Remerciements

Mes vifs remerciements sont d'abord adressés à monsieur le professeur ***BETTAYEB Saïd*** *qui m'a fait l'honneur de diriger ce travail de recherche. Je tiens à lui exprimer ma gratitude et mon profond respect.*

Mes vifs remerciements à monsieur le professeur ***ZERARKA Abderwaheb*** *qui m'avez aussi honoré de guider ce travail, collaborer à son enrichissement par ses directions judicieuses et son soutien constant.*

Je tiens également à exprimer une reconnaissance aux membres de jury :

- Monsieur le professeur ***Mohamed Tayeb LASKRI*** *de m'avoir honoré de présider le jury de la soutenance.*

- Monsieur le professeur ***Mokhtar SELLAMI,*** *Monsieur le maître de conférence* ***Okba KAZAR*** *et le docteur chargé de cours* ***Belkacem BOUCHANA*** *d'avoir bien accepté d'examiner le contenu du présent travail.*

Tous mes collègues de département d'informatique de l'université Mohamed KHIDER Biskra particulièrement Monsieur ***Khaled REZEG****,* ***Dr Bachir BENSALAH, Kamel MOUADAA*** *et* ***Louardi ZERIBI.*** *A vous tous, j'adresses mes remerciements empressés et le témoignage de ma sincère et fidèle amitiée.*

Dédicace

Je dédie ce travail

à la mémoire de mes parents ;

à ma femme

à mes enfants Mohamed Achraf, Amira Cherifa et Mohamed Wail ;

à ma toute ma famille et ma belle famille ;

à mes neveux ;

à tous mes amis.

Résumé.

Obtenir une grande performance pour la résolution d'un seul problème, plusieurs processeurs doivent coopérer entre eux-mêmes afin d'assurer une bonne communication. Ces processeurs doivent utiliser des réseaux d'interconnexion ou des bus. Les machines parallèles sont classées selon leur flot de données ou selon leur flot d'instructions. Où ces processeurs s'échangent des messages en utilisant une mémoire commune ou plusieurs mémoires.

Dans notre travail de recherche nous nous focalisant essentiellement sur la fonction d'inter-échange des messages entre les différents processus des différents processeurs. Les machines parallèle basées sur la topologie d'hypercube ont obtenu un grand respect dans le calcul parallèle parce qu'ils ont plusieurs propriétés très attractives. Plusieurs versions de l'hypercube sont introduites par plusieurs recherches, principalement celles qui améliorent la communication. L'hypercube et l'une des versions les plus attractive, cet non seulement elle préserve les meilleures propriétés, mais aussi elle réduit le diamètre par un facteur égal à deux.

Dans ce mémoire nous montrons la capacité de l'hypercube croisé de simuler une architecture maille par l'approche du plongement one by one et le plongement many by one en utilisant deux fonctions de bonne qualité a savoir : une dilatation égale à deux, une expansion égale à un, une congestion égale à deux et un facteur de charge égal à deux.

Abstract.

To obtain a great performance, many processors are allowed to cooperate to solve a single problem. These processors communicate via an interconnection network or a bus. Parallel machines are classified as either message passing machines where processors have their own memory or shared memory machines where several processors share the same memory. In this work we focus on the former. The most essential function of underlying interconnection network is the efficient interchanging of messages between processes in different processors. Parallel machines based on the hypercube topology have gained a great respect in parallel computation because of its many attractive properties. Many versions of the hypercube have been introduced by many researchers mainly to enhance communications. The twisted hypercube is one of the most attractive versions of the hypercube. It preserves the important features of the hypercube and reduces its diameter by a factor of two. In this dissertation we show the ability of twisted hypercube to simulate the meshes architecture by the embedding One by one and many by one with a good qualities dilation two, expansion one, the congestion two and load factor one.

LISTE FIGURES

Chapitre 1 : Architecture parallèle

Chapitre 2 : L'hypercube croisé

Chapitre 03 : Le plongement

Chapitre 04 : Simulation du plongement de la maille dans l'hypercube croisé

SOMMAIRE

Chapitre 1 : Architecture parallèle

Chapitre 2 : L'hypercube croisé

Chapitre 3 : Le plongement

Chapitre 4 : Simulation de plongement de la maille dans l'hypercube croisé

Chapitre 5 : Implémentation de la simulation du plongement de la maille dans l'hypercube croisé

Introduction Générale

La demande de puissance de calcul n'a cessé de croître depuis le début de l'âge informatique. En effet, des nouvelles applications qui surgissent dans plusieurs domaines poussent la conception des circuits aux limites physiques. L'industrie informatique a montré une augmentation constante dans les puissances des processeurs. La haute technologie des circuits intégrés (VLSI) ne peut satisfaire la croissante demande de calcul dans nombreuses filières scientifiques et techniques telles que le traitement des images et les prévisions météorologiques. Sans les ordinateurs de haute puissance, plusieurs de ces défis ne peuvent avoir des solutions dans un temps acceptable.

Dans les deux dernières décennies, les progrès dans la technologie des circuits intégrés a entraîné à la fabrication de processeurs d'une très haute puissance, à un coût bas et d'une très petite dimension. Ceci a donné lieu à la fabrication des ordinateurs parallèles qui sont composés d'un grand nombre de processeurs. L'idée du parallélisme n'est pas nouvelle, mais sa déviation du modèle traditionnel de Van Newman a introduit d'autres problèmes tels que le débit des liens de communications qui, faute d'absence d'algorithmes de communication efficaces, peut entraîner la dégradation de l'ordinateur parallèle.

Les deux principales composantes d'un ordinateur parallèle sont les processeurs et le réseau d'interconnexion qui lie ces processeurs. Plusieurs réseaux ont été suggérés et étudiés dans la littérature. Des ordinateurs parallèles dont les réseaux d'interconnexion ayant les topologies de l'hypercube, de la maille, de papillons (butterfly), qui ont été fabriquées et sont disponibles dans le marché. Des algorithmes parallèles ont été conçus dans plusieurs domaines pour des machines parallèles basées sur une topologie particulière. Supposons que plusieurs algorithmes qui résolvent un ensemble de problèmes ont été conçus pour une topologie A et il existe de nouvelles topologies plus robustes, attractives et populaires. Donc au lieu de concevoir d'autres algorithmes qui résolvent ces mêmes problèmes pour une autre topologie B, il serait désirable de trouver un moyen de traduire ces algorithmes. Ceci est accompli par le plongement de A dans B.

Dans ce travail de recherche, nous essayerons de développer cette idée dans le cas d'une topologie source maille et d'une topologie hôte hypercube croisé.

Ce mémoire est composé de cinq chapitres dont le premier consiste à présenter un état de l'art sur l'architecture parallèle selon la classification de Flynn et la classification topologique et le routage des données.

Dans le second chapitre nous étudierons l'architecture hypercube croisé, ses propriétés, sa capacité de simuler d'autres architectures, le routage de données et les opérations de base dans cette architecture.

Le troisième chapitre sera réservé aux différentes définitions du plongement et ces mesures de qualité. Un état de l'art sur les différentes architectures plongeables dans l'hypercube croisé fera partie également du contenu de ce chapitre.

Dans le chapitre quatre nous proposerons deux solutions du plongement de l'architecture maille dans un hypercube croisé à savoir :

- ✓ Le plongement many by one avec une qualité de plongement de dilatation égale à deux (02), d'une expansion égale à un (01) et d'un facteur de charge égal à deux (02) considéré comme solution optimale d'un plongement.
- ✓ Le plongement one by one avec une qualité de plongement de dilatation égale à deux (02), d'une expansion égale à presque un (01) et d'une congestion égale à deux (02).

Le chapitre cinq contient l'implémentation de la solution proposée et les tests réalisés sur cette dernière.

Enfin une conclusion générale et des perspectives termineront notre mémoire.

Motivation

Dans les machines parallèle a accès aléatoire (PRAM) un modèle théorique standard est utilisé dans le calcul parallèle .la machine PRAM est synchrone avec un nombre de processeurs identique et une mémoire globale avec une lecture et écriture simultanée dan la même mémoire. [GA,AG89],[MQ,ND84],[EU82] .ces machines ont la capacité d' exécuter très rapidement des algorithmes en temps réel. les machines actuel basé sur le PRAM sont construite vu qu'il offre un temps d'accès très signifiant. le meilleur des délai des temps d'accès est proportionnel a logN avec un nombre de processeurs égale à N [GA,AG89]. L'exemple classique le plu connue en architecture et le MARK-Comic Cube.

La machine réseau de connexion cube et les mailles basé respectivement sur le tri de Batcher et le tri de Kung [CT,HK77],Nassimi et Sahni ont prouvé que la lecture a accès aléatoire (RAR) est accompli avec une complexité O(q2n) pour la maille O(log2N) pour le

réseau de connexion cube ,l'écriture a accès aléatoire et accompli avec une complexité égale à O(q2n+dqn) dans la maille de dimension et une complexité égale à O(log2 N+dlogN) sur le réseau de connexion cube ou shuffle machine pour chaque processeur. Plusieurs chercheurs ont concentrés leur travaux sur la recherche de nouvelles simulation sur des machines PRAM . la première pour les machines déterministe est proposé par Upfal et Wigderson [EU,AW87] ,leur simulation s'est terminé avec O(log2NloglogN) fois pour une étape d'un algorithme PRAM pour une machine a N processeurs et Alt et al. [HA,TH,KM,FP87] ont démontré que la complexité et de O(log2N).

Valiant[LV82]reporte les algorithmes de routage non déterministe qui font des permutations sur les machines hypercube de taille N en un nombre d'étape égale à O(logN) . L'algorithme est compose de deux phases. La première consiste à envoyer p paquet à v nœud choisi aléatoirement ou le choix est équiprobable est égale a 1/N et le choix entre les paquets est indépendant. Dans la deuxième phase, le routage des paquets se fait via des nœuds intermédiaire v. à chaque instant il y'a exactement une copie pour chaque paquet. La transmission des paquets le long de la liaison entre les nœuds se fait à partir d'une file d'attente associé à chaque liaison est ceci pour les paquets perdu entre nœud intermédiaire. L'algorithme travail d'une façon synchrone, il utilise l'alternance entre deux nœuds le mode de transmission est celui de la transmission de chaque paquet tête de file d'attente associé à chaque liaison. Le deuxième mode est celui de la transmission des paquets perdu à la reception.

Valiant prouve que l'algorithme de la distribution aléatoire peut assurer le routage des paquets à d'autres destination dans l'hypercube avec deux dans la même liaison au même temps avec O(logN) et aussi avec une grade probabilité. Chaque paquet peut porter O(logN)bits d'information sans avoir recours à un autre nœud . ce résultat implique que l'hypercube peut simuler une machine PRAM avec une augmentation du temps d'exécution pour chaque étape qui elle-même peut être simuler à approximativement O(logn) étapes pour un réseau de connexion bounded degré de taille N.

1-INTRODUCTION

Dés le premier âge de l'informatique, les demandes d'accélérations du traitement de l'information n'ont cessé de s'améliorer de plus en plus. Cependant, malgré le développement croissant et important de la technologie VLSI, qui a crée des nouveaux processeurs qui sont très rapides, miniaturisés de taille petite, avec un coût très réduit. Les performances de ces processeurs dans les systèmes monoprocesseur, même les plus sophistiqués, ne parviennent pas à satisfaire la grande variété des problèmes scientifiques qui nécessitent un temps de résolution (exécution) très important.

Les concepteurs d'ordinateur ont essayé de recourir à l'amélioration de l'architecture de nouvelles machines dont le but de répondre aux insuffisances des machines monoprocesseurs par la réduction du temps de calcul aux maximum, en créant des nouvelles structures architecturales, utilisant l'architecture parallèle. Cette technologie consiste à faire travailler plusieurs processeurs en même temps afin de réaliser l'exécution d'une seule tâche.

2-Classification de Flynn

Les futurs ordinateurs, toutes tailles confondues, profiteront encore plus du parallélisme qu'ils ne le font aujourd'hui, on pense que l'exploitation des machines parallèles ouvrira dans le courant de la décennie des possibilités extraordinaires. Ceux qui comprennent les applications, les algorithmes et l'architecture seront préparés à cette opportunité.

Puisque le parallélisme peut apparaître à beaucoup de niveaux, il est utile de classer les différentes possibilités. En 1966, Flynn a proposé un modèle simple de classement des ordinateurs, qui est toujours utile aujourd'hui.

En examinant attentivement la composante la plus contrainte de la machine, Flynn a compté le nombre de flots parallèles d'instructions et de données, puis classé l'ordinateur en fonction du résultat obtenu : **[EA93]**

- Un seul flot d'instructions, un seul flot de données
  (**SISD**: Single Instruction Stream, Sing Data Streams. Le monoprocesseur)
- Un seul flot d'instruction, plusieurs flots de données
  (**SIMD**: Single Instruction Stream,Multiple Data Streams)
- Plusieurs flots d'instructions, un seul flot de données

(**MISD**: Multiple Instructions Streams, Single Data Stream)

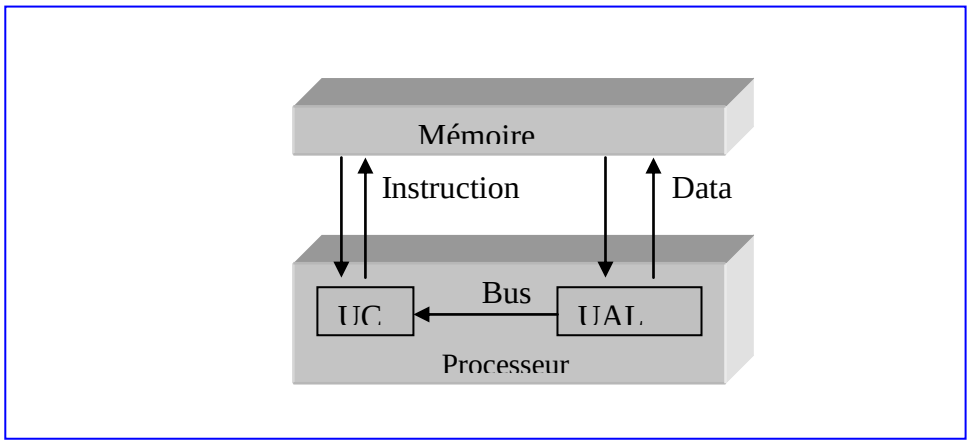
- Plusieurs flots d'instructions, plusieurs flots de données

(**MIMD**: Multiple Instructions Streams, Multiple Data Streams).

2.1-Le modèle SISD

C'est le modèle qui représente le type d'architecture de van Neuman. Il se caractérisé par une mémoire et un processeur unique, ce dernier est formé d'une unité de contrôle et d'une unité de traitement. Dans ce modèle, il n'existe pas de parallélisme.

