ANNEXE B

Bases du parallélisme

Introduction

Cette annexe suis le même objectif que l'annexe A : rendre cette thèse accessible à tout informaticien. Aucun prérequis n'est nécessaire pour cette introduction au parallélisme. Le lecteur découvrant le parallélisme se doit de lire cette annexe avant le chapitre 2.

B.1 Les machines parallèles selon Flynn

L'univers des machines parallèles est complexe, quelques définitions de base s'imposent. Flynn [Fly66] classifie les machines parallèles selon leur architecture. L'ordinateur est considéré comme une machine qui applique un flux d'instructions sur un flux de données. De cette analyse, viennent quatre architectures dont trois sont couramment utilisées :

- Les ordinateurs SISD (*Single Instruction stream, Single Data stream*) appliquent un flux d'instructions unique sur un flux de données unique. Cette classe est celle des machines séquentielles. La figure B.1 schématise l'architecture SISD de Von Neumann [GF93]. Une unité de contrôle récupère une instruction et ses opérandes en mémoire, pour les envoyer à une unité processeur. Le résultat est éventuellement stocké en mémoire.

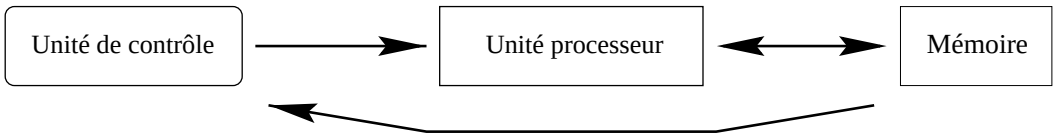

FIG. B.1 – *Architecture séquentielle SISD*

- Les ordinateurs SIMD (*Single Instruction stream, Multiple Data stream*) travaillent avec un flux d'instructions unique sur des flux de données différents. A un instant donné, une même instruction opère sur toutes les données, ou

éventuellement une sous-partie. Cette classe de machines parallèles impose de repenser les algorithmes séquentiels pour atteindre de bonnes performances.

– Les ordinateurs MIMD (*Multiple Instruction stream, Multiple Data stream*) appliquent des flux d'instructions différents, chacun sur un flux de données propre. En pratique, les ordinateurs MIMD sont les plus polyvalents et les plus performants. L'architecture SIMD ne prend le dessus que sur des problèmes réguliers.

B.2 Quelques définitions

Cette section définit quelques termes du parallélisme : la capacité d'extension, l'accélération, la latence et la bande passante, les pipelines, ainsi que la granularité.

B.2.1 Capacité d'évolution (« scalabilité »)

La capacité à généraliser une architecture ou un algorithme à différentes échelles est une caractéristique importante d'une parallélisation. Le monde du parallélisme parle de « scalabilité ».

Une machine multiprocesseur respecte la définition d'une architecture MIMD. Cependant, le principe de fonctionnement est plus proche d'une station de travail monoprocesseur que d'une machine parallèle : un système d'exploitation multitâche attribue les processeurs disponibles à des processus généralement indépendants, de façon transparente pour l'utilisateur. Une telle architecture est rarement étendue à plus de 2 ou 4 processeurs.

B.2.2 Accélération (*Speedup*)

L'accélération quantifie le gain induit par l'utilisation de plusieurs processeurs. Le calcul nécessite la connaissance du temps d'exécution du meilleur programme séquentiel possible (équation B.1). Le but est de juger les performances de la « manière de paralléliser », plus que le résultat brut.

$$Accélération = \frac{Temps\ d'exécution\ du\ meilleur\ algorithme\ séquentiel}{Temps\ d'exécution\ de\ l'algorithme\ parallèle\ évalué} \quad \text{(B.1)}$$

Mesure pratique de l'accélération

Sur une architecture MIMD, une valeur significative de l'accélération peut être mesurée en prenant comme temps séquentiel le temps d'exécution du programme sur un des processeurs de la machine parallèle (équation B.2). Nous utiliserons cette mesure pour juger de la qualité des parallélisations implémentées, et pour les comparer entre elles.

$$Accélération(p) = \frac{Temps\ d'exécution\ du\ programme\ sur\ 1\ processeur}{Temps\ d'exécution\ du\ programme\ sur\ p\ processeurs} \quad \text{(B.2)}$$

Accélération linéaire

On parle d'une « accélération linéaire » lorsqu'elle est égale au nombre de processeurs. Cette situation implique que la charge de travail de chaque processeur est constamment maintenue à son maximum. Peu de programmes parallélisés arrivent à ce niveau de performance car de nombreuses communications sont souvent nécessaires au bon déroulement d'un algorithme parallèle.

Remarque

Notons qu'une accélération « supra-linéaire » (rapport supérieur au nombre de processeurs) est impossible selon la définition donnée par l'équation B.1 car l'on considère *le meilleur algorithme séquentiel*. Cependant, l'accélération peut être supérieure au nombre de processeurs avec la formule B.2 utilisée en pratique. Nous rencontrerons et commenterons ce cas de figure section 4.9.

B.2.3 Latence et bande passante

La qualité d'un système de communication est déterminée par deux paramètres :

- la latence d'une communication (*the startup time*), notée β, est le temps nécessaire à l'initialisation de la communication (la latence est indépendante de la quantité de données) ;
- la bande passante du média (*bandwidth*) est le débit maximal.

Taux de transfert

On utilise aussi souvent l'inverse de la bande passante. Cette quantité est appelée le taux de transfert et est notée τ.

