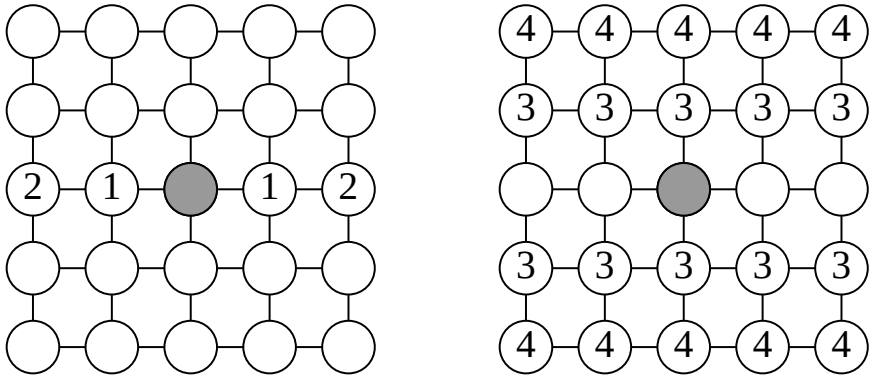

FIGURE 3.10 – *Broadcast sur une grille de processeurs bidimensionnelle, réalisé en distribuant d'abord le message le long de la ligne du processeur source (image de gauche), puis le long des colonnes, simultanément par tous les processeurs de la ligne (image de droite). Les nombres indiquent l'étape de temps à laquelle le message est reçu.*

processeurs plus proches voisins, grâce à des connexions selon les diagonales. A chaque intersection de la grille en X se trouve un noeud tridirectionnel permettant de rediriger un message vers l'un des quelconques trois voisins possibles.

3.2.3 L'hypercube

Le réseau en hypercube représente un moyen très courant pour relier entre eux un ensemble de processeurs. Cette topologie est utilisée dans la Connection Machine en particulier, mais est aussi une option importante pour les architectures MIMD. C'était sans doute le réseau le plus populaire dans les années 80.

Au sens mathématique, un hypercube désigne la généralisation d'un cube dans des espaces de dimension supérieure à 3. Dans notre contexte, ce terme suppose que les côtés sont de longueur 2. Ainsi, dans un hypercube de dimension k, on a exactement $N = 2^k$ processeurs, chacun de degré k (c'est-à-dire liés à k autres noeuds du réseau). Des exemples de réseaux hypercubiques de dimension $k = 0, 1, 2, 3$ sont donnés sur la figure 3.12. La figure 3.13 montre un hypercube de dimension $k = 4$.

On constate que le réseau hypercube se construit récursivement de la manière suivante. Un hypercube de dimension k se construit à partir de deux hypercubes de dimension $k - 1$. On relie ensuite les sommets correspondants de ces deux hypercubes.

Un hypercube de dimension k est donc une grille de dimension k et d'arêtes égales à 2. Cependant, la différence dans la discussion que l'on fait des propriétés de ces deux types de réseaux provient du fait que, pour l'hypercube, on considère une arête de taille fixe (=2) avec une dimension k qui varie, alors que pour la

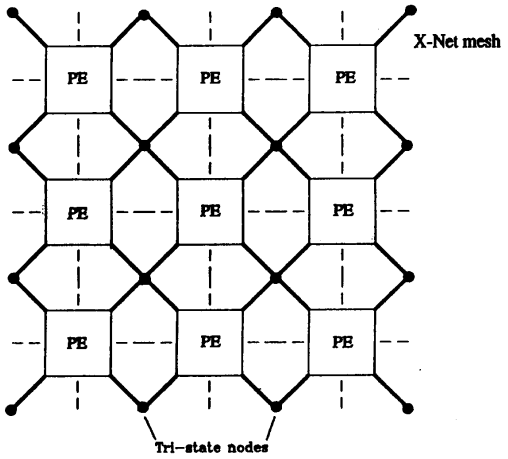

FIGURE 3.11 – *Réseau X-net de la MasPar (Past, Present and Parallel, fig. 2.3, p. 30).*

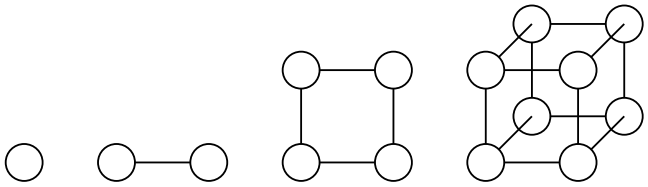

FIGURE 3.12 – *Hypercubes de dimension 0, 1,2 et 3.*

grille, on a une dimension d donnée et on varie la taille des arêtes.