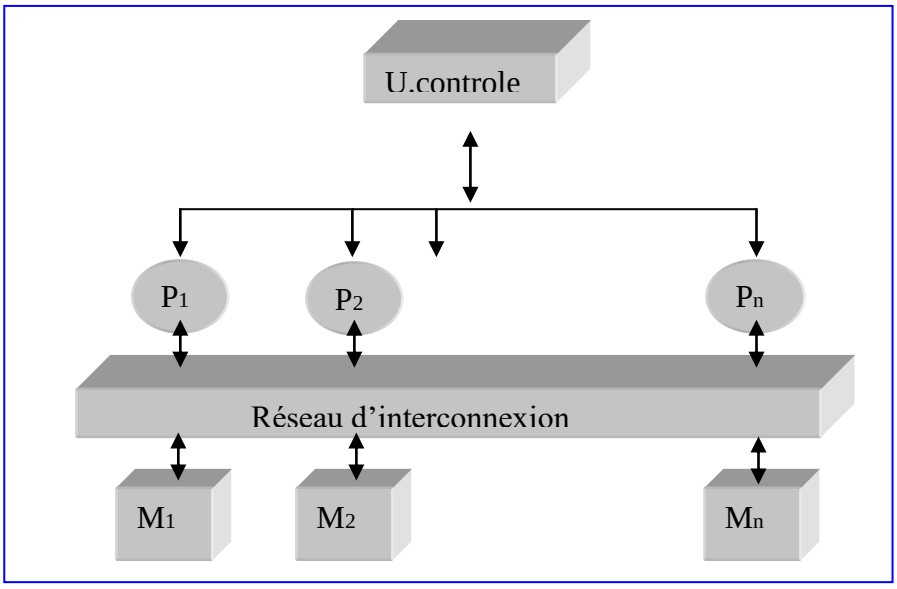
L'unité de contrôle lit dans la mémoire les instructions de programme à exécuter et donne des ordres à l'unité de traitement, celle-ci effectue les opérations nécessaires sur les données stockées également dans la mémoire. **[DT00]**



Architecture SISD

2.2- Le modèle SIMD

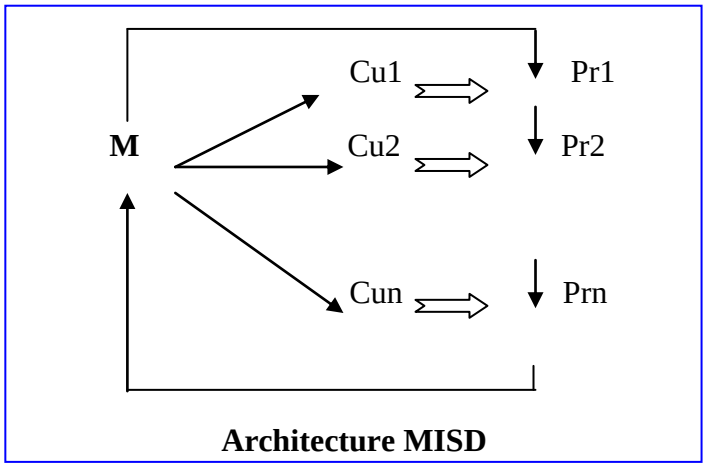
Dans l'architecture de ce modèle, on trouve plusieurs unités de traitement supervisées par la même unité de contrôle. Toutes les unités de traitement reçoivent la même instruction diffusée par l'unité de contrôle, et opèrent sur des ensembles de données distincts, provenant de flot de données distinctes. Chaque unité de traitement exécute la même instruction au même instant. Ceci permet d'obtenir un fonctionnement synchrone des processeurs. La mémoire partagé peut être subdivisée en plusieurs modules. Dans ce cas, l'accès des unités de traitement aux différents modules se fait par un réseau d'interconnexion. Ce modèle réalise le parallélisme entre les différentes instructions non identiques mais bien adapter aux traitements d'opérations vectorielles. **[MC93]**



Architecture SIMD

2.3- Le modèle MISD

Une machine à base d'architecture MISD peut exécuter plusieurs instructions en même temps sur la même donnée. Cette machine n'est pas commercialisée puisque la structure MISD n'a jamais été implantée « réellement » **[EA93]**



Architecture MISD

2.4- Le modèle MIMD

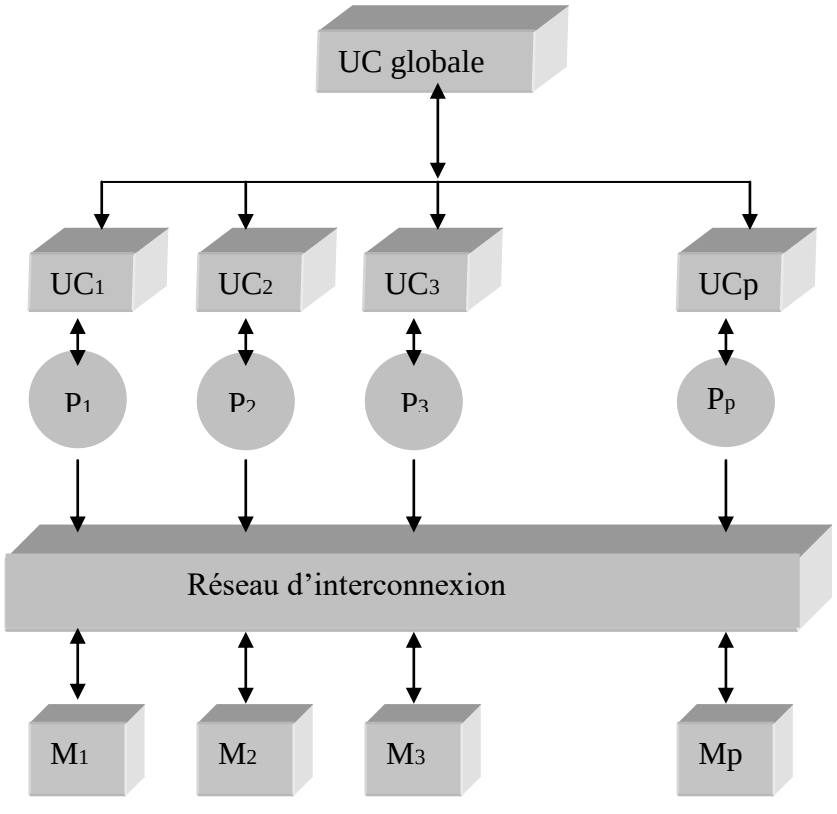
C'est une machine a N processeurs qui exécutent simultanément des séquences d'instructions différentes sur des données différentes. Il existe deux types de machine MIMD :

2.4.1-Le modèle MIMD à mémoire partagée

Dans une machine à base d'architecture MIMD, le fonctionnement de chaque processeur évolue indépendamment des autres et les échanges d'informations entre les processeurs se font par l'intermédiaire de la mémoire commune. D'une manière générale, l'accès s'effectue en lecture simultanée et l'écriture est exclusive. En fait, le partage de la mémoire dans les machines de type MIMD limite le nombre de tâches que ces machines peuvent effectuer en même temps. **[MC93]**

2.4.2- Le modèle MIMD a mémoire distribuée

Dans ce cas, on associe à chaque processeur une mémoire locale (registres ou mémoires caches), la mémoire est divisée en un nombre de modules distincts, chacun a son canal d'E/S. Ces canaux sont reliés aux processeurs par l'intermédiaire d'un réseau d'interconnexion, c'est à dire l'interaction des processeurs est assurée par la transmission de message à travers le réseau. **[EA93]**



Architecture MIMD a mémoire distribuée

3-Classification topologique

Cette classification est composée de trois classes de réseaux de connexion, à savoir :

a) Les réseaux simples.

b) Les réseaux hybrides.

c) Les réseaux hypercubes.

3.1- Les réseaux de connexions simple.

Dans cette partie, on essayera de définir l'architecture parallèle simple avec ses deux types à savoir : l'architecture parallèle simple avec un réseau d'interconnexion vecteur linéaire et l' architecture parallèle simple avec un réseau d'interconnexion d'arbre binaire complet.

Ils sont définis par les connexions entre les différentes paires de processeurs. Ces connexions ne varient pas avec le temps notamment :

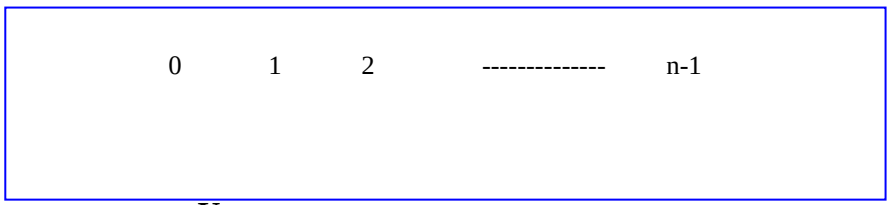
1. architecture vectorielle.
2. architecture arbre binaire complet.

3.1.1-Architecture vectorielle

3.1.1.1-Les vecteurs

Les vecteurs ont une représentation structurale qui regroupe un ensemble de processeurs alignés. Chaque processeur à une liaison bidimensionnelle, avec son voisin à droite (prédécesseur) et son voisin à gauche (un successeur), sauf pour ceux situés les plus à l'extérieur ayant une seule liaison et qui vont servir d'E/S dans le réseau.

Ce genre de réseau est très convenable puisque les processeurs qui forment ce réseau n'ont besoin que de deux ports pour les liaisons avec le reste du réseau. On peut donc faire agrandir le réseau comme on veut. On n'est pas limité par le nombre de ports que les processeurs possèdent. Il n'y a aucune restriction sur le nombre de processeurs qu'un tel réseau peut avoir.



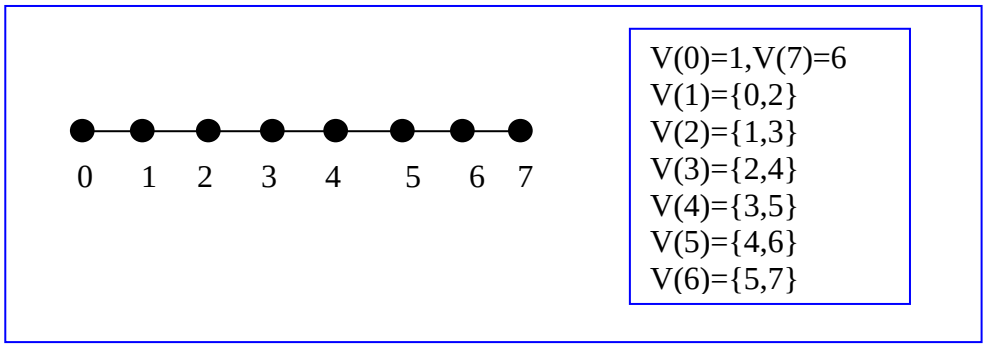
Un vecteur avec n-processeur

3.1.1.2-Réseaux d'interconnexions linéaires et anneaux

Les topologies en réseau linéaire et anneau ont été très utilisées dès l'apparition des multiprocesseurs à mémoire distribuée. Dans un réseau d'interconnexion linéaire, les nœuds

sont ordonnés par ordre croissant de leurs numéros, de 0 à (p-1). Chaque nœud possède deux voisins, son prédécesseur et son successeur. Le voisinage du nœud i pour i=1,2,….(p-2) est v(i) = {i-1,i+1}, à l'exception du premier et du dernier qui n'ont chacun qu'un seul voisin v(0) = 1, v(p-1)={p-2}. Le diamètre du réseau linéaire est égal à (p-1) **[MC93]**.

Exemple :

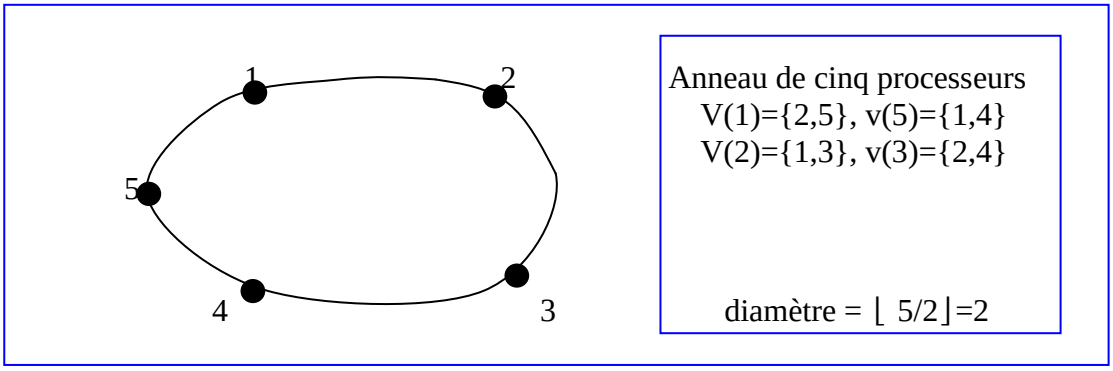


Réseau d'interconnexion linéaire
à huit processeurs
Diamètre =7

Dans un anneau, tous les nœuds possèdent exactement deux voisins, le premier et le dernier sont reliés entre eux :

V (i)= {i-1 modulo p, i+1 modulo p} pour tout i=0,….p-1 Le diamètre d'un anneau est égale à ⌊ p/2⌋ **[MC93]**.

Exemple :



Anneau de cinq processeurs.

3.1.1.2- Maille (Composition de vecteurs)

Nous combinons N vecteurs linéaires de k-cellules pour former un vecteur k*N (ou maille). Cette composition de vecteurs peut être faite en appliquant l'effet miroir. En

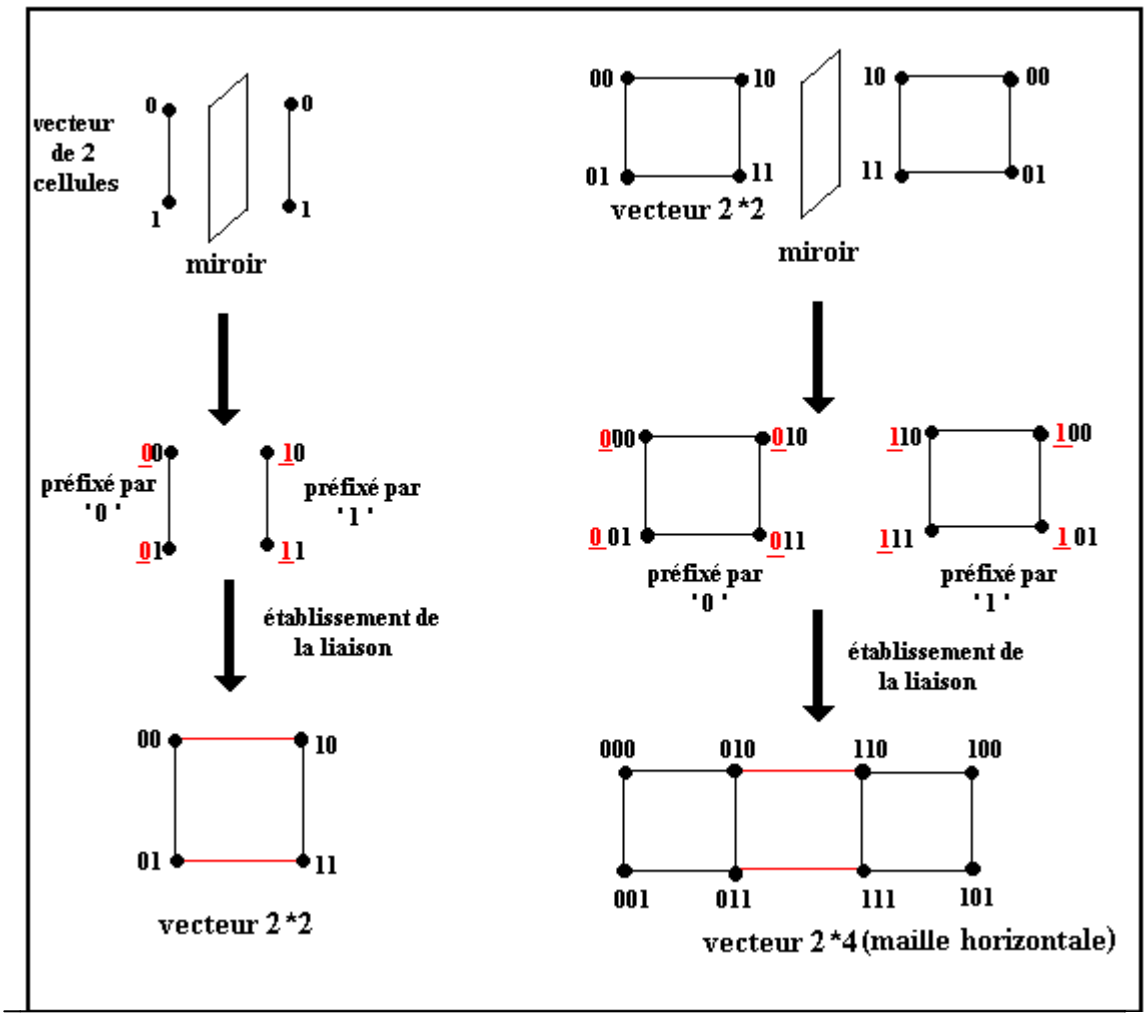
appliquant l'effet miroir sur un vecteur k-cellules, on obtient un autre vecteur similaire. On préfixe toutes les étiquettes des nœuds de la première copie par « 0 » et toutes les étiquettes des nœuds de l'autre copie par « 1 ». La liaison entre les deux copies est établie seulement entre k-nœuds de chaque copie à condition que les étiquettes des nœuds de l'une des copies diffèrent avec les étiquettes des nœuds de l'autre copie en un seul bit.

On a trois cas à étudier :

Premier cas :

Si on applique l'effet miroir sur un vecteur de k-cellules en utilisant un générateur horizontal, on obtient un autre vecteur similaire. Si on répète ce procédé plusieurs fois, on va obtenir une maille k*N rectangulaire horizontale.

