Contents

0.1 2.5 Architectures hybrides

Mémoire partagée combinée avec mémoire distribuée (Multicoeurs interconnectés entre eux...)

On peut aussi voir des GPU comme noeud de calcul.

On peut alors combiner MPI et openMP (ou C++17)

0.1.1 GPU

C'est un modèle mixte entre SIMD et MIMD. On a une collection de processeurs SIMD, chacun pouvant exécuter un autre code, de façon asynchrone. Les processeurs SIMD sont à mémoire partagée.

0.2 2.6 Architecture vectorielle

Actuellement, les processeurs modernes ont des unités dites vectorielles qui permettent de traîter quelques éléments d'un vecteur en même temps. (Pour nous, c'est plutôt une approche SIMD...) Nous on parlerait plutôt d'unités SIMD avec un faible degré de parallélisme.

Du point de vue historique, les architectures vectorielles se distinguent par:

- Registres vectoriels
- Exécution en pipeline
- Unités d'exécution multiples (C'était pour les supers ordinateurs des années 1980 à 2000)

0.2.1 Registre vectoriel

On peut accéder en même temps à plusieurs données en mémoire et les avoir dans le CPU dans des registres vectoriels. (Soucis: vitesse de chargement de la mémoire vers le cpu...) Dans les machines haut de gamme, ces registres avaient des tailles de 128, 256 ou 512 mots.

Exemple du gain dû à ces registres:

$$T_{seq} = NT_{fetch} + NT_{cpu} + NT_{store}$$

avec N le nombre de données et T_{fetch} le temsp de lecture mémoire

$$T_{vector} = T_{fetch} + NT_{cpu} + T_{store}$$

Gain:

$$\frac{T_{seq}}{T_{vector}} = \frac{NT_{fetch} + \cdots +}{T_{fetch} + \cdots +} \approx \frac{N(T_{fetch} + T_{cpu} + T_{store})}{NT_{cpu}} \approx T_{fetch} + \frac{T_{store}}{T_{cpu}} \Rightarrow >> 1$$

$$\Leftrightarrow T_{fetch} >> T_{cpu}$$

0.2.2 Exécution en pipeline

Toutes les unités de calcul sont divisées en q étages

$$T_{seq} = Nq\tau$$

$$T_{vector} = q\tau + (N-1)\tau$$

($q\tau$: première donnée... $(N-1)\tau$: données suivantes) Si N est grand, on a un speedup de q.

0.2.3 Parallélisme grâce à plusieurs unités de calcul

(voir figure polycopié)

0.3 2.7 Architecture Data-Flow

Habituellement, les ordinateurs sont "control-driven" ou "instructions-driven". On peut aussi avoir un modèle "Data-driven".

On a une donnée a, une donnée b. Une fois qu'elles sont disponibles, on active une opération. La disponibilité des opérandes active l'exécution de l'instruction qui les combine.

0.3.1 Variante: Demand-driven

?c = a + b

On va aller chercher les instructions a et b pour les exécuter

C'est le besoin du résultat (c en l'occurence) qui déclenche le calcul des opérandes.

L'intérêt du data-flow est que le parallélisme présent dans les calculs est automatiquement exploité.

0.4 2.8 Parallélisme interne (ILP)

On parle aussi de **ILP: instruction level parallelism**

Dans les processeurs modernes, il y a plusieurs instructions qui sont exécutées en même temps. Cela prend plusieurs formes:

- Pipeline d'exécution: fetch (prend l'instruction en mémoire) -> decode -> execute -> write-bach (résultat dans les registres)

  Avec ce pipeline, plusieurs instructions sont traitées en même temps, bien qu'à des stades différents. On dit alors que le CPI = 1 (Cycle Per Instruction)

Un aspect ILP est la précence de plusieurs unités d'exécution utilisables en même temps

- On a une approche qu'on peut désigner comme "**very long instruction word**": on sépare l'instruction en plusieurs sous instructions. C'est le compilateur qui construit ce VLIW (very long instruction word). Si nécessaire, certaines de ces instrucitons sont des NOP (no operation) s'il n'y a pas de parallélisme à disposition, ou s'il y a des dépendances.
- **Il y a aussi une version plus élaborée, appelée superscalavité où c'est le matériell directement qui gère le parallélisme, au moyen du principe "data-flow":** On a des stations de réservation d'instruction. Les instructions viennent s'empiler dans les stations de réservation correspondantes et elles sont exécutées dès que leurs opérandes sont disponibles. Les dépendances entre données sont résolues par le fait que si les données sont disponibles, elles sont utilisables.