Le diamètre D d'un hypercube est égal à sa dimension k car c'est le nombre d'arêtes qu'il faut suivre pour relier les deux noeuds les plus éloignés. La largeur bissectionnelle vaut $2^{k-1} = N/2$, soit la taille d'un des deux sous-cubes de dimension $k-1$ utilisés pour la construction de l'hypercube de dimension k.

Le prix d'un réseau en hypercube est proportionnel à $kN = N \log_2 N$ car $k = \log_2 N$ est le nombre de connexions par noeuds et N le nombre de noeuds.

La topologie en hypercube n'est pas véritablement scalable, malgré le fait que l'on peut envisager des hypercubes de taille arbitraire. Le problème est qu'en agrandissant un hypercube, on augmente la connectivité de chaque noeud, ce qui n'est évidement pas possible pour des processeurs donnés qui ont un nombre fixe de liens vers l'extérieur. On peut éviter ce problème en plaçant en chaque noeud de l'hypercube une chaîne cyclique de k processeurs (*cube connected cycles*), plutôt qu'un processeur unique. Avec des processeurs ayant trois liens extérieurs, on peut former un anneau de k processeurs ayant exactement les k liens extérieurs nécessaires aux noeuds de l'hypercube. De cette manière, on garde la plupart des avantages de l'hypercube tout en assurant sa scalabilité. Dans ce qui suit, nous nous contenterons de discuter l'hypercube simple.

La structure en hypercube a plusieurs avantages. Premièrement, le nombre

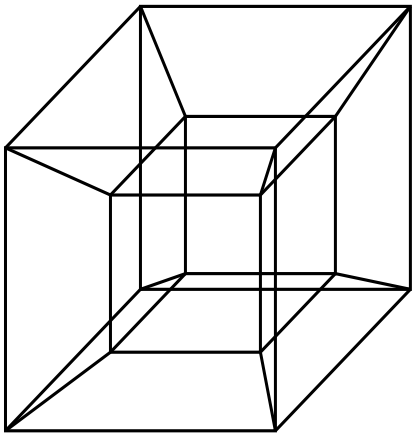

FIGURE 3.13 – *Hypercube de dimension 4.*

de noeuds du réseau croît comme une puissance de 2 du nombre de connexions. Avec une augmentation faible du nombre de connexions, on augmente de façon significative la taille du système. Deuxièmement, le nombre de chemins disponibles entre 2 noeuds augmente à mesure que l'hypercube s'agrandit, ce qui aide à éviter une congestion du trafic des messages.

Routage Il existe une manière efficace de diriger un message à travers un réseau hypercubique. Pour cela, on utilise tout d'abord une numérotation adéquate des noeuds. Cette numérotation se base sur la représentation binaire des nombres de 0 à $2^k - 1$. Chaque noeud est représenté par un nombre de k bits. On choisit arbitrairement le noeud initial qui reçoit l'adresse 0. Ensuite, on applique l'algorithme suivant : deux noeuds qui sont reliés le long de la dimension ℓ de l'hypercube ont une adresse qui ne diffère que par le ℓ^{eme} bit. Cette numérotation est illustrée dans la figure 3.14. Il faut remarquer que cette numérotation correspond aussi au coordonnées cartésiennes dans un espace de dimension k pour lequel les deux seules valeurs possibles dans chaque direction sont 0 et 1.