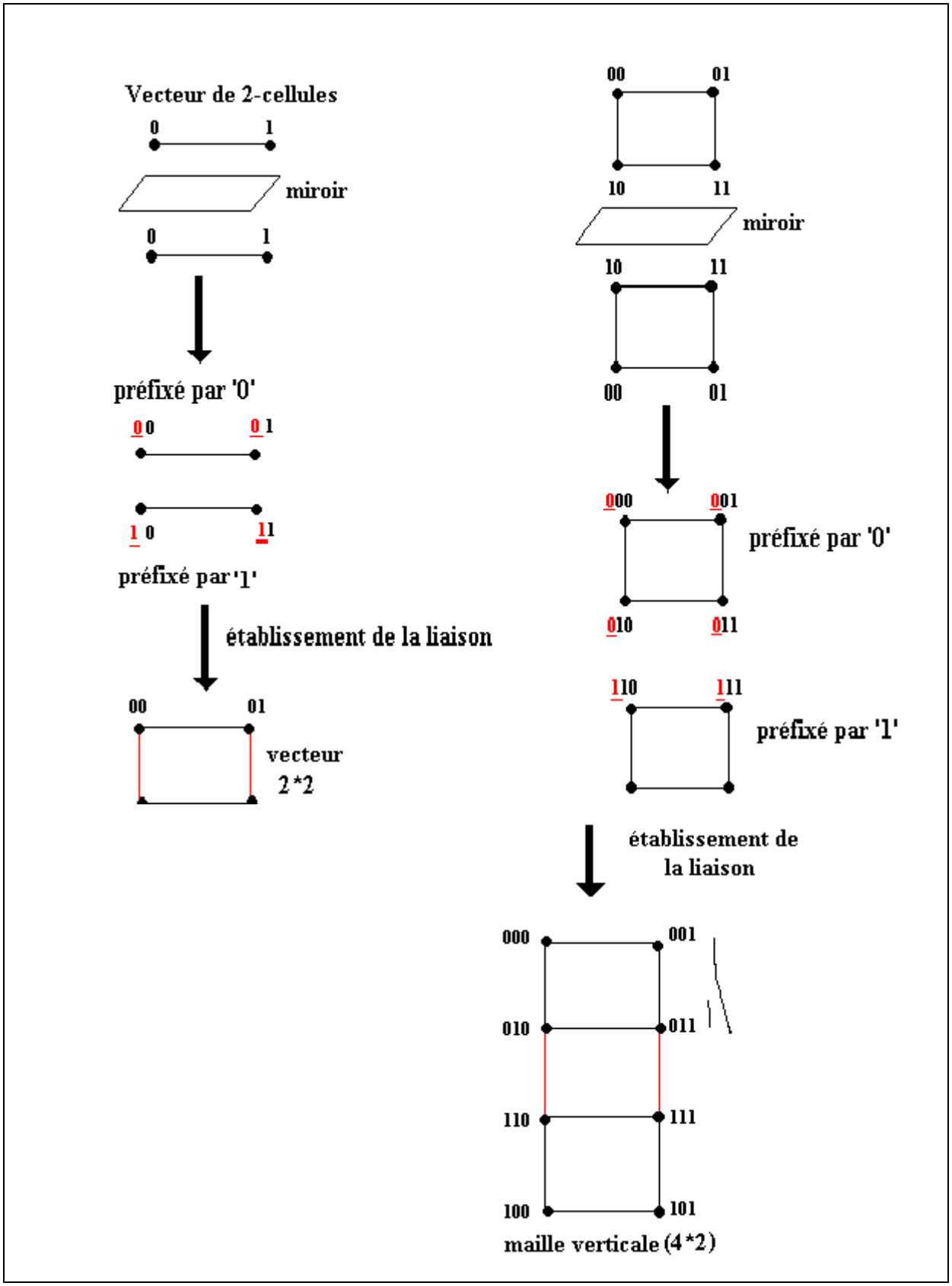
Exemple : **k= 2, N=4 (maille horizontale)**



Deuxième cas :

Si on applique l'effet miroir sur un vecteur de k-cellules en utilisant un générateur vertical, on obtient un autre vecteur similaire. Si on répète ce procédé plusieurs fois, on obtient une maille k*N rectangulaire verticale.

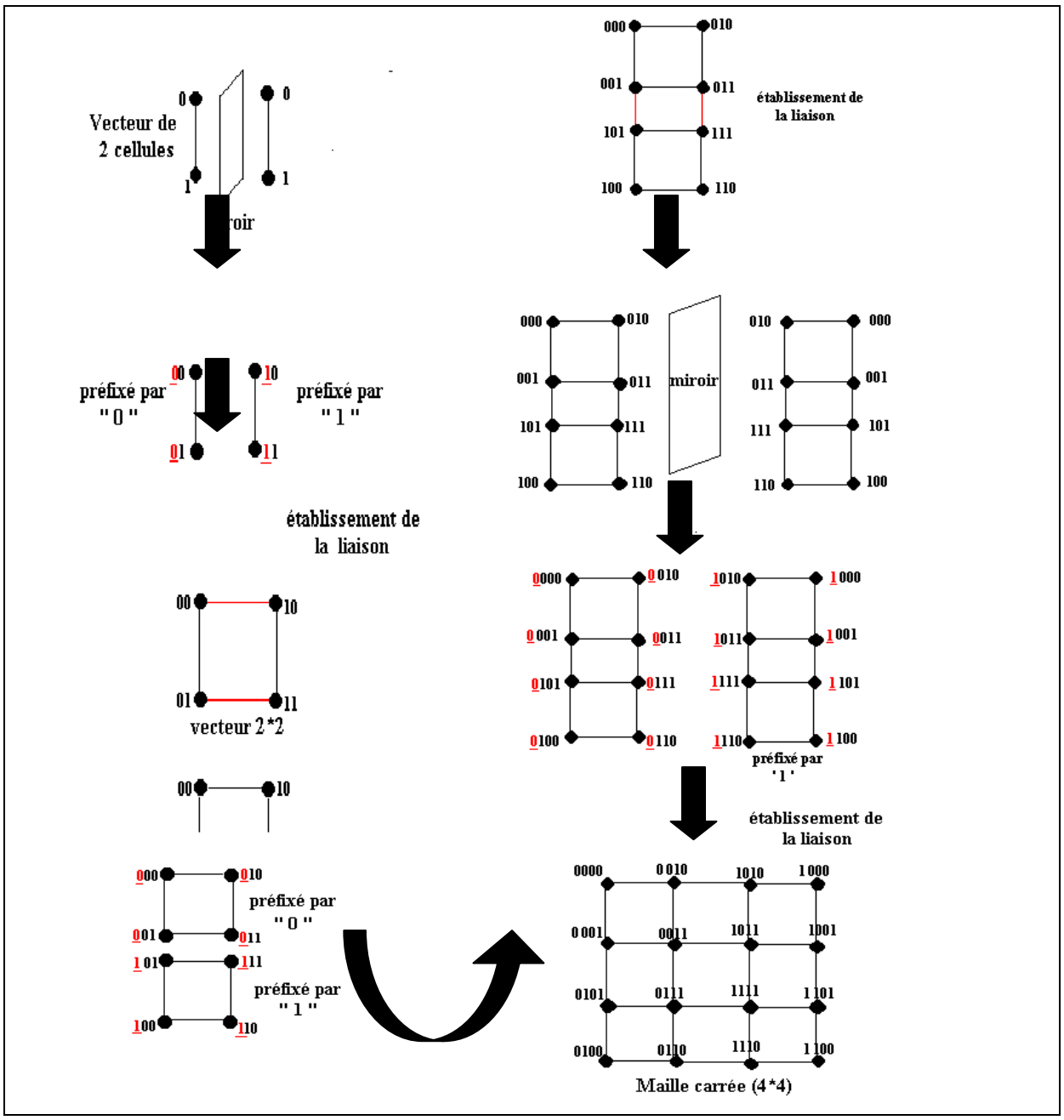
Exemple : **k=2, N=4 (maille verticale)**



Troisième cas :

Si on applique l'effet miroir sur un vecteur k-cellules en utilisant un générateur hybride (vertical et horizontal), on obtient une maille carrée N*N.

Exemple : **k = N = 4**



Une maille de taille (N*M) comprend N lignes de M processeurs. Chaque processeur a donc quatre voisins à l'exception des processeurs situés sur les premières et dernières lignes et colonnes. Si l'on utilise le couple d'indices (i, j) pour représenter le $j^{ième}$ processeur de la $i^{ième}$ ligne (en partant de (0, 0)), on obtient :

- Pour les sommets intérieurs, i = 1,2,…,N-2 et j = 1,2,…,M-2 :
  V (i, j) = {(i-1,j), (i+1,j), (i,j-1), (i,j+1)}

- Pour les sommets de la première ligne:

  V (0,0) = {(1,0),(0,1)} et V(0,M-1) = { (1,M-1),(0,M-2)}

  Pour 1 ≤ j ≤ M-2 : V (0,j) = { (1,j), (0,j-1), (0,j+1)}.
- Pour les sommets de la dernière ligne :

  V (N-1,0) = {(N-1,1), (N-2,0)} et V(N-1,M-1) = { (N-2,M-1), (N-1,M-2)}.

  Pour 1≤ j ≤ M-2 : V (N-1,j) = { (N-2,j), (N-1,j-1), (N-1,j+1)}.
- Pour les sommets intérieurs de la première colonne :

  Pour 1 ≤ i ≤ N-2 : V(i,0) = { (i,1), (i-1,0), (i+1,0)}.
- Pour les sommets intérieurs de la dernière colonne :

  Pour 1 ≤ i ≤ N-2 : V(i,M-1) = { (i,M-2), (i-1,M-1), (i+1,M-1)}.

La communication entre deux sommets opposés de la maille nécessite de traverser N+M-2 liens.

3.1.1.3-Propriétés de la maille

1. **Grand diamètre :** Le diamètre d'un réseau est la distance maximale entre n'importe quel couple de processeurs.

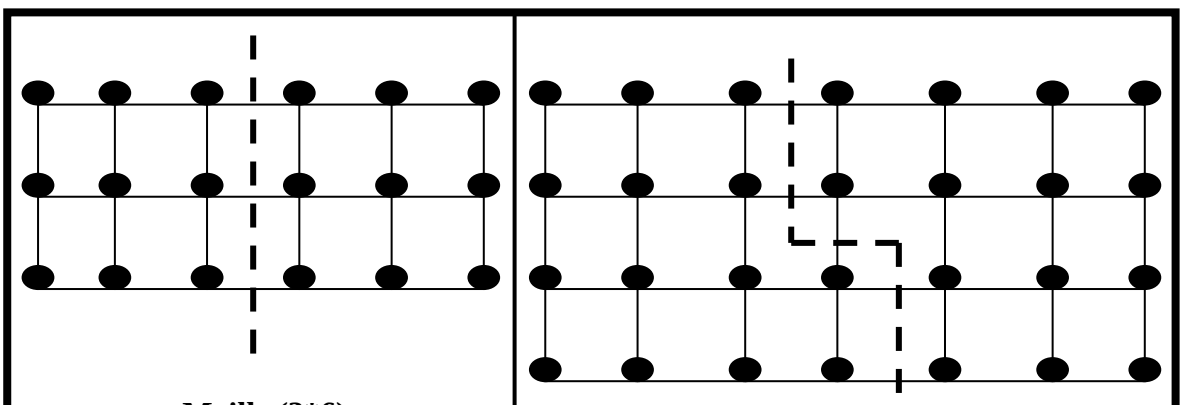
   La distance entre deux processeurs est le plus petit nombre de liaisons qui doivent être traversées pour passer d'un processeur à un autre. Le diamètre d'un vecteur N*M (Maille) est égal à (N+M-2). **[DT00]**

2. **Petite largeur de bissection :** La largeur de bissection d'un réseau est le nombre minimal de liaisons qui doivent être enlevées pour déconnecter le réseau en deux parties identiques. La largeur de bissection d'une maille (N*M) est :

   - Min (N, M) si Max (N, M) est pair

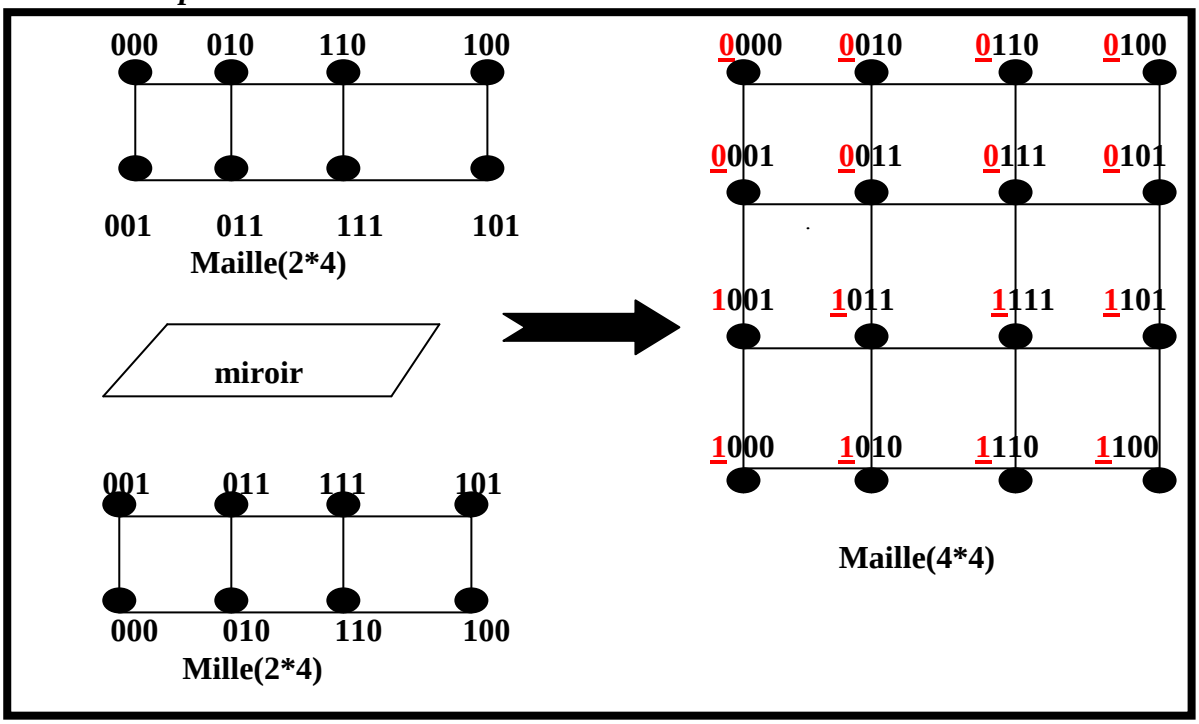
   - Min (N, M)+1 si Max(N, M) est impair **[DT00]**

Exemple :



3. **La récursivité :** La construction d'une maille (N*M) est obtenue par la liaison de deux copies résultantes de l'application de l'effet miroir, l'une est préfixé par '0' et l'autre par '1'.

Exemple :



3.2-Les Arbres :

Un arbre est un graphe connexe sans cycle. La notion d'orientation n'intervient pas dans l'arbre. Il existe plusieurs théorèmes qui définissent les propriétés de l'arbre.

Théorème1 :

Un arbre G=(X , U) tel que|X| = n >=2, Les propriétés suivantes sont équivalentes :

1 G est connexe et sans cycles

2 G est sans cycle et possède (n-1) arcs

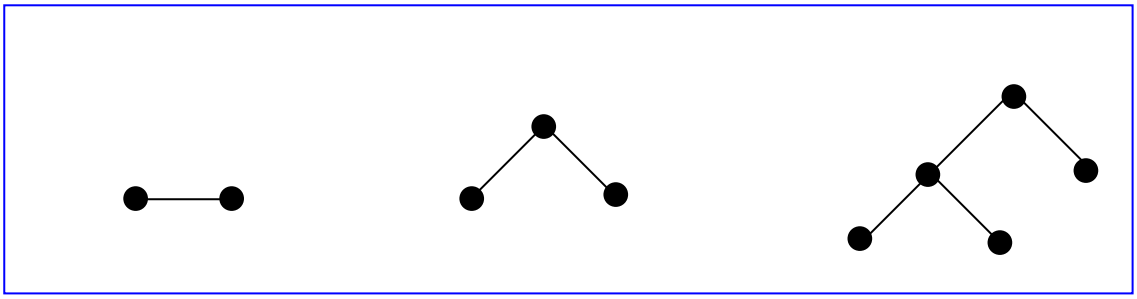
3 G est sans cycle est maximale dans cette propriété (l'ajout d'un arc conduit à la création d'un cycle ,et un seul).

4 G est connexe et possède (n-1) arcs.

5 G est connexe et minimale pour cette propriété (si on supprime un Arcs quelconque il n'est plus connexe) .

6 Il existe dans G une chaîne et une seule joignant tous les couples de sommets **[CB73]**.

Exemple **:**



« 2 sommets pondant » « 3 sommets pondant »

Théorème2 :

Si G est un graphe connexe avec au moins un arc, les conditions suivantes sont équivalentes :

1 G est fortement connexe.

2 par tout arc passe un circuit.

3 G ne contient pas un co-circuit. **[CB73]**

3.2.1- Les arbres particuliers

3.2.1.1-Arbre binaire

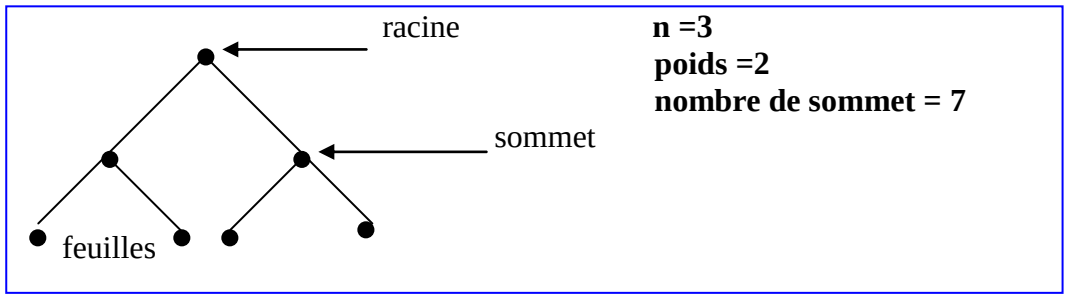
Les arbres permettent de représenter naturellement des machines hiérarchiques. Dans un arbre binaire, un nœud peut avoir aux plus deux fils. On ne compte pas les fils à partir de la gauche mais on réserve des emplacements, un pour un fils gauche et un autre pour un fils droit, l'un ou l'autre voire les deux emplacements peuvent être vide **[LEI92]**.

3.2.1.2-Arbre binaire complet

Définition :

Un arbre binaire complet de poids (n-1) dénoté par (CBT : Complet Binary Tree) est un graphe connexe et sans cycle contenant (2^n -1) sommets. Chaque sommet ayant une profondeur inférieure à n (les sommets non terminaux) et possède exactement deux fils (fils gauche, fils droit). Un sommet de profondeur (n-1) est une feuille.

