Contents

0.1 Hypercube suite

Deux noeuds voisins sur l'hypercube ont une adresse quei ne diffère que par le bit dans la direction de connexion.

0.1.1 Algorithme de routage

Soit l'adresse d'origine et l'adresse de destination:

On fait un xor de ces deux adresses et cela va nous donner le chemin à suivre. Un 0 dans la dimension i indique qu'il ne faut pas la traverser. Et le 1 veut dire de la traverser. Ex:

```
001
010
---
011
```

Il faut traverser les dimensions 1 et 2, mais pas la 3. L'ordre n'a pas d'importance et cela permet plusieurs chemins optimaux.

0.1.1.1 Broadcast

Temps pour faire un broadcast:

$$T_{broadcast} = O(d) = O(log_2(n))$$

C'est un résultat dont on est content !

C'est la référence de l'efficacité

0.1.1.2 Echange total

Chacun fait un broadcast

On échange les données présentes avec son voisin dans la dimension 2

Puis on échange les données présentes dans la dimension 2 (on a 2 valeurs maintenant....)

Donc temps de l'échange :

$$T_{all-to-all} = \sum_{i=1}^{d} 2^{i-1} = 2^d - 1 = O(N)$$

(Taille du message qui augmente d'un facteur 2 à chaque étape)

C'est aussi la référence en échange total ! L'échange total devrait prendre un temps linéaire avec le nombre de processeurs.

Quelles données va-t-on mettre sur quel processeur ?

0.2 Mapping d'une structure de donnée sur un hypercube

Comment distribuer les données sur les différents processeurs (décomposition du domaine ou partitionnement), avec la contrainte que les données devant être échangées soient sur des processeurs voisins ?

Ici, on va considérer le cas du mapping d'une grille (tableau, matrice, ...) sur un hypercube. Ici, on va demander que les données voisines sur la grille soient placées sur des processeurs voisins.

On va considérer une grille 1D avec 2^d élémentss.

L'élément i de la grille sera placée dans le processeur n où $n = G(i, d)$ est une fonction de Grey réfléchie.

Regarder exemple shéma....

Par construction de cette réflexion, on obtient le résultat escompté, ce qu'on démontre par induction: Si cela marche en d, cela marche en d + 1

Le résultat est qu'on a un peu enroulé notre donnée dans l'hypercube... On voit que 7 et 0 sont aussi voisins sur l'hypercube.

0.3 Mapping d'une grille 2D sur un hypercube

Soit un tableau (grille) de taille $2^r \times 2^s$ plongée dans un hypercuve de dimension $d = r + s$

Le point de coordonnée (i, j) de la grille sera assigné au processeur $n = G(i, r)||G(j, s)$.

$||$: concaténation des adresses $G(i, r)$ et $G(j, s)$.

r = s = 2. tableau de taille 4x4

L'élément i, j de ce tableau sera l'adresse de processeur qui le contient

	0	1	2	3
0	0000	0001	0011	0010
1	0100	0101	0111	0110
2	1100	1101	1111	1110
3	1000	1001	1011	1010

Donc on voit que (i, j) est placé sur un processeur dont les voisins contiennent (i +- 1, j) et (i, j +- 1)

0.4 Réseaux dynamiques

Le but est de relier les processeurs et/ou bancs de mémoire à travers un switch ou commutateur

Il y a 3 familles de réseaux dynamiques:

1. connexion par bus: les processeurs sont connectés sur des bus qui sont connectés à la mémoire

   C'est une approche non scallable limitée à peu de processeurs. La bande passante est partagée parmi les n processeurs. Donc, par processeur, la bande passante est $\frac{\omega}{n}$

   Par ailleurs, il faut arbitrer l'accès au bus su plusieurs processeurs veulent l'utiliser en même temps.

   C'est une solution bon marché, mais peu adaptée au parallélisme...

2. Crossbar switch C'est la solution idéale, mais chère et donc pratique sur de petits systèmes.

   On a n inputs et n outputs. Le but est de pouvoir connecter n'importe quelle entrée à n'importe quelle sortie, dynamiquement. Et ceci, pour toutes les entrées, simultanément. $i \to f(i) \forall i$ pour autant que f soit une bijection (sinon on aurait plusieurs entrées sur la même sortie $\Rightarrow$ conflit et arbitrage)

   Combien y a-t-il d'états possibles pour un crossbar ???

   il y a n! patterns de communication = nombre de permutations des entrées vers les sorties

0.4.1 Exemple d'un crossbar simple

On veut donner une réalisation d'un crossbar 3×3 (3 entrées, 3 sorties) Pour cela, on va construire un réseau carré de commutateurs 2×2

Regarder shéma cours...

Si on veut connecter E_i avec S_j, il faut que le commutateur (i, j) soit dans l'état 1, et les commutateurs avec i' > i soient dans l'état 0 et ceux avec j' < j aussi.

On place un 1 dans la première lighe: n possibilitées

Ensuite un 1 dans la deuxième ligne: n - 1 possibilitées etc ...

=> on a bien n! possibilitées

Cette architecture ne gère pas les conflits potentiels si 2 entrées souhaitent se connecter à la même sortie. Donc ces commutateurs doivent inclure de la logique d'arbitrage, de la mémoire, etc...

Cette architecture est donc chère car elle demande $O(n^2)$ éléments et son coût croît comme le carré du nombre d'entrées-sorties.

3. Réseaux multi étages C'est un compromis performance-prix qui permet d'avoir des switches de grande taille à des prix acceptables.

0.4.1.1 Exemple: le réseau Oméga regarder shéma

On peut en choisissant l'état des commutateurs internes connecter n'importe quelle entrée à n'importe quelle sortie

Algorithme de routage: Soit $d_1, d_2, \ldots, d_n$ les d bits qui représentent l'adresse de destination et $o_1, o_2, \ldots, o_n$ ceux de l'adresse d'origine.

Le chemin est défini ainsi: à l'étape k, on prend le k ème bit le plus significatif de la destination: si c'est 1 on sort par la sortie inférieure du commutateur et si c'est 0 par la sortie supérieure.

0.4.1.1.1 Exemple: on veut aller de 1 à 5. $5 = 101$.

On a un algorithme qui ne dépend que de la destination...

Il faut connaître la règle de communication:

```
0 -> 0
1 -> 2
2 -> 4
3 -> 6
4 -> 1
5 -> 3
6 -> 5
7 -> 7
```