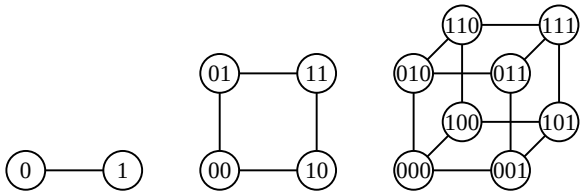

FIGURE 3.14 – *Numérotation des noeuds dans un hypercube : les noeuds voisins dans la direction i ont une adresse qui ne diffère que par le bit i.*

Le chemin que doit parcourir un message devant transiter entre deux noeuds est obtenu par un XOR (ou exclusif) des adresses respectives. Cette opération donne en effet un 1 dans chaque dimension à suivre pour atteindre la destination

désirée. Cela se comprend facilement dans le cas à trois dimensions de la figure 3.14. Pour aller du noeud 001 au noeud 111, il faut voyager dans la direction 1 puis la direction 2 (001 XOR 111 =110), ou inversement. Plus généralement, l'algorithme de transfert de messages ("routing algorithm") dans un hypercube de dimension k contient k étapes. Durant l'étape i, le message est transmis dans la direction i si le XOR de l'adresse contient un 1 en position i. Sinon, le message reste là où il est et attend l'étape suivante. Cela montre que le temps de transit d'un message par cette méthode est proportionnel à k, ou encore au logarithme du nombre N de processeurs ($k = \log_2(2^k)$).

Exemple de broadcast sur un hypercube

La figure 3.15 présente une opération de broadcast sur un hypercube de dimension $k = 4$, à partir du noeud racine (000). On constate que $k = \log_2 N$ étapes sont suffisantes pour que le message initial atteigne tous les noeuds. C'est donc beaucoup plus efficace qu'un broadcast sur un anneau qui requiert $N/2$ étapes.

L'algorithme de broadcast présenté ici est le suivant : un message arrivant de la dimension i de l'hypercube est redistribué simultanément dans les dimensions $i-1$, $i-2$,...,1. C'est un algorithme qui implique des communications multi-liens (ou multi-port) puisqu'un même processeur envoie simultanément un message dans plusieurs directions.

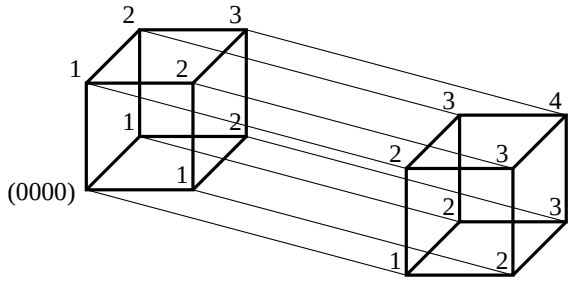

FIGURE 3.15 – *Broadcast sur un hypercube. Le noeud source (0000) envoie le même message dans les quatre dimensions de l'hypercube. Ensuite, chaque noeud redistribue le message selon l'algorithme décrit dans le texte. Les nombres indiquent l'étape de temps à laquelle le message atteint chaque noeud.*

On peut aussi imaginer un algorithme qui n'utilise qu'un lien à la fois (single port). Il est illustré sur la figure 3.16 et nécessite aussi autant d'étapes qu'il y a de dimension dans l'hypercube (il est donc logarithmique dans le nombre de processeurs). Le principe de cet algorithme de broadcast est le suivant : à la première étape, le processeur racine envoie le message dans le direction 1. Durant la deuxième étape, le processeur racine et celui qui a reçu le message à l'étape précédente envoient tous deux le message dans le direction 2 de l'hypercube. A ce stade, 4 processeurs sont "touchés." A l'étape suivante, ces 4 processeurs envoient

le message dans la direction 3 et ainsi de suite jusqu'à ce que tous les processeurs aient reçu le message d'origine.

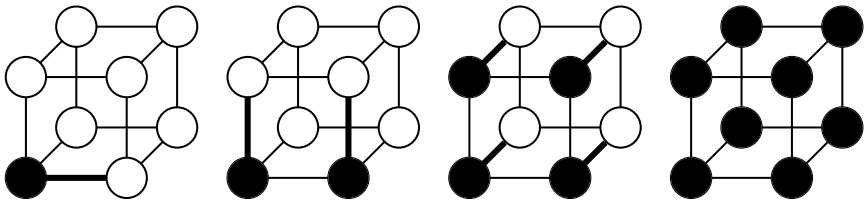

FIGURE 3.16 – *Broadcast sur un hypercube en utilisant des communications "single port." A l'étape i, tous les processeurs ayant reçu copie du message initial (il y en a 2^{i-1}) le renvoient dans la direction i de l'hypercube. Après $k = \log_2 N$ étapes, le broadcast est terminé.*