Exemple



Arbre binaire de profondeur 2

Degré dans un CBTn :

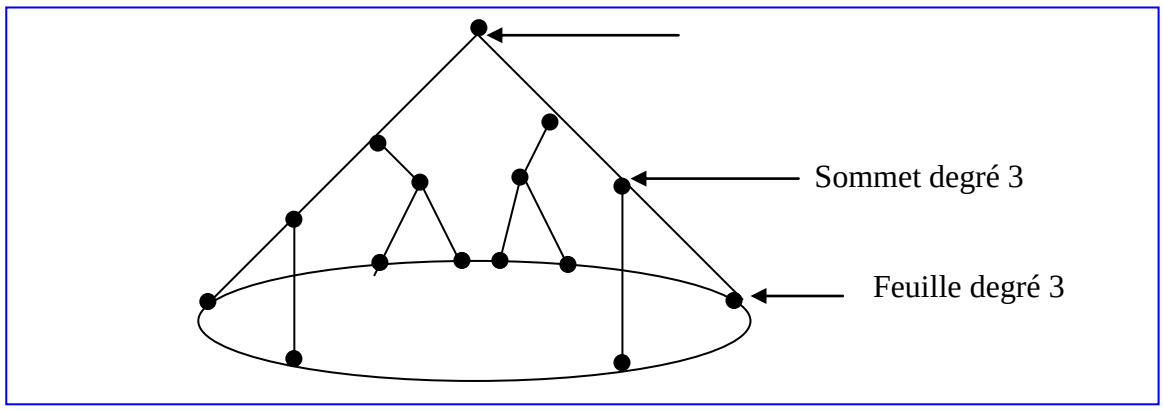
Le degré moyen d'un processeur est égale à 3, la racine a seulement deux liens c'est à dire de degré égal à 2, Les nœuds terminaux (feuilles) ont un degré égal à 1 seulement.

Remarque :

On peux ajouter à la racine un troisième servant par exemple de liaison avec l'extérieur (Pour les entrées sorties).

On peux se servir du faible degré des feuilles pour améliorer la communication au niveau le plus profond de l'arbre (on obtient un graphe régulier de degré 3) **[MC93]**.

Exemple



Arbre binaire-anneau

Diamètre :

Le diamètre de l'arbre binaire complet est égal à $2|\log_2(N)|$ ou N est le nombre de sommets. Tous les nœuds sont à une distance inférieure à $\log_2(N)$ de la racine

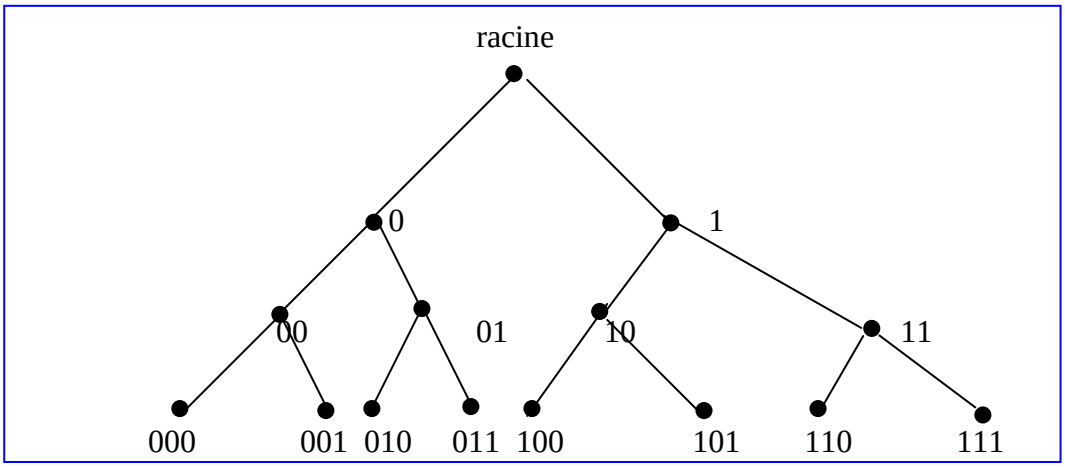
Numérotation des processeurs d'un arbre binaire complet:

Il existe plusieurs numérotations commodes des processeurs d'un arbre binaire complet, les deux représentations les plus populaires sont :

- La représentation binaire.
- La représentation entière **[MC93]**.

a-Représentation Binaire :

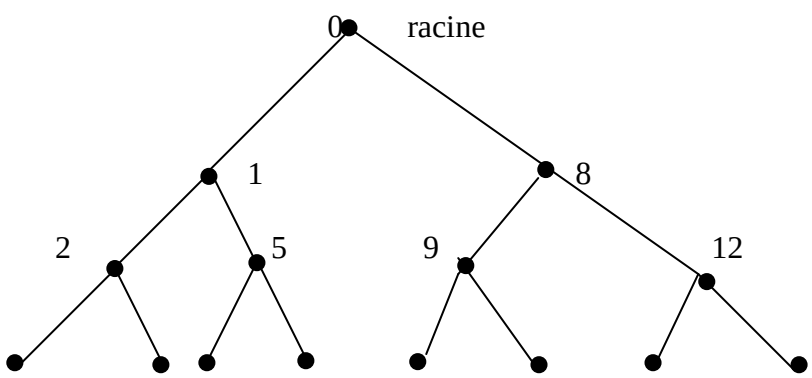
Le principe de numérotation des deux fils d'un sommet revient à garder le même préfixe et d'ajouter 0 à droite pour le fils gauche et 1 pour le fils droit. L'intérêt est de conserver une numérotation croissante de gauche à droite à l'intérieure d'un même niveau **[MC93]** .



Représentation Binaire

b-Représentation Entière :

Dans cette numérotation, tous les sommets à gauche d'un sommet donné sont de numéros inférieurs à tous ceux de droite **[MC93]**.



3 4 6 7 10 11 13 14

Représentation Entière

Propriétés de l'arbre binaire complet :

- L'ABC est un graphe connexe et sans cycle, car chaque nœud non terminal possède deux fils exactement.
- L'ABC à une profondeur égale à $\log_2(p)$ tel que p : est le nombre de sommets et un diamètre égale à $2*\log_2(p)$, ou le diamètre d'un arbre représente le parcours de sa structure de la racine aux feuilles.
- L'ABC est un graphe symétrique.
- L'ABC est un graphe planaire. **[LEI92]**

les avantages et les inconvénients de l'arbre binaire complet :

L'avantage de l'arbre binaire complet est utilisé pour résoudre les applications divide-conquer. Les inconvénients sont :

- L'arbre binaire complet a une grand diamètre égal $2|\log_2(N)|$.
- L'arbre binaire a une largeur de bissection égale 1 c'est à dire que si on supprime une arête tout le réseau tombe en panne.
- L'arbre est un réseau ou il n'existe qu'un seul chemin entre n'importe quels deux nœuds (connectivité égale 1), si un processeur A veut envoyer un message à un autre processeur B, quel est le chemin que ce message doit prendre? (il n'y a qu'un seul chemin)?

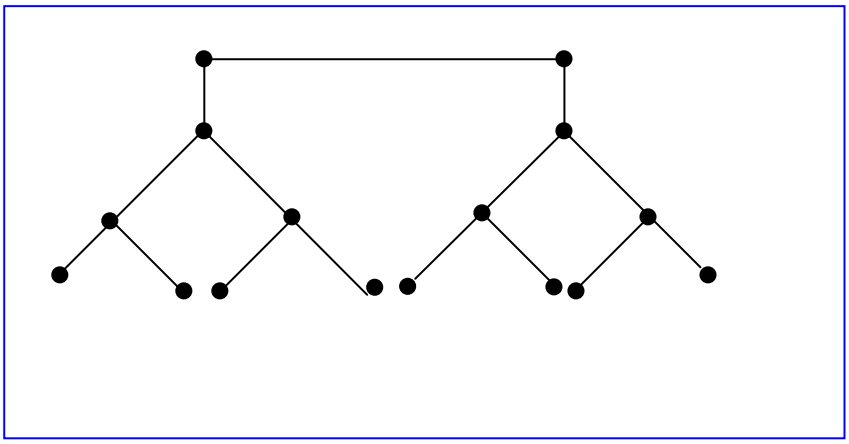
  Tous les chemins doivent passer par la racine. Imaginez l'embouteillage (goulot d'étranglement) si plusieurs processeurs décident en même temps de communiquer entre eux. C'est un problème majeur dans ce genre de réseau.
- L'arbre binaire n'a pas de symétrie de nœuds ni symétrie des arcs parce que, la racine ne peut être similaire à aucun autre nœud pour la simple raison que la racine n'a pas de prédécesseur (ou de parent), tous les autres nœuds ont un parent. La même chose, les feuilles n'ont pas de successeurs ou d'enfants. **[SB02],[SB94]**

3.2.1.3-Arbre binaire complet avec double racine :

Il existe un graphe très semblable à l'arbre binaire complet, qui est l'arbre binaire complet avec double racine de N nœuds dénoté par BCDRT

L'arbre BCDRT est un arbre binaire complet avec la racine remplacée par un chemin de longueur deux (qui relie 2 nœuds) **[LEI92]**.

Exemple



" Arbre binaire complet avec double racine de 16 nœuds"

Conclusion:

Après les détails donnés sur l'architecture des réseaux d'interconnexions simple, les vecteurs et les arbres que nous avons examinés sont relativement élémentaires pour construire une architecture parallèle, et ils sont aussi très efficaces pour la résolution de quelques problèmes, mais ils présentent certains inconvénients, dont les deux les plus populaires sont :

1. Un grand diamètre.
2. Une petite largeur de bissection.

En conséquence, la vitesse qu'ils employent pour résoudre beaucoup de problèmes est fortement limitée.

3.3-Réseaux d'interconnexion hybride

3.3.1-Introduction*:*

Dans cette partie, nous considérons une architecture de ' réseau d'interconnexion hybride' basée sur des vecteurs et des arbres appelée 'les mailles d'arbres'.

3.3.2-La maille d'arbre bidimensionnel

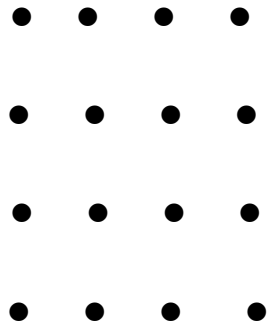
Définition :

Une maille d'arbres bidimensionnelle (n*n) est construite à partir d'une grille de n*n processeurs, et par un ensemble de processeurs et d'arrêtes ajoutés afin de former un arbre binaire complet dans chaque ligne et dans chaque colonne. Les feuilles des arbres sont

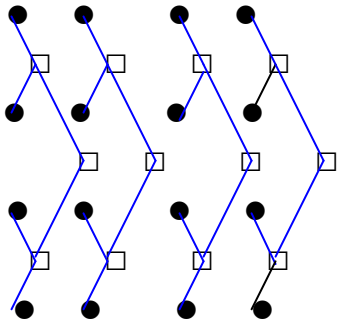
les N^2 nœuds originaux de la grille, et les nœuds supplémentaires sont les nœuds internes des arbres.

Les processeurs des feuilles et des racines ont un degré égal à 2, tout les autres processeurs des nœuds intermédiaires ont un degré égal à 3. La maille d'arbres bidimensionnel contient $3N^2$-2N processeurs **[B3]**.

Exemple :

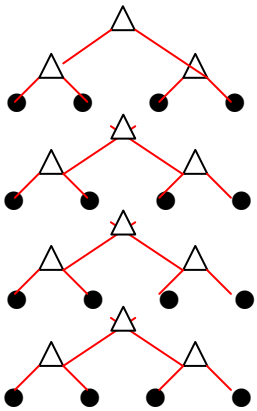


une grille de N*N nœuds

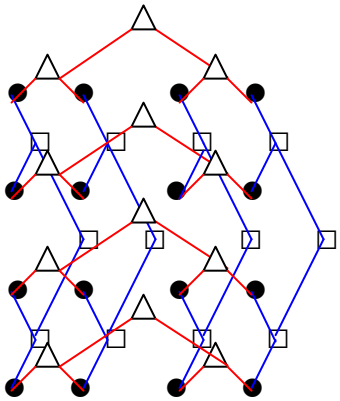


nœuds et liens ajoutés pour former les arbres de colonnes

Introductin d'arbre vertical



nœuds et liens ajoutés pour former les arbres de lignes



la maille d'arbre bidimensionnelle N*N

3.3.2.1-Propriétés de la maille d'arbres bidimensionnel

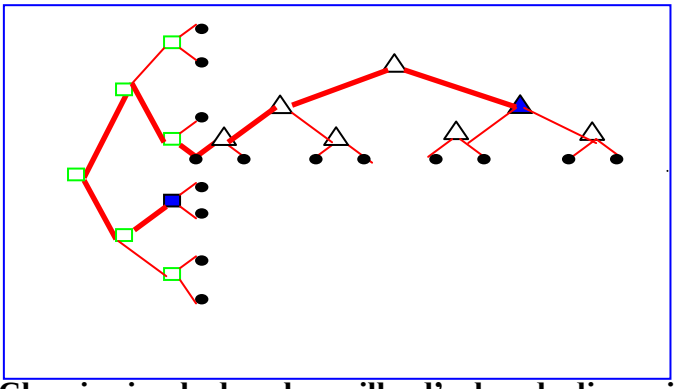
Les mailles des arbres disposent de plusieurs propriétés qui les rendent très efficace et puissant. Nous nous contenterons d'en citer trois propriétés seulement :

Le petit diamètre :

On peut vérifier que le diamètre de la maille d'arbre N*N est $4\log_2 N$ par exemple, pour construire un chemin de longueur $4\log_2 N$ au plus , de n'importe quel nœud 'u' dans l'arbre de i ème ligne à n'importe quel autre nœud 'v' dans l'arbre de j ème colonne, nous construisons d'abord le chemin de longueur $2\log_2 N$ de 'u' à 'z' dans l'arbre de i ème ligne, ou 'z' est la feuille unique partagée par l'arbre de la i ème ligne et de la j ème colonne, ce qui conduit à un chemin de longueur $2*\log_2 N$ de 'z' à 'v' dans l'arbre de la j ème colonne.

Un argument de la symétrie indique que la distance entre deux nœuds quelconques des arbres des colonnes est égal à $4*\log_2 N$ au plus. **[LEI92]**

Exemple :



Chemin simple dans la maille d'arbre de dimension N*N

Une grande largeur de bissection :

En plus d'avoir un petit diamètre, les mailles bidimensionnelles d'arbres ont également une largeur relativement grande de bissection. En fait, les 'N' nœuds et les arêtes doivent être enlevées de la maille du N*N d'arbre pour le déconnecter dans deux parties de la même dimension. Une largeur de bissection de taille N, comme elle peut être efficace pour la résolution de la plupart des problèmes de taille N, elle est aussi bien efficace pour beaucoup de problèmes de taille N^2. La maille d'arbre bidimensionnelle à une structure très naturelle et régulière. **[LEI92]**

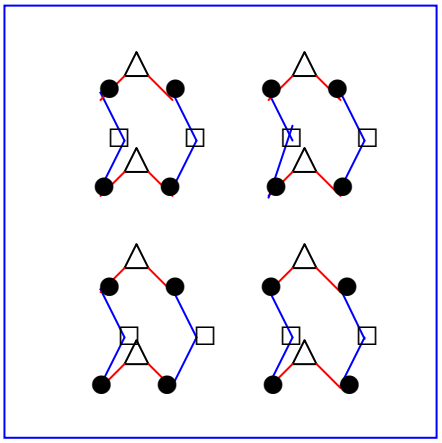
Décomposition récursive :

Lorsqu'on enlève de la maille d'arbre bidimensionnelle (N*N), les racines de la ligne et la colonne 2N, ainsi que les arêtes qui lui sont incident, on obtient 4 copies disjointes de maille d'arbres (N/2*N/2). Cette décomposition récursive, est efficace pour plusieurs raisons :

d'abord, la décomposition peut être facilement exploitée par des algorithmes récursifs dans le calcul parallèle ; ensuite la décomposition peut être exploitée en fabriquant un réseau puisque 2^{2i} (N*N) mailles d'arbres peuvent être facilement relier ensemble pour former une maille d'arbres de dimension (($N2^{i}$)*(N^{i} 2)), enfin La décomposition récursive est aussi utile pour concevoir des dispositions effectives pour le réseau. **[LEI92]**

Exemple :

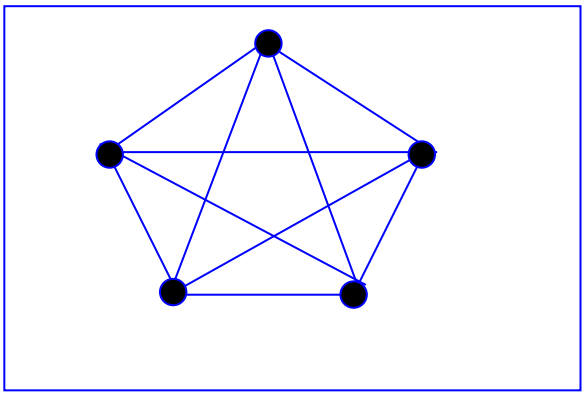
La maille d'arbre 4*4 avec enlèvement des racines des arbres de lignes et de colonnes



Dérivation de $K_{N,N}$:

Naturellement le meilleur réseau du point de vue informatique est celui ou chaque processeur est lié à tous les autres processeurs du réseau directement. Dans ce cas, le graphe du réseau est complément connecté. Ce graphe est appelé 'graphe complet', dénoté K_n ou n est le nombre de processeurs (nœuds). **[LEI92]**

