Module 9: Multiprocesseur

1 Introduction

Le parallélisme consiste, d'une manière générale, à exécuter un algorithme en utilisant plusieurs processeurs plutôt qu'un seul. Par conséquent, un algorithme, parallélisable, pourrait être divisé en plusieurs tâches qui peuvent être exécutées simultanément sur plusieurs processeurs. L'objectif principal du parallélisme est de réduire le temps de résolution des problèmes en utilisant le calcul parallèle.

Les processeurs, depuis 1985, utilisent le principe de pipeline pour chevaucher l'exécution des instructions afin d'améliorer les performances. Ce chevauchement entre les instructions s'appelle le parallélisme au niveau des instructions, ou simplement Parallélisme d'instructions (**ILP : Instruction-level parallelism**). Cette appellation vient du fait que les instructions peuvent être évaluées en parallèle.

Le parallélisme au niveau des microprocesseurs est un autre type de parallélisme au niveau matériel appelé aussi micro-parallélisme. À titre d'exemple, dans les ordinateurs modernes, nous pouvons avoir plusieurs unités de calcul arithmétique (dans le cas du Pentium, d'unités de calcul flottant, ou dans le cas des processeurs MIPS (Microprocessor without Interlocked Pipeline Stages), plusieurs additionneurs, multiplicateurs, etc.) pouvant fonctionner en parallèle.

Dans ce module, nous allons aborder, de façon panoramique, les deux techniques pour mettre en évidence les avantages et les limites de chacune de ces techniques et les améliorations possibles.

2 Modèles d'architectures parallèles

Le parallélisme est un mécanisme complexe qui pose beaucoup de défis. Des modèles d'architectures ont été proposés pour simplifier le fonctionnement des ordinateurs parallèles. Une taxonomie de ces modèles a été proposé par Flynn [1]. Cette taxonomie se base principalement sur le type d'organisation du flux de données et du flux d'instructions.

2.1 SISD (Single Instruction Single Data)

Ce modèle d'architecture signifie : unique flux d'instructions, unique flux de données. Il s'agit d'une architecture séquentielle, comme nous l'avons vu jusqu'à

1. `https://fr.wikipedia.org/wiki/Taxonomie_de_Flynn`

maintenant dans les modules de ce cours. Il s'agit bien de l'architecture de Von Neumann. Donc, il n'y a aucun aspect de parallélisme que ce soit au niveau des instructions ou au niveau des données.

Comme nous l'avons vu dans les modules précédents, une seule unité de contrôle (CU) s'occupe de récupérer un unique flux d'instructions à partir de la mémoire. Les anciens ordinateurs monoprocesseurs fonctionnaient avec ce modèle d'architecture.

2.2 SIMD (Single Instruction Multiple Data)

Ce modèle d'architecture signifie : unique flux d'instructions, multiples flux de données. Il s'agit d'une architecture qui exploite le parallélisme au niveau de la mémoire.

Dans ce modèle d'architecture, les instructions sont exécutées séquentiellement, mais peuvent être exécutées selon le principe de pipeline pour améliorer les performances. Nous pouvons trouver ce modèle d'architecture dans les processeurs vectoriels [2] en pipeline.

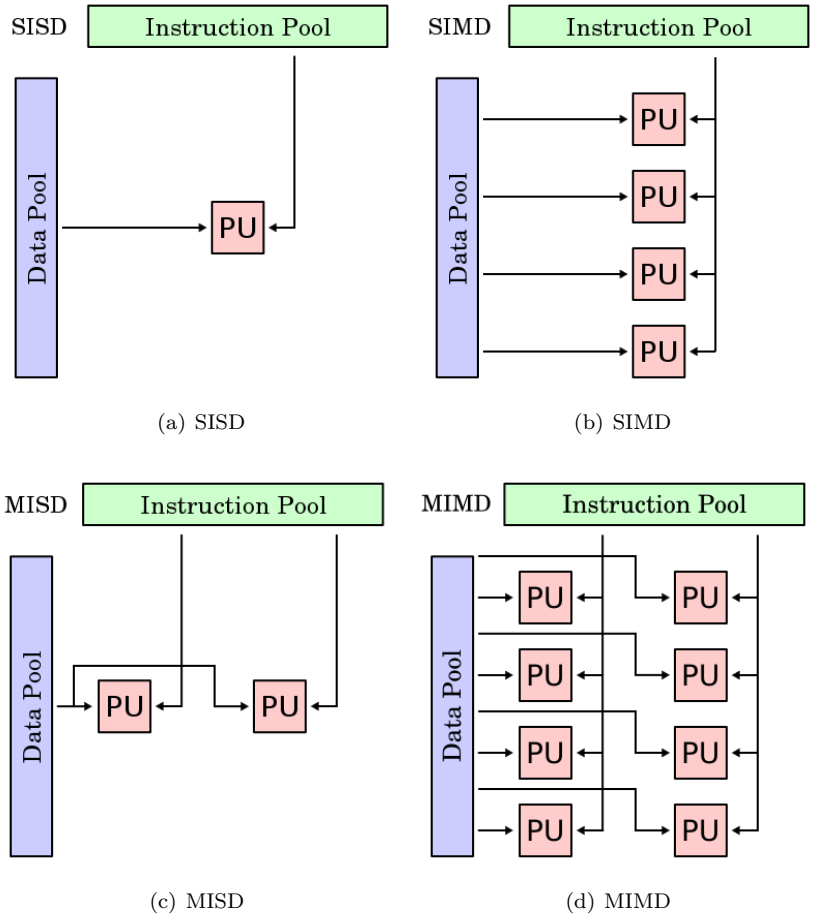
2.3 MISD (Multiple Instruction Single Data)