Exemple d'échange total sur un hypercube

Dans une opération d'échange total, chaque noeud envoie une donnée différente à tous les autres noeuds. A la fin de l'échange, chaque noeud contiendra autant de données qu'il y a de processeurs. Une méthode simple pour réaliser un échange total sur un hypercube est de procéder à des échanges de paires de données, successivement le long de chaque dimension de l'hypercube. A la première étape, chaque noeud envoie une seule donnée au noeud voisin dans la dimension 1. A la ℓ^{eme} étape, chaque noeud envoie son contenu initial, plus toute les données reçues aux étapes précédentes. Sur un hypercube de dimension k, il faut k étapes pour achever un tel échange total. Comme le nombre de données à communiquer au cours de l'étape i vaut 2^{i-1}, le temps total T de la communication est

$$T = \sum_{i=1}^{k} 2^{i-1} = 2^k - 1 = N - 1$$

ce qui est proportionnel au nombre de processeurs. Il existe des techniques plus complexes et plus rapides permettant de réaliser un échange total en distribuant les données simultanément dans toutes les dimensions de l'hypercube.

Mapping d'une grille sur un hypercube

D'une manière générale, le problème du mapping est de plonger une structure de données sur un réseau de communication physique ayant une topologie différente. On aimerait par exemple répartir un arbre sur une topologie en hypercube, tout en préservant la connectivité naturelle de l'arbre.

Un autre exemple courant est le mapping d'une grille sur un hypercube, de sorte que les plus proches voisins sur la grille soient répartis sur des noeuds

adjacents de l'hypercube. Cette contrainte est importante car nombreux sont les algorithmes numériques utilisant des données disposées selon une grille de points et nécessitant des échanges entre points voisins.

De façon générale, ce mapping est réalisé à l'aide d'une fonction G appelée code de Gray (binary reflected Gray code).

Considérons tout d'abord le cas d'une grille unidimensionnelle (vecteur) composée de 2^d points, numérotés $0, 1, 2, ..., 2^d - 1$. On peut associer chaque point i du vecteur à un processeur n dans un hypercube de dimension d par la relation

$$n = G(i, d)$$

où la fonction G est définie récursivement par

$$G(0, 1) = 0 \qquad G(1, 1) = 1$$

pour $d = 1$ et

$$G(i, d+1) = \begin{cases} G(i, d) & \text{pour } i < 2^d \\ 2^d + G(2^{d+1} - 1 - i, d) & \text{pour } i \geq 2^d \end{cases}$$

pour les dimensions supérieures. La numérotation des processeurs dans l'hypercube se fait selon le codage standard : deux processeurs voisins le long de la dimension $k \leq d$ ont une adresse qui ne diffère que par le $k$ème bit.

La définition de G indique que lorsqu'on considère un hypercube de dimension $d + 1$, les 2^d premiers points du vecteur sont envoyés sur le sous hypercube de dimension inférieure d. Les suivants sont envoyés sur les processeurs dont l'adresse s'obtient par "réflexion", selon la construction illustrée sur la figure 3.17. On constate que ce mapping conduit naturellement à des conditions aux bords périodiques puisque le dernier point du vecteur est connecté au premier et que, par construction, les points voisins sur le vecteur sont placés sur des processeurs dont l'adresse ne diffère que par un seul bit (voir figure 3.18).

Le mapping d'une grille de dimension supérieure se fait par une extension de la méthode du code de Gray que l'on vient de voir. Illustrons ceci pour le cas d'une grille rectangulaire bidimensionnelle de taille $2^r \times 2^s$ plongée dans un hypercube de dimension 2^{r+s}. Le point de coordonnée (i, j) de la grille est assigné au processeur d'adresse $G(i, r)\|G(j, s)$ où l'opération $\|$ dénote la concaténation des deux codes de Gray. Comme chacun des codes de Gray préserve la connectivité à une dimension, on a aussi que les points (i, j) et $(i \pm 1, j)$ ou $(i, j \pm 1)$ sont adjacents sur l'hypercube.

Par exemple, pour un tableau de taille 4×8, l'élément $(2, 4)$ est placé sur le processeur d'adresse 11110 d'un hypercube de dimension 5 ($11110 = 11\|110$ qui sont respectivement les code de Gray $G(2, 2)$ et $G(4, 3)$).

