Graph* ou DFG) un graphe qui représente les dépendances entre des opérations. Chaque nœud est une opération, les arêtes décrivent les dépendances. On peut imaginer les arêtes comme des canaux qui permettent aux données de circuler entre les opérations.

Question 2.1 : *Dessinez un graphe de flux pour l'équation ci-dessus.*

Question 2.2 : *Combien d'opération élémentaires ($+, -, \sqrt{}, \times, ^2$) faut-il à un programme séquentiel pour trouver les 2 résultats ?*

Question 2.3 : *Si on dispose de plusieurs unités de calcul quel est le nombre minimum de pas de calcul pour trouver les 2 résultats ? Combien d'unités de calcul a-t-on utilisé ? Quelle est l'accélération ?*

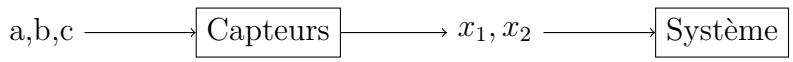
La recherche de ce type de parallélisme (de contrôle) peut être automatisée grâce à la construction et à l'exploitation de graphes de flux. Elle nécessite de :

— choisir l'ordre des opérations ;
— synchroniser (attendre) les résultats.

Les accélérations sont très variables, mais l'ordre de grandeur tourne entre 3 et 5.

En conception de logiciels, c'est le plus souvent ce type de parallélisme qui est programmé. Il faut identifier les activités (objets actifs) et les informations partagées (objets passifs). Puis assurer le partage correct des informations (exclusion mutuelle par exemple) et la synchronisation des activités.

Dans la formule précédente a, b, c sont des réels. On imagine un système où a, b, c sont le résultat de mesures de capteurs et que les racines sont utilisées pour configurer un système.



Question 2.4 : *Transformez le problème précédent en pipeline. Quelle accélération peut-on escompter ?*

▷ **Exercice 3 (*Parallélisme de donnée*) :**

La modélisation et la simulation de phénomènes physiques peuvent s'appuyer sur une approche dite *à éléments finis*. Cette approche (que nous n'introduirons pas ici) permet de résoudre (approximativement) des équations aux dérivées partielles. Elle repose sur une discrétisation des équations continues vers du calcul matriciel.

Nous allons considérer des matrices de n lignes et m colonnes.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{pmatrix}$$

Question 3.1 : *Écrire un algorithme d'addition de deux matrices (de mêmes tailles – h, l). Quelle est sa complexité ?*

Si $A = (a_{ij})$ est une matrice de type (m,n) et $B = (b_{ij})$ est une matrice de type (n,p), alors leur produit, noté $AB = (c_{ij})$ est une matrice de type (m,p) donnée par :

$$\forall i, j : c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}$$

Question 3.2 : *Écrire un algorithme de multiplication de deux matrices. Quelle est sa complexité ?*

Question 3.3 : *À l'aide de plusieurs unités de calcul, jusqu'à quelle valeur peut-on réduire la complexité de l'addition de matrices ?*

Question 3.4 : *Même question avec la multiplication.*

▷ **Exercice 4 (*Synthèse*) :**

Question 4.1 : *En comparant les différents types de parallélisme, quels sont ceux qui permettent les accélérations les plus efficaces ?*

Question 4.2 : *Comment (et où) réalise-t-on ces gains ?*