Ce modèle d'architecture signifie : multiples flux d'instructions, unique flux de données. Il s'agit d'un architecture où la donnée peut être traitée simultanément par plusieurs unités de calcul, c'est-à-dire, en mode parallèle. C'est un modèle d'architecture très peu utilisé en pratique. Nous pouvons trouver ce modèle d'architecture dans les tableaux systoliques (systolic array) [3].

2.4 MIMD (Multiple Instruction Multiple Data)

Ce modèle d'architecture signifie : multiples flux d'instructions, multiples flux de données. Il s'agit d'une architecture où plusieurs unités de calcul traitent des données différentes en parallèle. Dans ce cas, chaque unité de calcul possède sa propre mémoire. Ce modèle d'architecture est le modèle le plus répandu dans la pratique. Nous pouvons trouver ce modèle d'architecture dans les ordinateurs modernes multiprocesseurs. La figure 1 montre graphiquement le fonctionnement de chaque modèle d'architecture [4].

2. `https://fr.wikipedia.org/wiki/Processeur_vectoriel`
3. `https://en.wikipedia.org/wiki/Systolic_array`
4. Figures tirées du Wikipédia.



(a) SISD (b) SIMD

(c) MISD (d) MIMD

FIGURE 1 – Modèles d'architectures.

Depuis quelques années, les architectures parallèles deviennent très populaires et largement utilisées dans l'industrie. Nous pouvons citer par exemple :

— les ordinateurs multi-cores,
— les processeurs graphiques,
— l'informatique en nuage (cloud computing).

Il existe plusieurs techniques pour implémenter le parallélisme. Ces techniques peuvent se résumer dans les points suivants :

— Parallélisme en pipeline (d'instructions)
— Parallélisme de tâches
— Parallélisme de données
— Graphe de dépendance

3 Parallélisme d'instructions

Avant d'aborder le parallélisme d'instructions, rappelons d'abord les étapes requises pour l'exécution d'une instruction. Une instruction est généralement exécutée en cinq (5) étapes :

- **IF** : Instruction Fecht, charge l'instruction à exécuter dans le pipeline.
- **ID** : Instruction Decode/Register Fecht, décode l'instruction et adresse les registres.
- **EX** : Execution/Effective Address, exécute l'instruction (par la ou les unités arithmétiques et logiques).
- **MA** : Memory Access/ Cache Access, dénote un transfert depuis un registre vers la mémoire dans le cas d'une instruction du type STORE (accès en écriture) et de la mémoire vers un registre dans le cas d'un LOAD (accès en lecture).
- **WB** : Write-Back, stocke le résultat dans un registre.

La technique du pipeline permet de superposer les sous cycles d'exécution des instructions. Par exemple, supposons le processeur veut exécuter 5 instructions nommées respectivement i, i+1, i+2, i+3 et i+4. Le pipeline pourrait donc être schématisé comme suit :

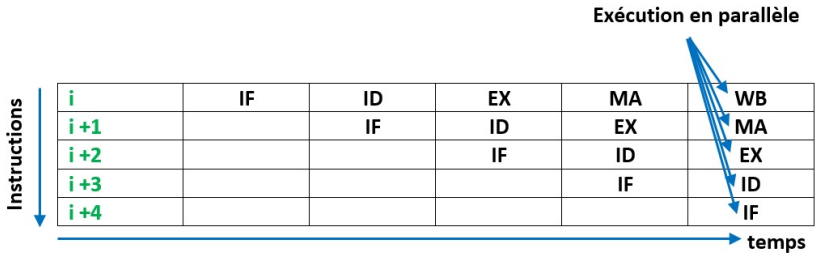


FIGURE 2 – Pipeline.