Le mapping d'un tableau 4×4 est illustré ci-dessous. Dans chaque élément du tableau figure un nombre indiquant l'adresse en binaire du processeur où il

1-bit Gray code	2-bit Gray code	3-bit Gray code	3-D hypercube	8-processor ring
0	0 0	0 0 0	0	0
1	0 1	0 0 1	1	1
	1 1	0 1 1	3	2
	1 0	0 1 0	2	3
		1 1 0	6	4
		1 1 1	7	5
		1 0 1	5	6
		1 0 0	4	7

FIGURE 3.17 – *Construction récursive du code de Gray réfléchi. Pour passer de la dimension d à la dimension $d+1$ on répète le mapping obtenu (régions ombrées) et on le réplique par symétrie par rapport aux lignes pointillées. On complète le mapping en faisant précéder le partie supérieure par un 0 et la partie inférieure par un 1.*

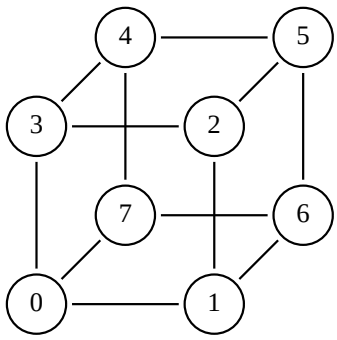

FIGURE 3.18 – *Mapping, par le code de Gray, d'un anneau à 8 noeuds sur un hypercube de dimension 3.*

sera placé.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0000 & 0001 & 0011 & 0010 \\ 0100 & 0101 & 0111 & 0110 \\ 1100 & 1101 & 1111 & 1110 \\ 1000 & 1001 & 1011 & 1010 \end{pmatrix} \end{array}$$

On remarque que ce mapping a de plus la propriété que les processeurs associés à une même ligne du tableau ont les deux bits d'adresse les plus significatifs identiques, alors que ceux qui correspondent à une même colonne ont les même deux derniers bits.

Les grilles dont la dimension n'est pas une puissance de 2 peuvent être plongées dans un hypercube de la même façon à condition d'être élargie à la puissance de 2 supérieure. Cependant, dans ce cas, on perd les conditions aux bords périodiques.

3.2.4 Les arbres binaires

Les réseaux en arbre binaire statiques sont peu utilisés dans les architectures parallèles. Cependant, une version améliorée est à la base du réseau d'interconnexion dit *fat tree* qui est un arbre dynamique que nous étudierons au paragraphe 3.3.3.

Pour l'instant, nous discutons les propriétés élémentaires d'un arbre binaire statique. Plusieurs d'entre elles se généraliseront aisément au cas du fat tree.

Le schéma d'un tel réseau est donné sur la figure 3.19. A l'exception des extrémités, chaque noeud est relié à 3 autres noeuds, un ancêtre et deux descendants. Chaque ligne horizontale de processeurs est appelée niveau ou étage de l'arbre. Le niveau ℓ ($\ell = 0, k$) contient 2^ℓ noeuds et le nombre total de noeuds est $N = 2^0 + 2^1 + 2^2 + ... + 2^k = 2^{k+1} - 1$.

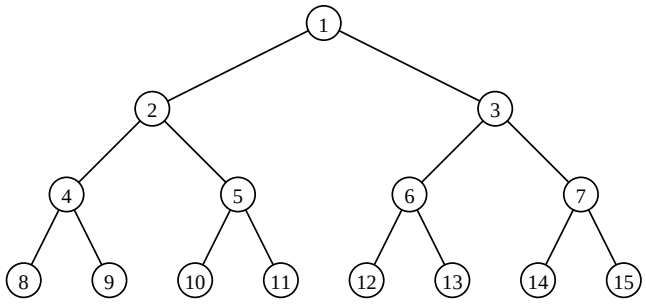

FIGURE 3.19 – *Topologie d'arbre binaire statique.*

Le diamètre d'un arbre binaire est $D = 2k$, car au pire, il faut remonter jusqu'à la racine et redescendre jusqu'en bas. La largeur bissectionnelle vaut 1, ce qui est le défaut majeur de cette architecture : la racine constitue un "bottleneck"