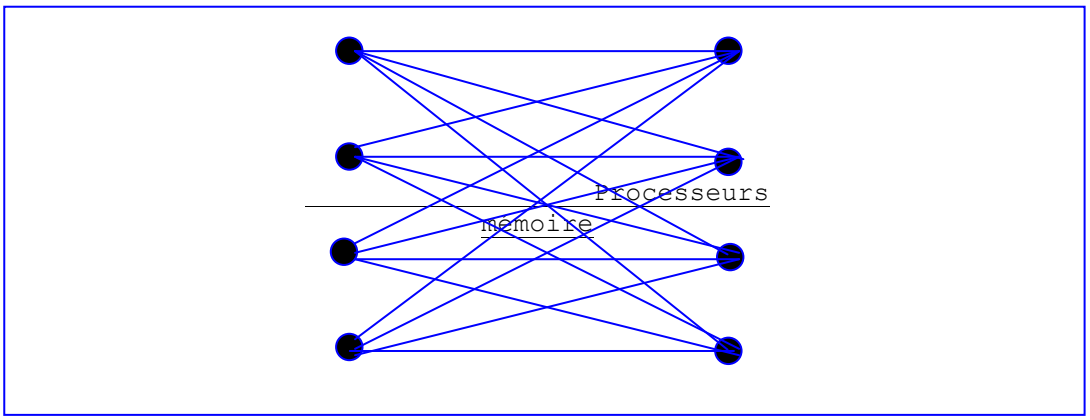
Exemple :



Le graphe complet avec 5 nœuds K5

Si on sépare le fonctionnement de chaque processeur au niveau de la mémoire et du calcul, le réseau idéal est '**le graphe biparti complet**' dénoté par $K_{n,n}$. Dans la partie calcul, chaque processeur peut accéder directement à chaque processeur de la partie mémoire. **[LEI92]**

Exemple :

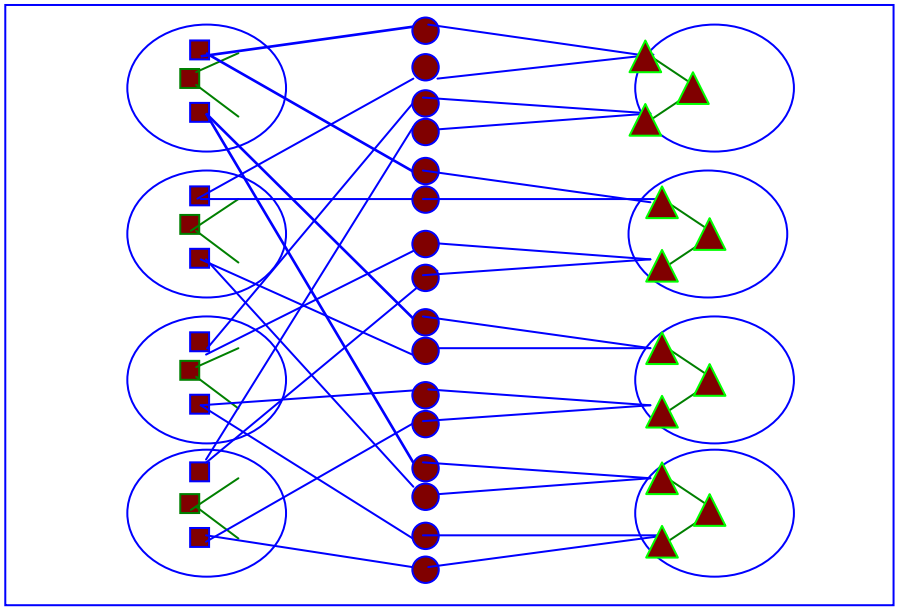


Le problème de ces réseaux idéaux est que le degré de chaque nœud devient insupportablement grand, d'où l'augmentation de nombre de processeur. Ils ne peuvent donc pas être fabriqués pour une grande dimension N.

La solution consiste à le simuler par la considération de la forme réseau 'bounded-degree' par le remplacement de chaque nœud de degré N par un arbre binaire complet de N feuilles, plus précisément remplacer chacune de N^2 arêtes du graphes $K_{n\ ,n}$ par un processeur feuille, chaque nœud de degré N par un processeur racine et la connexion de la racine aux N feuilles incidentes sera effectuée par un arbre binaire .

Le graphe résultant peut simuler n'importe quelle étape du $K_{n\ ,n}$ en ''$2*\log_2 N$'' étapes . **[LEI92]**

Exemple :



3.3.3-Les mailles des arbres de dimension N

Définition :

La maille d'arbre de dimension r (N*N* ….*N) est formée en ajoutant des arbres à une grille de N-faces et de r-dimension. Les nœuds de la grille deviennent les feuilles des arbres et peuvent être vue comme r- tuples(i_1,i_2 ,……, i_r) avec 0<= i_1 ,……., i_r<=N-1.

Deux feuilles sont dans un même arbre si et seulement si ils se différent en un seul chiffre , l'endroit de différents chiffres détermine la dimension de l'arbre qui contient les deux feuilles.

Par exemple : l'arbre (i_1,i_2,……,i_{r-1},*) dans le r^{eme} dimension contient les feuilles { (i_1,i_2,……,i_r) / 0<= i_r<=N-1 }.

La maille d'arbres de r- dimension contient |v|= (r+1) N^r- rN^{r-1} nœuds et |E| = $2rN^r$-rN^{r-1} arêtes distribuer entre ses rN^{r-1} arbre binaire complet de n-feuilles .

Le diamètre du réseau est 2*r*logN = (log|v|) , et le largeur de bissection est N^{r-1}= (|v|/rN).

Bien que les mailles d'arbres de r-dimension à plusieurs propriétés soient importantes pour le calcul parallèle, c'est très utile probablement quand r est très grand et N=2.

Dans ce cas, tous les arbres ont exactement deux feuilles qui sont connectées directement par l'élimination des racines. Ce qui permet la construction de l'hypercube de dimension –r.

Conclusion :

Une lecture effective au contenu de cette partie, nous permet de conclure que l'architecture du réseau d'interconnexions hybride possède des avantages tel que : le petit diamètre et la grande largeur de bissection, qu'ils lui donnent une efficacité et une rapidité pour la résolution des problèmes.

3.4-Les réseaux de connexions hypercube

Introduction

Le modèle de l' hypercube a été proposé dès 1962 pas Squire et Palais á l'université du Michigan , puis repris plus tard par (PASE 77) et (LOCANTHI 80) **[MC93]** . Il est l'un des réseaux les plus flexibles et effectifs découverts pour le calcul parallèle.

3.4.1-Définition

L'hypercube de dimension (d) est un réseau qui contient $N=2^d$ nœuds et $d*2^{(d-1)}$ arêtes , où chaque nœud a exactement (d) voisins**.** Chaque nœud est numéroté par un nombre binaire de d-bits. Les connexions sont établies seulement entre les nœuds dont le numéro ne différent que d'un seul bit. Dans l'espace a trois dimension (d =3 ,N =8), il existe trois familles de connexions selon les numéros des nœuds (processeurs) qui ne diffèrent que d'un bit du poids le plus faible (connexions appelés C0), du bit de poids le plus fort (connexion C2) et du bit de poids intermédiaire (connexions C1) **[JPS02]**

Par exemple: dans la famille de connexions C0.

Le nœud numéro (0) (000 en binaire) sera relié au nœud numéro (1) (001),

Le nœud numéro (2) (010) sera relié au nœud numéro (3) (011) .

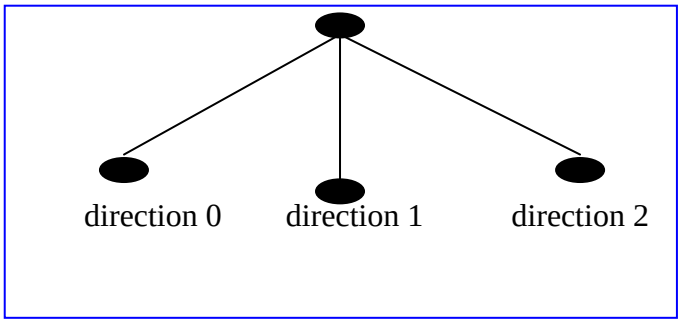
On peut résumer cela en disant que 2 nœuds ne sont reliés que si leur distance de Hamming est égale à 1 .

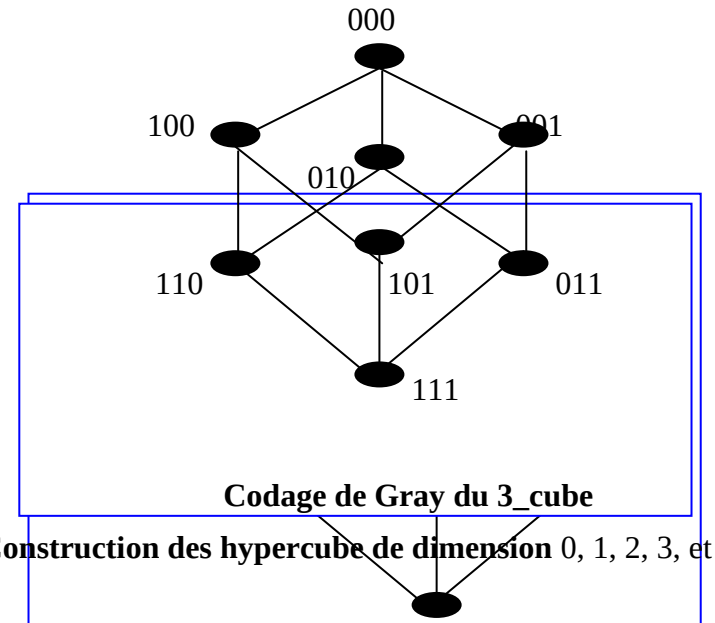
Par conséquent , chaque nœud est adjacent à (d)autres nœuds tel que d =$\log_2$N (le degré de chaque nœud) .

Code de Gray :

Il existe. À partir des codes de Gray une numérotation commode, où chaque nœud du réseau est un entier codé sur d bits en binaire , dont on déduit tous les voisins en complémentant successivement chaque bit de cette numérotation . **[MC93]**

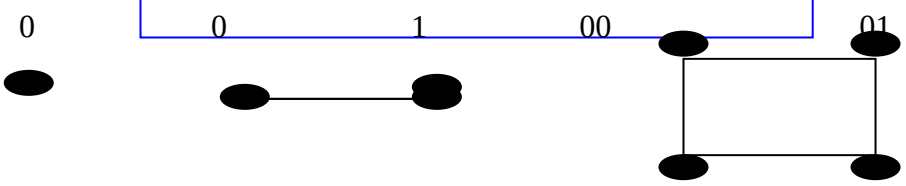
Par exemple : Le nœud 1 d'un 3-cube (correspondant au codage binaire 001) est relié aux nœuds 0,3,5 (resp. 000, 011 et 101) chaque position de bit correspond à une direction.



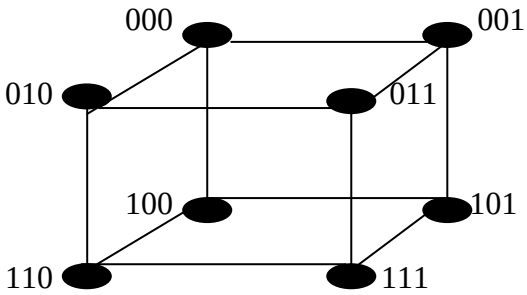


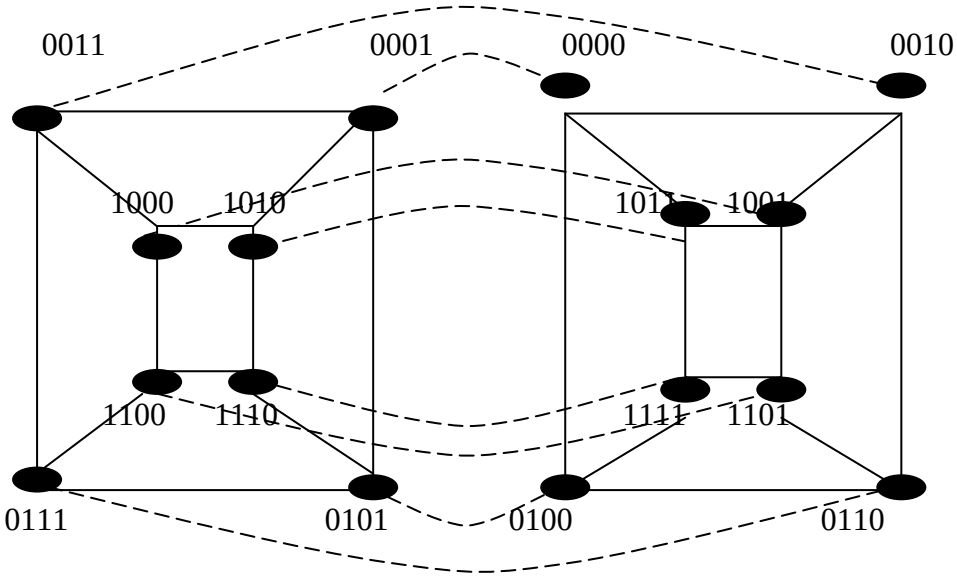
Codage de Gray du 3_cube

Exemple: **Construction des hypercube de dimension** 0, 1, 2, 3, et 4.



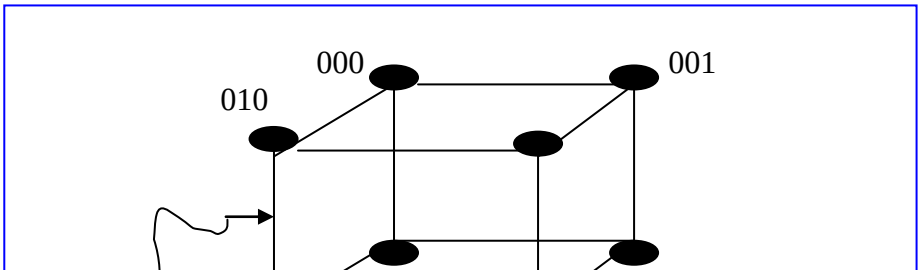
11 10

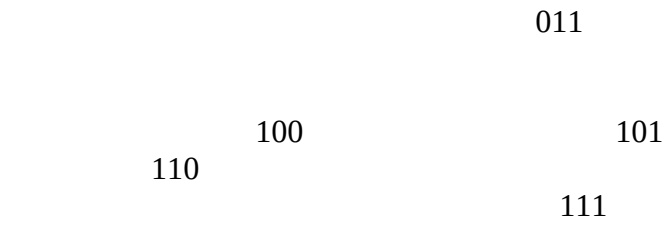




Les arêtes de l'hypercube peuvent être naturellement divisées selon les dimensions qu'elles traversent, particulièrement. une arête est appelée l'arête de dimension K ,si elle lie deux nœuds dont leurs adresses binaires se diffèrent dans la position du K^{ieme} bit . **[LEI92]**

Exemple:





arête de dimension 3

3.4.2-Propriétés de L'hypercube

L'hypercube posséde des propriétés trés importantes qui lui permettent d'être une bonne topolgie utilisée dans les machines modernes et qui lui donnent une puissance ,une souplesse et une efficacité .

- L'hypercube est un graphe fortement connexe du fait qu'il existe un chemin entre deux nœuds du graphe . **[LEI92]**

- L'hypercube possède un petit degré par rapport aux autres architectures. Tel que degré égal à la dimension de l'hypercube. **[LEI92]**

- L'hypercube possède une topologie régulière, c'est à dire que tous les nœuds ont le même degré **[LEI92]** .