Il est important de se rendre compte que certaines instructions, comme un branchement, peuvent être problématiques car elles peuvent casser le pipeline.

→ **Branchement spéculatif**: processeur fait un choix aléatoire de quelle branche il va prendre, et s'il se trompe, il retourne en arrière. Branchement spéculatif statique: deviner quelle branche on va prendre grâce à la façon de programmer des personnes. Techniques plus dynamiques: sur chaque branchement, on prend celle de gauche ou celle de droite, et chaque fois qu'on réussi, on augmente les chances de prendre le côté qu'on avait choisit et vice-versa.

sisc vs risk:

- sisc instructions peut faire plusieurs choses à la fois.
- risk: on fait des instructions simples et si on veut faire une chose plus complexe, on fait une combinaison d'instructions simples. Tendance d'avoir plus d'instructions par cycles plutôt que d'avoir des instructions complexes par cycle...

1 Réseaux d'interconnexions

1.1 3.1 Introduction

C'est ce qui permet aux processeurs de communiquer, et c'est donc une partie essentielle d'une architecture parallèle, si on veut des performances.

On va voir qu'il y a des réseaux d'interconnexion dits statiques ou dynamiques. (statiques fait penser plutôt à un graphe, alors que dynamiques fait plutôt penser à un gros switch ou commutateur.)

1.1.1 Valeurs de communication

Bande passante CPU-mémoire: 37GB/s en 2012 vs 95 GB/s en 2020

Temps de latence: 150 ns (2012) vs 50 ns (2020)

Liens entre PE:

- 2GB/s par liens
- 10 liens par PE (C'était IBM BGQ en 2012)

Infiniband: 100GB/s latence 1 μs (comme dans baobab)

Il faut aussi se rappeler que dans un ordinateur, il y a une hiérarchie de composantes, et des vitesses qui le reflètent.

registres -> mémoire cache -> mémoire centrale -> interconnexion entre PE -> accès disque

1.1.2 Propriétés associées aux réseaux d'interconnexion

- **Diamètre**: Distance qui sépare des noeud les plus éloignés, mesuré en nombre d'étapes pour y accéder.

  C'est la distance entre les 2 noeuds les plus éloignés, compté en nombre de noeuds à traverser. Ce qui nous intéresse, c'est le lien entre le diamètre et le nombre n de PE $O(f(n))$

- **Latence**: temps pour accéder à un composant plus le temps pour préparer le message. (cela se mesure en μs)

- **Bandwidth ou bande passante**: mesure le débit de l'échange des données (GB/s)
- **Largeur bisectionnelle**: c'est le nombre de liens qui relie les 2 moitiés d'une même machine. (Il faut s'imaginer qu'on sectionne une ville en deux par une rivière, et on compte le nombre de ponts faisant le lien entre les deux parties de la ville) Ici aussi on s'intéresse à cette valeur de façon asymptotique avec n le nombre de noeuds.
- **Degré**: le nombre de liens qui part de chaque PE vers un autre.
- **Prix ou complexité**: c'est la relation entre le prix et le nombre de PE. Prix = O(n)
- **Scalabilité (passage à l'échelle)**: C'est le lien entre le nombre de PE et la performance espérée. On aimerait que cela soit proportionnel.

Ces réseaux sont caractérisés par des **algorithmes de routage**.

Ces algorithmes doivent exploiter la topologie d'interconnexion pour être optimaux.

- Ils sont asynchrones (Chaque message va à son propre rythme...)
- Ils doivent garantir l'absence de "deadlocks"
- Ils peuvent être multi-port (potentiellement, plusieurs messages en même temps sur tous les liens) ou single-port (un port à la fois).