Notez bien que le processus du pipeline ne modifie pas le temps d'exécution des instructions, mais, il permet d'augmenter le débit (fréquence d'exécution) des instructions. Cela permet donc d'augmenter les performances de l'ordinateur.

Comme vous pouvez le constater dans la figure 2, nous avons plusieurs niveaux (5 niveaux) qui permettent d'exécuter plus d'instructions. Par exemple, dans le Pentium IV, nous avons 20 niveaux. Vous pouvez imaginer comment les performances peuvent être améliorées avec ce nombre de niveaux.

Cependant, la technique du pipeline présente aussi quelques inconvénients. Par exemple, en cas de rupture de séquence, le pipeline doit être vidé. S'il n'est pas vidé, nous aurons un blocage dans l'exécution des instructions. L'autre inconvénient, est que nous pouvons toujours avoir des dépendances entre les instructions dans le pipeline, cela est appelé le problème des **aléas de pipeline** qui peut être divisé en trois sous catégories :

— aléas structurels : qui est engendré par un conflit d'accès à des ressources.
— aléas de données : qui est engendré lorsque nous avons des dépendances de données entre instructions.
— aléa de contrôle : qui est engendré lorsque nous avons une rupture de séquence.

3.1 Aléas structurels

Les aléas structurels ou conflit de ressources se produisent lorsque la même ressource est requise par plusieurs instructions dans des niveaux différents. Par exemple, nous pouvons avoir le scénario de conflit d'accès à la mémoire si le processeur exécute les instructions MA (Memory Access) dans dans des niveaux différents sur la même colonne comme le montre la figure 3.

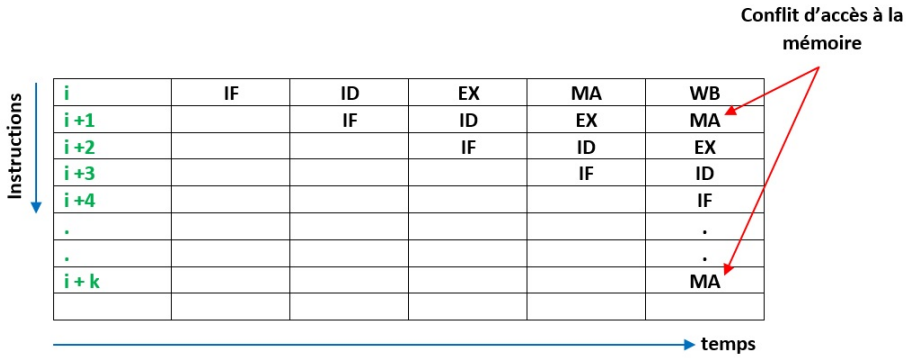


FIGURE 3 – Conflit d'accès à la ressource.

Il existe plusieurs solutions pour faire face au conflit structurel.
— La première solution consiste à réduire le nombre d'instructions par cycle.
— La deuxième solution consiste à utiliser des ressources à accès multiple à savoir l'utilisation des mémoires caches distinctes.

3.2 Aléas de données

Les aléas de données se produisent lorsque plusieurs instructions dans le pipeline présentent des dépendances de registres. Par exemple, ce cas pourrait se présenter lorsqu'une instruction se charge de mettre la valeur de soustraction de deux nombres dans le registre (R1) et une deuxième instruction utilise le résultat dans (R1) pour effectuer une autre opération comme le montre le code assembleur suivant :

```
1 ; Exemple de conflit de données
2 MOV    AL, 25     ; mettre le registre AL à 25.
```

```
3 MOV    BL, 10     ; mettre le registre BL 10.
4 ADD    CL, AL, BL
5 LOAD   DL, CL
```

Dans ce code assembleur, nous pouvons déduire que l'instruction LOAD DL, CL ne peux être exécutée que si l'instruction ADD CL, AL, BL est exécutée. Par conséquent, les deux instructions ont une dépendance au registre CL.

Pour faire face au problème des conflits de données, nous pouvons réduire également le nombre d'instructions par cycle. Cela permettra d'éviter les dépendances entre les registres.

3.3 Aléas de contrôle

Les aléas de contrôle se produisent lors de l'exécution des branchements conditionnels, plus particulièrement lorsque nous voulons charger une instruction dans le pipeline dont la condition n'est pas encore connue.