- On peut noter que l'hypercube possède beaucoup de symétries. Il est nœud et arête Symétrique **[BP94]**. On peut dresser une carte de n'importe quel nœud sur un autre nœud, et n'importe quelle arête sur une autre arête. Plus précisément, pour n'importe quelle paire d'arêtes (u, v)et (u', v') dans un hypercube de N-nœuds (H) Il existe un automorphisme de H tel que : σ(u) =u' et σ(v) = v' **[LEI92]**

- L'hypercube est un graphe de Cayley. Le Graphe de Cayley est un graphe hamiltonien,, ceci rend l'hypercube un graphe hamiltonien . **[MC93]**

- En plus d'une structure simple et récursive, l'hypercube possède beaucoup d'autres propriétés importantes. Particulièrement , il a un petit diamètre (Log_2N) quand N est petit et une grande largeur de Bissection (N/2). **[LEI92]**

L'attachement des nœuds de diamètre est facilement prouvé en observant que n'importe quels deux nœuds U=U1 U2. .. U log N , et V=V1V2...Vlog N sont connectés par le chemin :

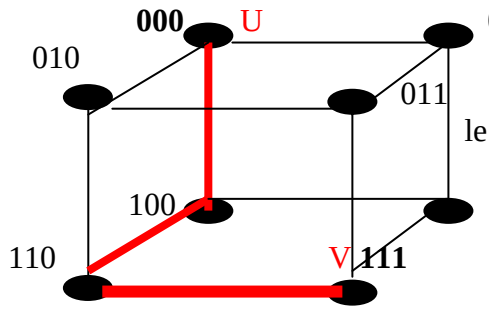
$U_1U_2...U_{logN} \rightarrow$V1U2…Ul og N $\rightarrow V_1V_2U_3\ldots U_{logN}$

$\rightarrow \ldots \rightarrow V_1V_2 \ldots V_{logN-1}\ U_{logN} \rightarrow V_1V_2\ldots\ V_{logN}$

Exemple :

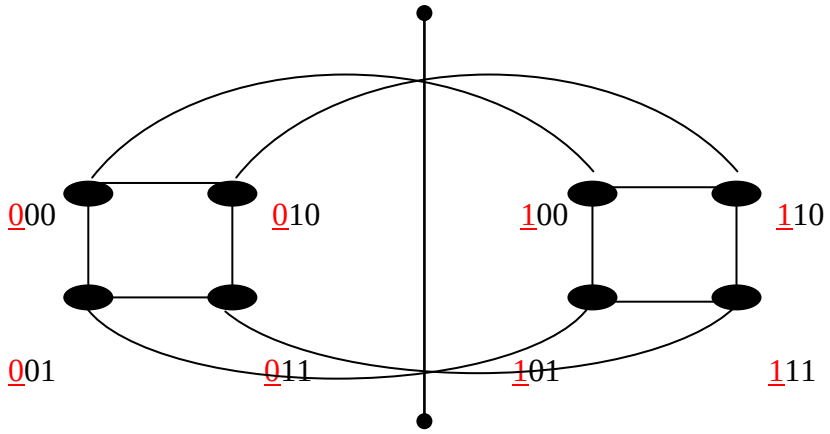
Pour les nœuds $U = U_1U_2U_3 = (000)$ et $V = V_1V_2V_3 = (111)$ dans un hypercube de dimension 3. Ils sont reliés par le chemin :

$U_1U_2U_3(\ 000\ ) \rightarrow V_1U_2U_3\ (100\ ) \rightarrow V_1V_2U_3\ (\ 110\ ) \rightarrow V_1V_2V_3\ (\ 111\ )$



le chemin du nœud U(000) au nœud V(111)

Diamètre = 3

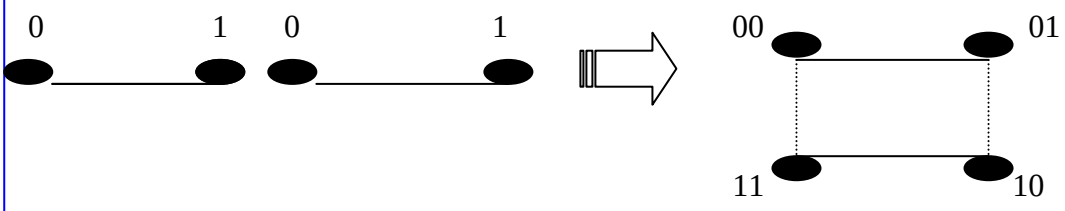


largeur de Bissection = 8/2= 4

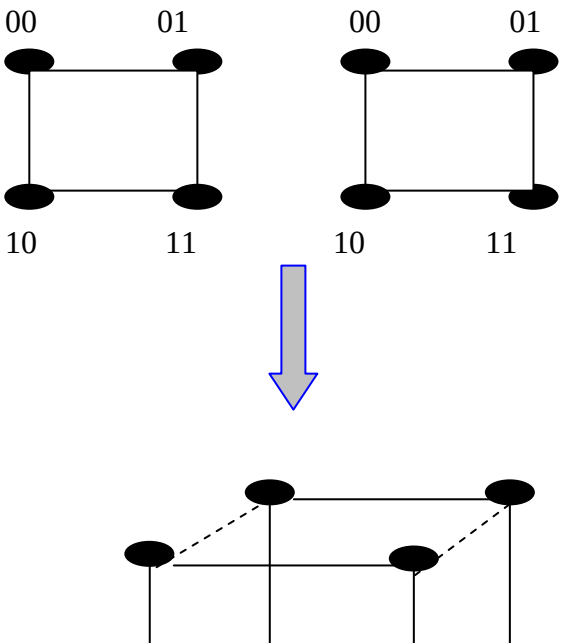
- La construction d'un hypercube est récursive, car l' hypercube de N-nœuds peut être construit à partir de deux hypercubes de N/2 nœuds en connectant simplement le i ème nœuds d' un hypercube de N/2 nœuds à l'autre i ème nœud de l'autre hypercube de N/2 nœuds pour

$0 \leq i \leq N/2$. **[LEI92]**

Exemple:



Construction d'un hypercube de dimension 2 à partir de deux hypercubes de dimension 1



000 010

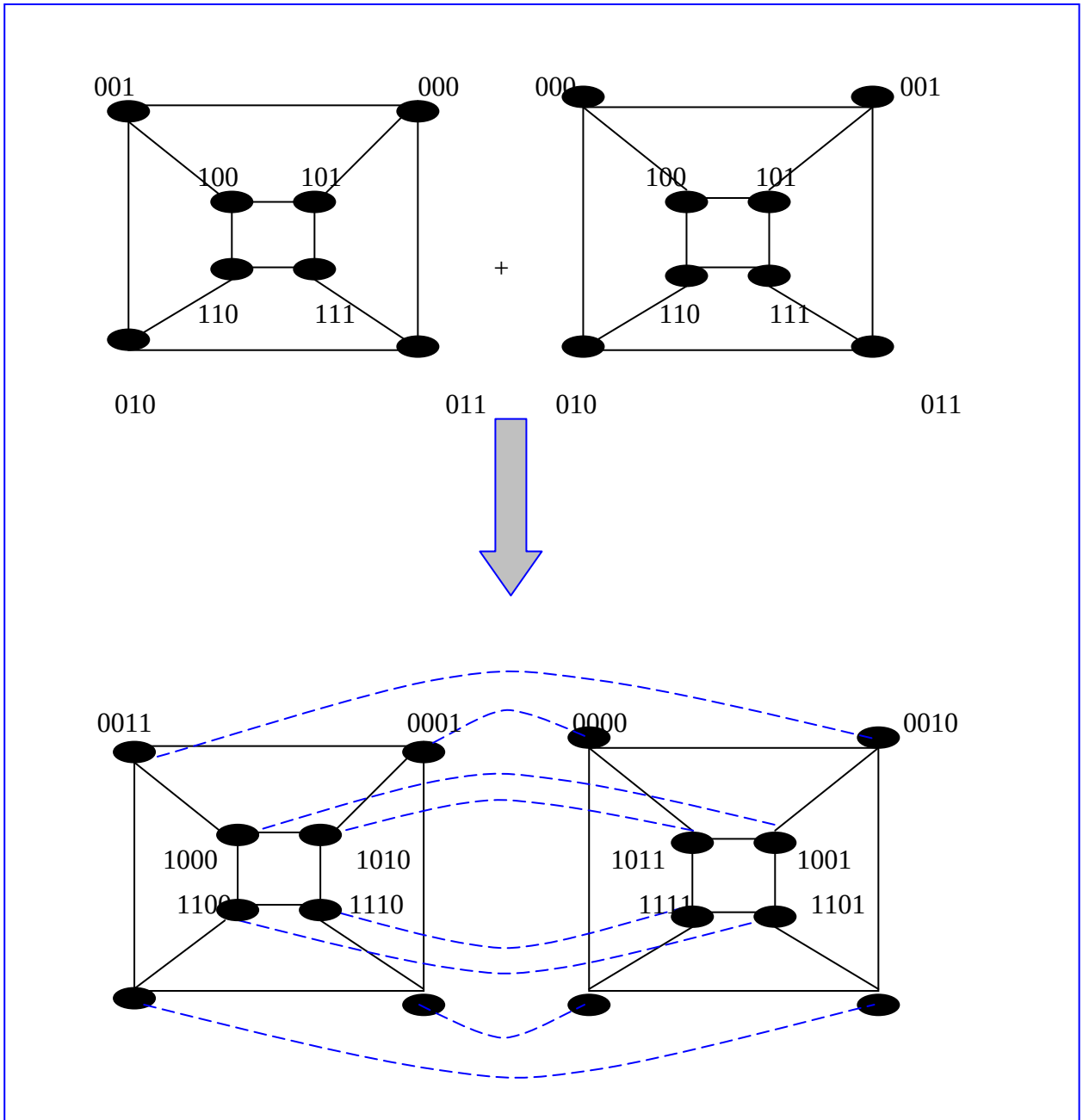
100

110

001 011

101 111

Construction d'un hypercube de dimension 3 à partir de deux hypercubes de dimension 2



0111 0101 0100 0110

Construction d'un hypercube de dimension 4 à partir de deux hypercubes de dimension 3

3.4.3-Les Réseaux contenus dans un Hypercube

les Vecteurs

L' hypercube de N-nœuds contient un vecteur linéaire de N-nœuds comme sous graphe, c'est l'une des proprietés la plus intéressante de l'hypercube de N-nœuds . **[LEI92]**

Lemme :

L'hypercube de N-nœuds contient un vecteur linéaire de N-cellules comme étant un sous graphe pour ($N \geq 4$) **[LEI92]**

3.4.2.2-Les anneaux

On obtient facilement un anneau à partir d'une topologie hypercube. Un anneau dont le nombre de sommets est une puissance de 2, est un graphe partiel de l'hypercube correspondant. **[LEI92]**

3.4.2.3-Les Grilles Toriques

Un des résultats les plus utiles est le passage d'une grille torique à un hypercube. Pour le cas de grilles toriques carrés dont le nombre de processeurs est une puissance de 2, il a été montré que la grille torique est un graphe partiel de l'hypercube . **[LEI92]**

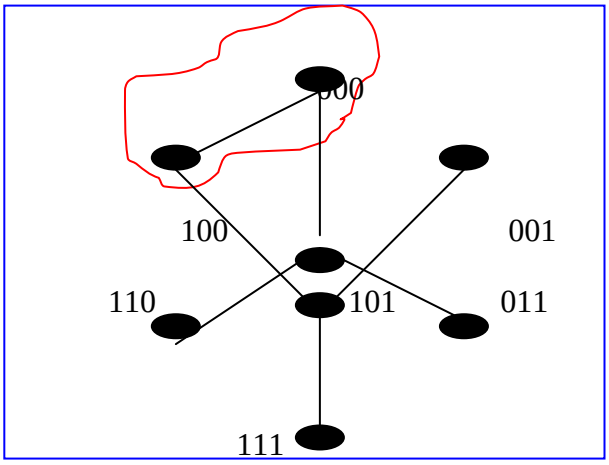
3.4.2.4-Les Mailles d'arbres

Les plus importants réseaux qui sont contenus dans l'hypercube sont les mailles d'arbres, vue qu'ils sont considérés les plus puissants pour le traitement parallèle **[LEI92]**.

3.4.2.5-Arbre binaire complet avec double racine

Un arbre binaire complet avec double racine est un graphe partiel de l'hypercube de N-nœuds**[LEI92]** .

Exemple : Arbre binaire complet à double racine dans un hypercube de degré 3



3.4.4-Les avantages et les inconvénients de l'hypercube

Les avantages :

Le choix d'une bonne topologie est de trouver des réseaux à degré borné avec un petit diamètre. Pour cela, l'hypercube est un choix excellent pour une machine parallèle à base de réseau d'interconnexion. Il possède plusieurs avantages à savoir :

-Petit nombre de connexions de chaque processeur.

-Nombre élèvé de sommets pour autoriser un parallélisme matériel massif .

- Un petit diamètre.

- Une grande largeur de bissection tel que (largeur de bissection =N/2).

- Une tolérance aux pannes : c'est à dire si un processeur tombe en panne dans un réseau hypercube, on peut le remplacer par un autre processeur puisque tous les processeurs (nœuds) sont similaires (l'hypercube est un arête et nœud symétrique).

- Il adapte les architectures vecteurs, arbres, anneaux, grilles, mailles d'arbre, de sorte que des voisins sur ces topologies se retrouvent également voisins sur l'hypercube, ce qui en fait une topologie utilisable dans une vaste gamme de calcul.

Sur le plan commercial, les fabricants d'hypercube ont dominé le marché american des machines MIMD à mémoire distribuée pendant les années 80 du fait que les concepteurs d'hypercube FOX et SEITZ ,avaient montré qu' un grand nombre de problèmes scientifiques et d'ingénierie trouvaient leurs solutions sur cette topologie . **[LEI92]**

Les inconvénients

D'un point de vue informatique l'hypercube est le plus puissant, mais il existe quelques inconvénients à son utilisation comme architecture pour le calcul parallèle. Ces inconvénients sont :

-Le degré du nœud de l'hypercube croit avec sa taille, ceci signifie que des processeurs conçus pour un hypercube de N nœuds ne peuvent pas plus tard être employés dans un hypercube de 2N nœuds .

-Le degré dépend du diamètre et si on veut réaliser des réseaux avec un grand nombre de processeurs, il y aura des liens de longueur différente, ce qui rend difficile la construction matérielle de cette machine .

3.4.5-Les machines à base d'hypercube

La réalisation d'une machine à base d'une topologie hypercube n' a été mise en œuvre dans le première fois qu'en 1983 au Caltceh [SEITZ85] [Fox85] sous le nom de COSMIC CUBE, cette machine comprenait 64 nœuds, chaque nœud étant constitue d'un microprocesseur INTEL 8086 et d'un coprocesseur 8087. Les réalisations suivantes mises au point par les mêmes auteurs, eurent pour noms MARKI, MARKII, MARKIII, puis MARKIII fp réalisée en 1988 est un hypercube de 128 nœuds. Chaque nœud est constitue de deux microprocesseurs 68020 de MOTOROLA, d'une unité virgule flottante 68882 et d'une carte accélératrice WEITEK XL. Le système total dispose d'une mémoire de 0.5 G octets et autorise une performance de crête de plus d'un GFlops. C'est également au Caltceh qu'a été réalisé le NCUBE constitue d'un assemblage de cartes contenant 64 unités centrales, chacune possède 11 canaux de communication et 500 K octet de mémoires. Il existe une version qui dispose de 512 nœuds et une autres de 1024. D'autres machines utilisant les avantages de l'hypercube comme connexion machine5, cette machine est composée de

1) 32 à 65536 nœuds qui chacun comporte :

- un processeur SPARC(32 Mhz, 5 MFlops, 22 Mips).
- 4 unités vectorielles pour les opérations flottantes (chacune possède 32 Mflops) .
- 4 bancs mémoires de 8 M octets de DRAM.

3.4.6-Les variations de l'hypercube

Bien que l'hypercube soit assez puissant d'un point de vue informatique, il présente plusieurs inconvénients, parmi eux le degré des nœuds et le diamètre qui évoluent avec sa taille. Pour remédier ces déficiences, les informaticiens ont fait des variations sur l'hypercube pour atténuer les difficultés associes au degré du nœud et du diamètre de l'hypercube.

Plusieurs variations sur l'hypercube ont été faites ; celles jugées les plus importantes sont :

1) Butterfly .

2) Le Cube-Connected Cycle.

3) Le réseau de Benes .

4) L'hypercube croisé .

3.4.6.1-Le butterfly

Le butterfly de dimension d contient : $(d+1)2^d$ nœuds et $d2^{d+1}$ arêtes .

Le nœud qui correspond à la paire (w, i) où i est le niveau ou la dimension du nœud telle que ($0 \leq i \leq d$) , et w est un nombre binaire de d-bits qui dénote la ligne du nœud .