Durée de la communication

La connaissance de la latence et du taux de transfert d'un média permet le calcul du temps T d'une communication. La durée de la communication dépend bien sûr de la quantité de données à transmettre (équation B.3).

$$T = \beta + (quantité)\ \tau \quad \text{(B.3)}$$

On constate que l'on a intérêt à regrouper les données à transmettre dans un même message pour économiser le temps de latence. Cependant, cette stratégie peut impliquer des retards et un compromis doit être trouvé pour arriver aux performances maximales. Les paramètres β et τ variant beaucoup d'un média à un autre, le compromis n'est souvent valable que pour une machine déterminée ce qui nuit à la « portabilité » des programmes développés.

B.2.4 Portabilité

Bien que la notion de portabilité ne soit pas propre au parallélisme, il peut être utile de la préciser. Une solution est dite « portable » si elle peut facilement être réutilisée.

Portabilité d'un langage de programmation

Par exemple, l'utilisation d'un langage éloigné de la machine, on parle de langage de « haut niveau », est un premier gage de portabilité. Ainsi, le « langage C » [KR78, KR88] que nous utilisons pour toutes nos implémentations est plus portable que le langage « assembleur » spécifique à une famille de processeurs, lui-même plus portable que le « langage machine » que comprend le processeur. Le langage C a de plus l'avantage d'être standard et se retrouve sur toute machine moderne. La plupart des systèmes d'exploitation sont écrits en C, notamment pour être eux-mêmes portables. Mais utiliser un langage portable n'est pas suffisant, il faut veiller à respecter une méthodologie rigoureuse.

Portabilité des algorithmes

Au-delà du langage utilisé et de la propreté du programme, il est nécessaire de respecter la philosophie d'une architecture. Dans le cas d'une architecture séquentielle dite de Von Neumann (c'est-à-dire SISD), on s'interdira par exemple d'écrire un programme se modifiant lui-même bien que ce soit possible et potentiellement très pratique. Un tel comportement rendrait le programme moins portable (par exemple si l'ordinateur utilise une « antémémoire d'instructions »).

B.2.5 Mise en œuvre d'un pipeline

Définition

La mise en œuvre d'un pipeline consiste à découper une tâche en sous-tâches (si possible de la même durée), et à exécuter toutes les sous-tâches simultanément. Cette stratégie suppose que la tâche initiale doive être répétée un grand nombre de fois. L'accélération maximale est égale au nombre de sous-tâches, elle diminue si les sous-tâches ne sont pas de la même durée. Cette efficacité n'est atteinte qu'une fois la première tâche terminée, et tant que les tâches s'enchaînent sans contretemps.

Exemple

Une illustration de la mise en œuvre d'un pipeline est le « travail à la chaîne ». Prenons l'exemple d'une chaîne de montage d'automobiles. Le travail est découpé en un maximum d'étapes (n) confiées à autant d'ouvriers afin de maximiser le parallélisme et donc la vitesse de production. Si les étapes n'ont pas toutes la même durée, les ouvriers doivent constamment attendre que leur camarade le plus chargé ait terminé. Appelons t_{max} la durée de l'étape la plus longue. Après un temps $n \times t_{max}$ la première voiture arrive en fin de chaîne, dès lors un temps t_{max} sépare la sortie de deux nouveaux véhicules.

Profitons de cette exemple pour faire quelques remarques utiles sur la mise en œuvre d'un pipeline.

- Tant que la première voiture n'est pas sortie de l'usine, beaucoup d'ouvriers sont inoccupés. Le nombre de voitures à produire doit donc être largement supérieur à n pour que la méthode ait un intérêt. De même, un pipeline n'est efficace qu'une fois « chargé ». En conséquence, un pipeline n'est viable que si le travail doit être répété un grand nombre de fois.
- Une accélération linéaire (c'est-à-dire égale à n) n'est possible que si à aucun moment un ouvrier est inoccupé.

B.2.6 Granularité et parallélisme

Le grain d'un algorithme

Le niveau de parallélisme d'un algorithme est défini par sa granularité. Le grain le plus fin d'un algorithme fait s'exécuter de façon concurrente toute suite d'instructions inséparables. Un grain plus important fait travailler simultanément des blocs d'instructions plus longs. Un algorithme à gros grain - ou à grain grossier - parallélise une application en exécutant des processus complets.

Le grain d'une architecture

La notion de granularité se retrouve au niveau matériel. Un grain fin correspond à de petits processeurs dont le faible coût permet une utilisation massive. Un fort grain met en œuvre de gros processeurs, généralement en nombre limité. Bien que la définition de Flynn ne précise pas la puissance des processeurs, les machines SIMD ont généralement une faible granularité alors que les machines MIMD utilise les plus puissants processeurs séquentiels disponibles sur le marché.

Le grain d'un réseau de neurones

La granularité d'un réseau de neurones n'est pas définie dans la littérature connexionniste. Bien que la thèse détaille cette notion, il est intéressant d'essayer de définir dès à présent ce qu'est la granularité d'un modèle connexionniste.

- Le grain le plus fin d'une exécution parallèle correspond au parallélisme intrinsèque des réseaux de neurones. Toutes les cellules travaillent simultanément. Le classifieur incrémental a été conçu pour être parallélisé à ce niveau. Hélas, une telle approche est irréalisable sur une machine MIMD moderne, avec un faible nombre de processeurs puissants.
- Le grain le plus élevé correspond au fonctionnement concourant de plusieurs réseaux complets. Cette stratégie est mise en œuvre dans la thèse au travers de plusieurs parallélisation.