Vous pouvons regarder dans le code assembleur suivant, comment l'aléa de contrôle pourrait se produire :

```
1 ; Exemple de conflit de contrôle
2 MOV    AL, 25     ; mettre le registre AL à 25.
3 MOV    BL, 10     ; mettre le registre BL 10.
4 CALL branchement
5 LOAD   DL, BL
6 ADD    DL, AL
```

Dans ce code assembleur, les deux instructions LOAD AD, BL et ADD DL, AL se trouvent déjà dans le pipeline avant même l'exécution de l'instruction CALL. Dans ce cas, les conditions ne sont pas satisfaites pour effectuer le branchement ce qui provoque un conflit de contrôle.

Pour éviter que ce type de conflit puisse se reproduire, il est recommandé de vider le pipeline avant de charger de nouvelles instructions.

3.4 Prédiction des branchements

Nous avons vu dans les sections précédentes, que le fait de vider le pipeline pourrait résoudre la plus part des conflits plus particulièrement les conflits de contrôle. Cependant, vider et remplir le pipeline plus souvent affecter les performances du système. Par conséquent, pour éviter de vider et remplir le pipeline, nous pourrons utiliser la méthode de prédiction des branchements.

Les raisons de cette méthode est que les branchements sont très fréquents avec la technique de plusieurs instructions dans le pipeline. Dans l'objectif d'améliorer davantage les performances du systèmes, cette méthode exploite au maximum le principe du parallélisme.

Nous distinguons deux types de prédiction de branchements :

— **Prédiction statique** : consiste à prendre toujours la même décision en cas de branchement. Une des méthodes recommandée pour améliorer les prédictions des branchements, est d'utiliser les profils d'exécutions précédents. La raison d'utiliser les profils d'exécutions est que les comportements des branchements suivent généralement une distribution bimodale comme le montre la figure 4 qui représente le taux de succès des prédictions de branchements avec la technique des profils d'exécutions [5].

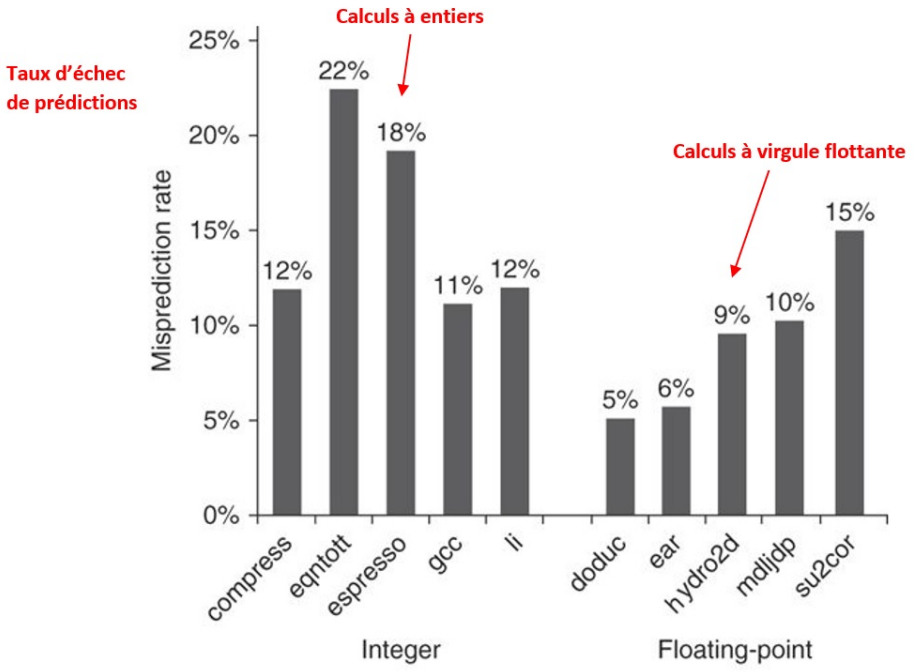


FIGURE 4 – Taux d'échec de prédictions.

— **Prédiction dynamique** : consiste à prendre une décision en fonction des décisions précédentes. Dans la prédiction dynamique, nous utilisons une table d'historique de branchements (branch history table, appelé aussi branch-prediction buffer). Cette table est une petite mémoire indexée par les bits de poids faible de l'adresse de l'instruction de branchement. Cette mémoire contient un bit qui indique si le branchement est récemment pris ou pas.

5. Figure tirée du livre : Computer architecture.