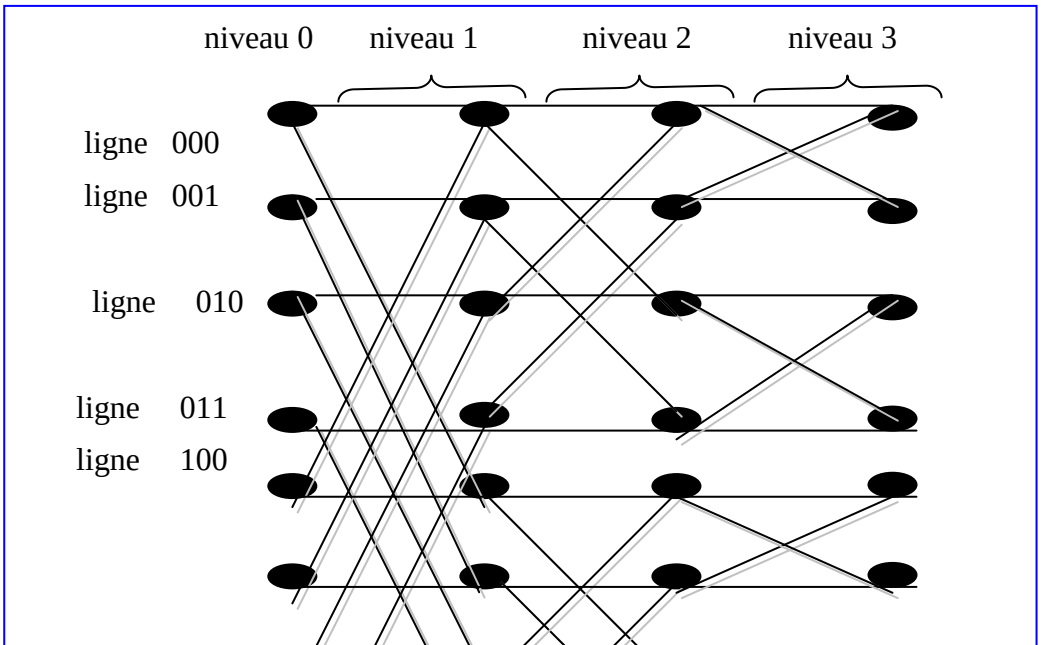
Deux nœuds (w, i),(ẃ, í) sont liés par une arête si et seulement si :

í =i+1 et 1 ou 2 sont réalisées :

1) w et ẃ sont identiques,

2) w et ẃ se différent dans le i^{eme} bit si w et ẃ sont identiques , l'arête serait «une arête droite » sinon , l'arête est une « arête croisée » **[LEI92]**

Exemple:



ligne 101

ligne 110

ligne 111

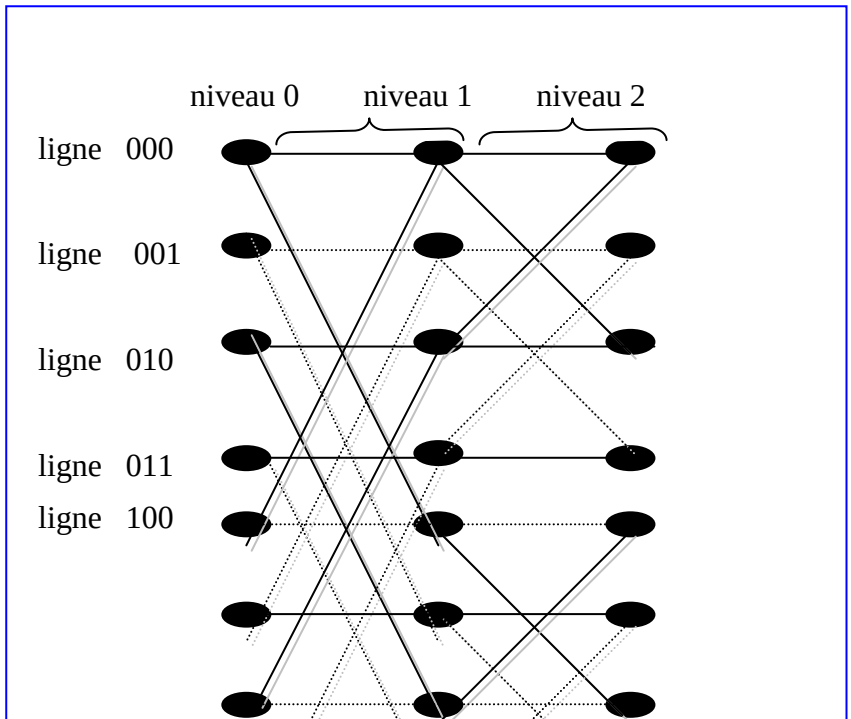
Les arêtes relient des nœuds aux niveaux i et i+1 seraient des arêtes du niveau i+1.

Le butterfly et l'hypercube sont tout à fait semblables en structure en particulier , le i^{eme} nœud de l'hypercube de dimension d correspond naturellement à la $i^{ème}$ ligne du butterfly de dimension d et la $i^{ème}$ arête de dimension (u, v) de l'hypercube correspond à l'arête croisée ((u ,i-1) , (v ,i)) et ((v ,i-1) , (u ,i)) dans le niveau i du butterfly . En effet, nous pouvons obtenir un hypercube à partir d'un butterfly en fusionnant tous les nœuds du butterfly qui sont dans la même ligne et puis on enlève la copie supplémentaire de chaque arête. Par conséquent n'importe quelle étape de calcul d'hypercube de N-nœuds peut être simulée en $\log_2 N$ étapes sur un butterfly de $N(\log_2 N+1)$ nœuds en ayant la $i^{ème}$ ligne du butterfly qui simule l'opération du $i^{ème}$ nœud d'hypercube pour chaque i . **[LEI92]**

Les propriétés du réseau butterfly

1) Il a une structure simple et récursive. Un butterfly de dimension d contient deux butterfly de dimension (d-1) comme sous graphe :

Exemple:



ligne 101

ligne 110

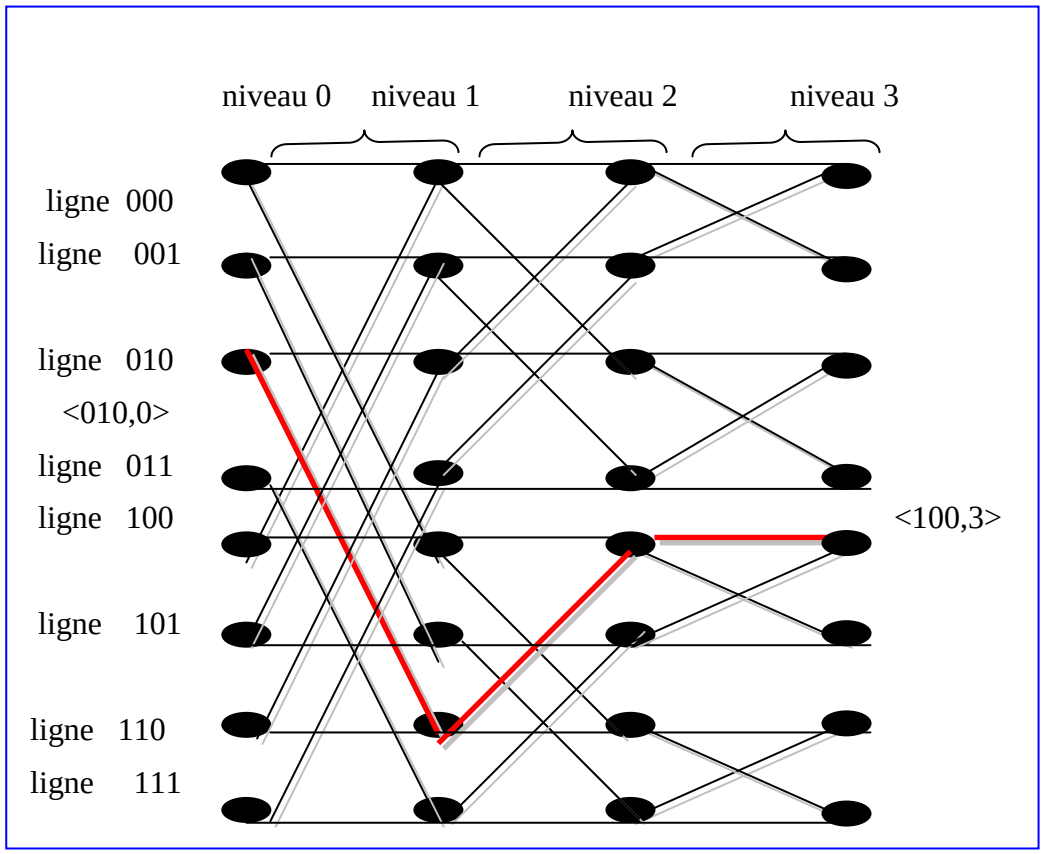
ligne 111

Deux copies disjointes d'un butterfly de dimension 2 à partir d'un butterfly de dimension 3.

L'une est construite par des arêtes continues et l'autre par des arêtes hachurées.

2)Une autre propriété importante du butterfly de dimension d est que le nœud du niveau 0 dans n'importe quelle ligne w est liée au nœud du niveau d dans n'importe quelle ligne ẃ par un chemin unique de longueur d , le chemin traverse chaque niveau exactement une seule fois . **[LEI92]**

Exemple:



3) Comme l'hypercube, le butterfly a également une grande largeur de bissection égale: $\theta = (N/\log_2 N)$ **[LEI92]**

3.4.6.2-Le réseau de Benes

Le réseau de Benes est considéré comme une autre variation de l'hypercube, on l'obtient si on met bout à bout un réseau butterfly .

Le réseau de Benes de dimension d contient 2d +1 niveaux , chaque niveau a 2^d nœuds.

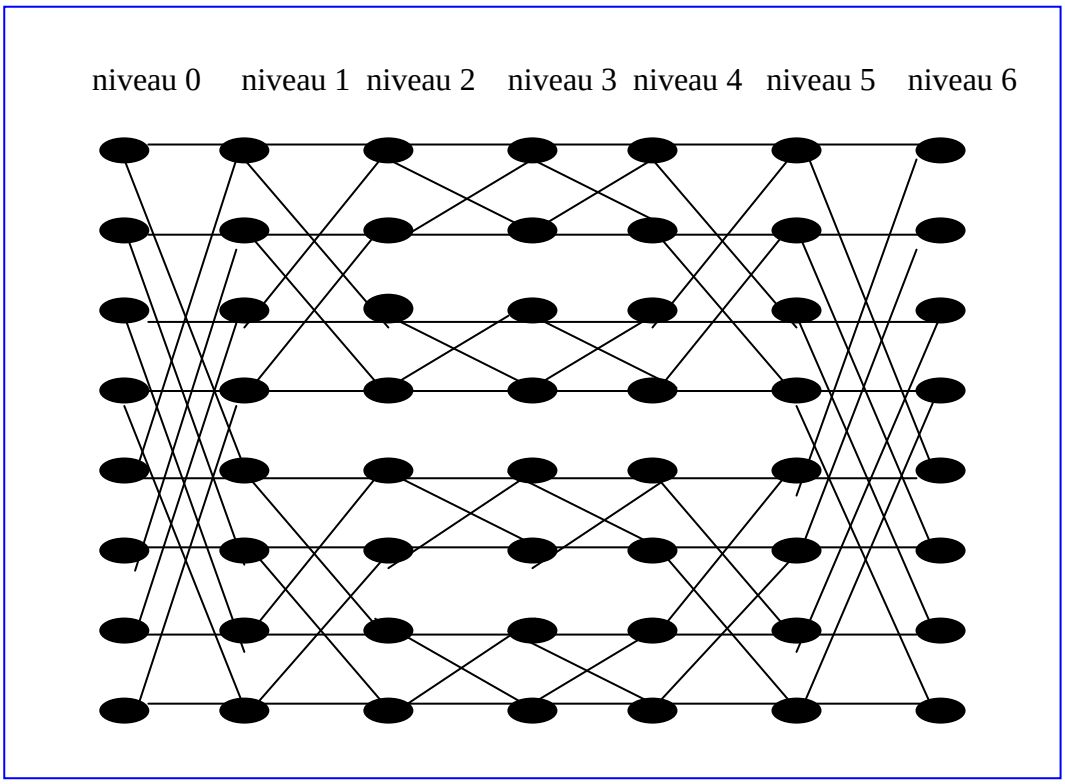
-Le premier et dernier (d+1) niveaux dans le réseau formé le butterfly de dimension d

(le niveau central au réseau du Benes est partagé par les butterfly).

Le réseau est très similaire au réseau butterfly , en termes de sa puissance Informatique et sa structure de réseau .

Dans le cas du réseau de Benes de dimension d, on peut avoir deux entrées pour chaque nœud du niveau 0 et deux sorties pour chaque nœud du niveau 2d et nous relions toujours chaque permutation des entrées aux sorties par des chemins d'arêtes disjointes. **[LEI92]**

Exemple :



réseau de Benes de dimension 3

3.4.6.3-Cube -connected-cycles

Le Cube_Connected Cycles (ou CCC) a été inventé par Archimède et remis à la mode par Preparata et Vuillemin**[FP,JV81]** . Il est organisé comme un hypercube de dimension N où chaque sommet est remplacé par un cycle de N PEs (Processor Elements) ; chaque PE est connecté dans le cycle et porte un des anciens liens de l'hypercube. Son degré est de deux dans le cycle et de 1 dans l'hypercube ; il est égal à trois en tout et il est fixe quelque soit la

dimension de l'hypercube considéré ; c'est l'avantage majeur des CCC sur les Hypercube (leur degré ne croît pas avec le nombre de processeurs). L'arête de dimension i qui est l'incident à un nœud d'un hypercube est alors relie au $i^{ème}$ nœud du cycle qui correspond au CCC. Le nombre de sommets de CCC est égal : $N.2^N$. Son diamètre est : $d_{CCC}=2^N-2+N/2$ pour N>3. **[EA93]**

Nous pouvons représenter chaque nœud par une paire (w, i) où i ($1\leq i\leq N$) est la position du nœud dans son cycle , et w (n'importe quelle adresse de N-bits) est l'étiquette du nœud dans l'hypercube qui correspond au cycle . Puis deux (w, i) et (ẃ, í) sont liés par une arête dans le (CCC) Si et seulement si 1 ou 2 soit réalisé :

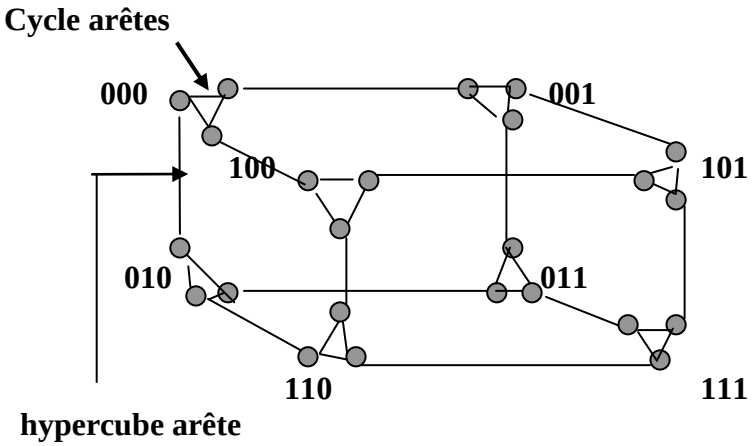
1)w =ẃ et i = í =± 1 mod N .

ou

2) i = í et w diffère de ẃ dans la position du $i^{ème}$ bit.

Les arêtes du premier type s'appellent les arêtes du cycle et les arêtes du deuxième type sont visées comme arêtes de l'hypercube . **[LEI92]**

Exemple:



Cube-Connected Cycle de dimension 3

3.4.7-L'importance de hypercube

L'hypercube a été focalisé par les différentes activités de plusieurs recherches. Il offre une structure d'interconnexion très riche,un degré logarithmique ainsi que son diamètre une transmission et un routage très simple, une tolérance aux pannes et une structure récursive. L'hypercube peut simuler d'autres réseaux d'interconnexion avec un facteur de charge minimum. Beaucoup de travaux de recherches ont montré que l'hypercube est un réseau d'interconnexion très puissant et possède une architecture capable de simuler d'autre réseau d'interconnexion tels que la maille, l'arbre, l'étoile et d'autres avec un minimum de facteur de

charge**[GA,EG75],[SB,FC,FL,AR86],[SB,IS89],[LB,SL,SD89],[SB,ZM,IS92], [SB,BC,MG,AR92], [AG,HW92], [LEI92], [BM,IS88], [YS,MS88].** Ils ont montré ainsi que l'hypercube une machine robuste et très tolérante aux pannes et elle se reconfigure dans la présence de fautes concernant des liens ou des nœuds **[SB,IS89],[MS,SL91],[JH,FL,MN87], [SL,AE88],[FP,RR89],[AW,RC,EM90].**

Finalement plusieurs recherches ont proposé des modifications sur la structure de l'hypercube pour montrer sa puissance de calcul **[SB,BC,MG,AR92],[KE,PB,TS,WS88], [AE,NN,BS88],[TL,HE92][FP,JV81],[SL,AE88]**.

4-CONCLUSION

Pour une machine à base d'architecture parallèle le choix d'une bonne topologie s'est varié au fur et à mesure du développement de l'architecture d'un ordinateur. Dans ce chapitre nous avons présenté une introduction aux architectures parallèles, ou nous avons cité les classifications les plus populaires : celle de flynn et celle dite la classification topologiques structurales d'architectures parallèles simples tel que les vecteurs, les mailles, les anneaux et les arbres binaires complets tout en citant leurs propriétés, leurs avantages et leurs inconvénients.

1-Introduction

Beaucoup des réseaux d'interconnexion ont été considérés comme une base pour les architectures de calcul parallèle. Le réseau d'interconnexion est un composant très important à la composition de l'ordinateur parallèle. L'architecte des réseaux est affrontée à un nombre irrésistible de choix a savoir : cross bar, butterfly, la maille, l'arbre et l'hypercube. Ce dernier est un réseau d'interconnexion populaire, puissant et très attractive grâce à ces propriétés tel que : la régularité, la symétrie, la connectivité forte, la largeur de bissection, et Hamiltonien. Malgré la puissance de ces propriétés, l'hypercube présente certains inconvénients tel que : le grand degré de nœud et un diamètre très grand ($\log_2 N$). Pour répondre à ces déficiences, plusieurs variations sur l'architecture de l'hypercube ont été faites. Celle qui maintient des propriétés semblables et qui réponde au problème du degré et du diamètre est régie par les règles suivantes :

1- Esfahanian et al ont introduit une classe de réseaux dénotée TQ_n hypercube croisé de Dimension n ''n-dimensional twisted cube '' pour $n \geq 3$, obtenu de l'hypercube en Échangeant les extrémités de seulement deux arêtes, TQ_n réduire le diamètre de n à (n-1) Et garde beaucoup des propriétés désirables de l'hypercube.

2- Shiau et al ont introduit une méthode d'échange systématique des multiples paires d'arêtes dans l'hypercube de dimension n, ce nouveau réseau conserve beaucoup des propriétés désirables de l'hypercube et réduit le diamètre à $\lceil (n+1)/2 \rceil$ ($\lceil x \rceil$ par excès a x).

3- Ces résultats (1,2) ont permit d'obtenir l'hypercube croise de dimension n ''n-dimensional multiply- twisted cube '' dénote MQ_n et avec un diamètre égal $\lceil (n+1)/2 \rceil$, une réduction presque de cinquante pour cent par rapport au diamètre de l'hypercube.

Notation

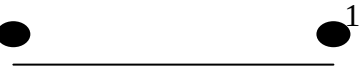
Deux chaînes binaires x=x1x0 et y=y1y0 de longueur 2 sont en relation de parité dénotée (X~y) si et seulement si :

$(X, y) \in \{(00,00), (10,10), (01,11), (11,01)\}$

2-Définition

Tout comme l'hypercube binaire ordinaire, l'hypercube croisé de dimension n (MQ_n) possède 2^n nœuds (sommets) et $n * 2^{n-1}$ arcs, les nœuds de MQ_n sont étiquetés par des chaînes binaire de longueur n. La seule différence réside dans les liens entre les nœuds. L'hypercube croisé de dimension n est un graphe non orienté défini récursivement comme suit :

1) n=1, MQ_1 est un graphe complet de deux nœuds dont les étiquettes sont 0 et 1.



2) pour n>1, MQ_n contient deux copies de MQ_{n-1} (hypercube croisé de dimension n-1) l'un préfixé par 0 dénote MQ^0_{n-1} et l'autre préfixe par 1 dénote par MQ^1_{n-1} ces deux copies sont jointes d'après la règle suivant :

Soient $U=0U_{n-2}$ …… ……U_1U_0

et $V=1V_{n-2}$……V_1V_0

(n étant plus grand que 0 , n>0) les étiquettes de deux nœuds.

- les deux nœuds U de MQ^0_{n-1} et V de MQ^1_{n-1} sont adjacents si et seulement si :

1) $U_{n-2}=V_{n-2}$ si n paire.

2) pour $0 \leq i < \lfloor (n-1)/2$, $U_{2i+1}U_{2i}$ et $V_{2i+1}V_{2i}$ sont en relation de parité.

($\lfloor x \rfloor$ c'est l'entier qui est égale au juste inférieur que x)

- A partir de la définition ci dessus, chaque nœud de MQ_n préfixé par 0 est adjacent à un seul nœud préfixé par 1 et vice versa. dans l'hypercube ordinaire, chaque arête est incidente à deux nœuds U, V si et seulement si, l'étiquette de U est différente à l'étiquette de V dans un seul bit, mais dans MQ_n (Hypercube Croisé) la liaisons entre deux nœuds ce fait à partir de règle suivant :

- Pour tout $n \geq 1$, ($U_{n-1}U_{n-2}$ ……U_1U_0 , $V_{n-1}V_{n-2}$……V_1V_0)

est une arête de MQ_n si et seulement si existe un ''L'' avec

1) U_{n-1} ……U_L = V_{n-1}……V_L .

2) $U_{L-1} \neq V_{L-1}$.

3) $U_{L-2} \neq V_{L-2}$ si L paire.

4) pour $0 \leq i < \lfloor (n-1)/2 \rfloor$, $U_{2i+1}U_{2i}$ et $V_{2i+1}V_{2i}$ sont en relation de parité.

À partir des conditions 1 et 2, les étiquettes U et V ont un préfixe identique et diffèrent dans le bit de la position L -1. Deux nœuds U,V sont différents à la position le plus gauche d, ceci implique que U est d- voisin de V et l'arête (U, V) est une arêtes de dimension d. Les conditions 3 et 4 testent la relation de parité.

3-propriétés de l'hypercube Croisé

Pour faciliter la discussion de les propriétés topologiques de MQ_n, on introduire un mécanisme pour identifier les sous graphes, notamment $\Gamma_{A,B}(G)$ est un sous graphe de G tel que chaque nœud du sous graphe préfixé par A ou B. $\Gamma_A(G)$ est un sous graphe de G tel que chaque nœud du sous graphe préfixé par A.

3.1-Composition et décomposition

3.1.1-Décomposition

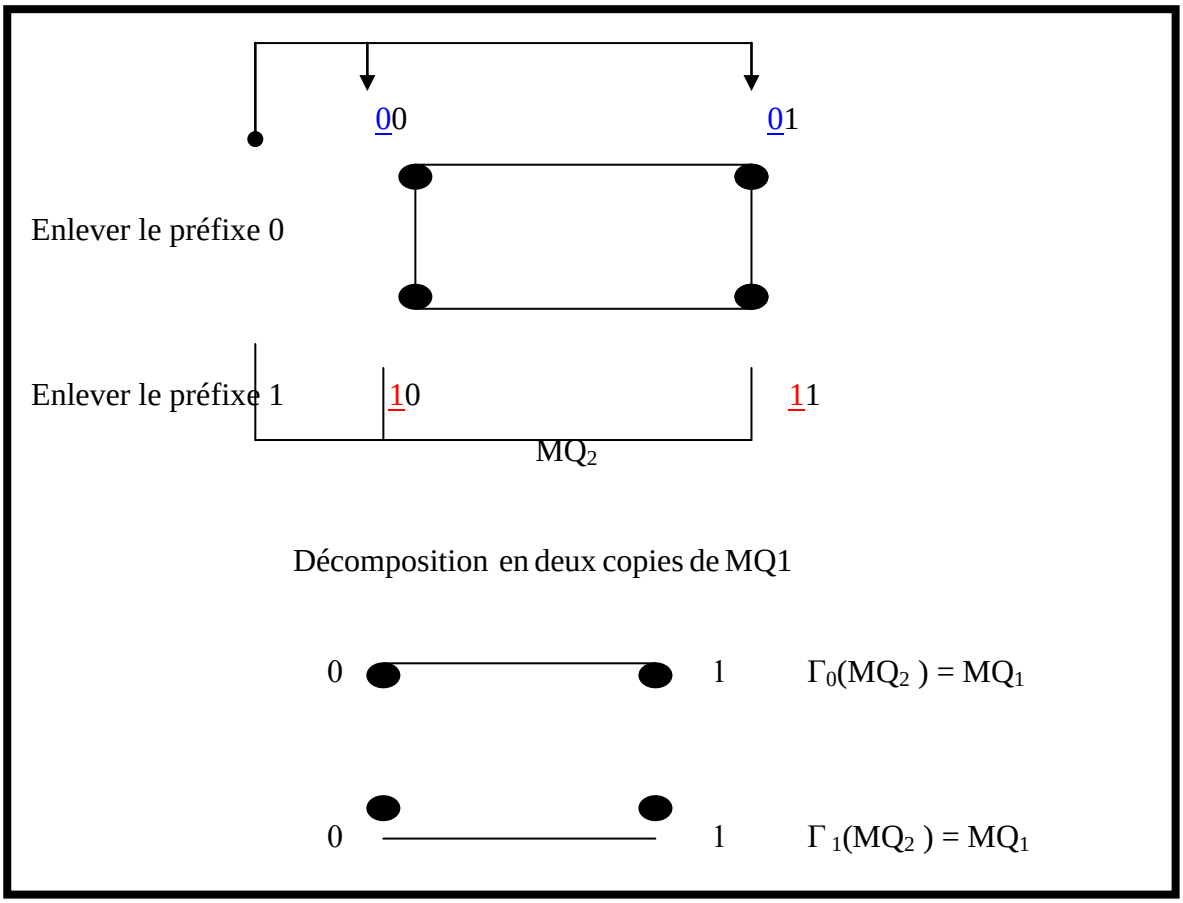
La construction de MQ_n est récursive car pour n>1 , MQ_n contient deux copies de MQ_{n-1} l'un préfixé par 0 et l'autre préfixé par 1 .

règle1

Pour n>=2, $\Gamma_0(MQ_n) = MQ_{n-1}$ et $\Gamma_1(MQ_n) = MQ_{n-1}$. de plus, l'isomorphisme est donné par la fonction qui enlève le préfixe 0 pour chaque étiquette des nœuds qui appartient à $\Gamma_0(MQ_n)$ et le préfixe 1 pour chaque étiquette des nœuds qui appartiennent à $\Gamma_1(MQ_n)$.

Exemple :

pour n=2 , $\Gamma_0(MQ_2) = MQ_1$ et $\Gamma_1(MQ_2) = MQ_1$, MQ_2 contient deux partie l'un préfixé par 0 et l'autre par 1 .

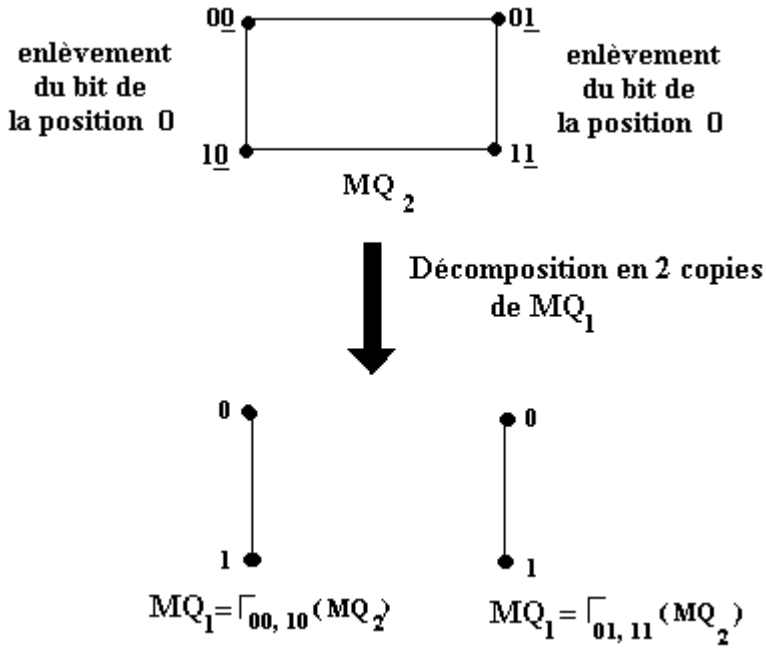


règle2

Pour tout K>=1 , $\Gamma_{00,10}$ ($MQ_{2\ K}$)= MQ_{2K-1} et $\Gamma_{01,11}$ (MQ_{2K})=MQ_{2K-1} . de plus, l'isomorphisme est donné par la fonction qui enlève le bit de position (2K-2) pour chaque étiquette de nœud.

Exemple:

pour K=1, $\Gamma_{00,10}$ (MQ_2) =MQ_1 et $\Gamma_{01,11}$ (MQ_2)= MQ_1 , enlève le bit de position 0 pour chaque étiquette de nœud .



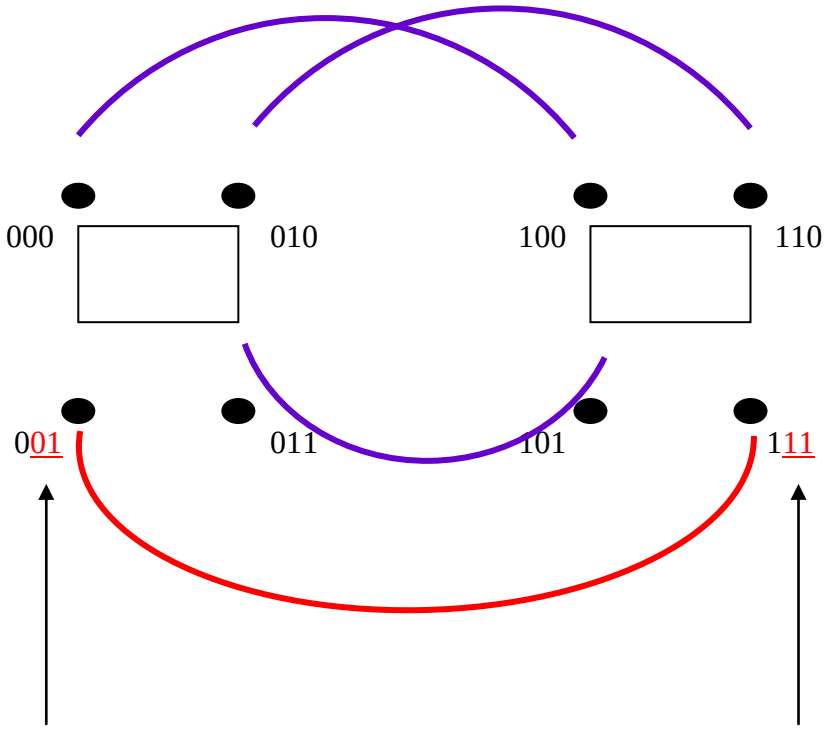
règle 3:

Pour tout K>=1, $\Gamma_{A, B}$ ($MQ2K_{+1}$)= MQ_{2K-1} pour tout A,B appartient {(001,111),(011,101), (000,100), (010,110)}. De plus, l'isomorphisme est donné par la fonction qui enlève les bits de position (2K-1) et (2K-2) pour chaque étiquette des nœuds qui appartient à $\Gamma_{A\ ,B}$ (MQ_{2K+1}) .

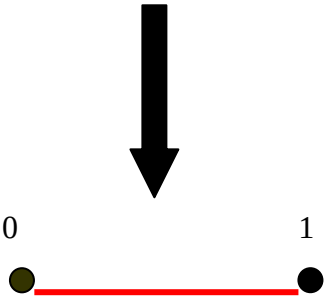
Exemple :

Pour K=1 , $\Gamma_{A\ ,B}$ (MQ_3) = MQ_1

Application de la règle pour un seul sous graphe $\Gamma_{001\ ,\ 111}$ (MQ_3) permet l'enlèvement des bits de position 1et 0 pour les étiquettes des nœuds (001 et 111) .



Enlèvement des bits de position 1 et 0

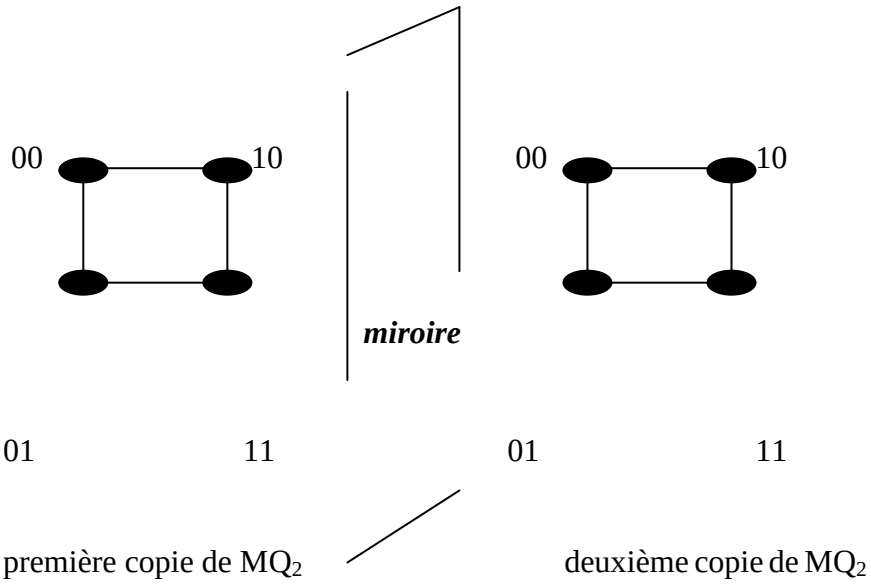


$\Gamma_{001,111}(MQ_3) = MQ_1$

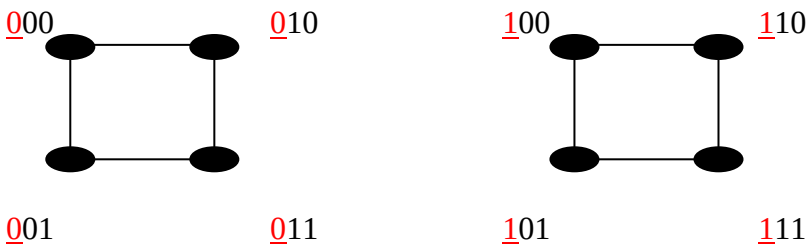
3.1.2-Composition

Un MQ_n de N nœuds peut être construit à partir de deux copies MQ_{n-1} tel que chaque copie contient N/2 nœuds, la liaison entre les nœuds de ces deux copies se fait par la définition 1.

Exemple :

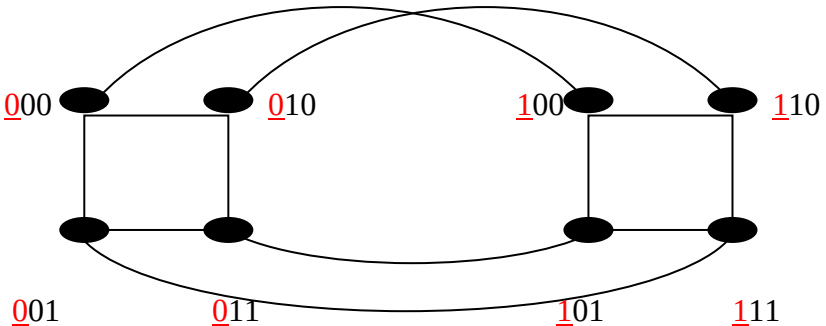


La composition des deux copies de MQ_2 en préfixant la première copie par « 0 » et la deuxième copie par « 1 » .



la construction de MQ_3 à partir de deux copies de MQ_2

(création des liaisons entre les nœuds)



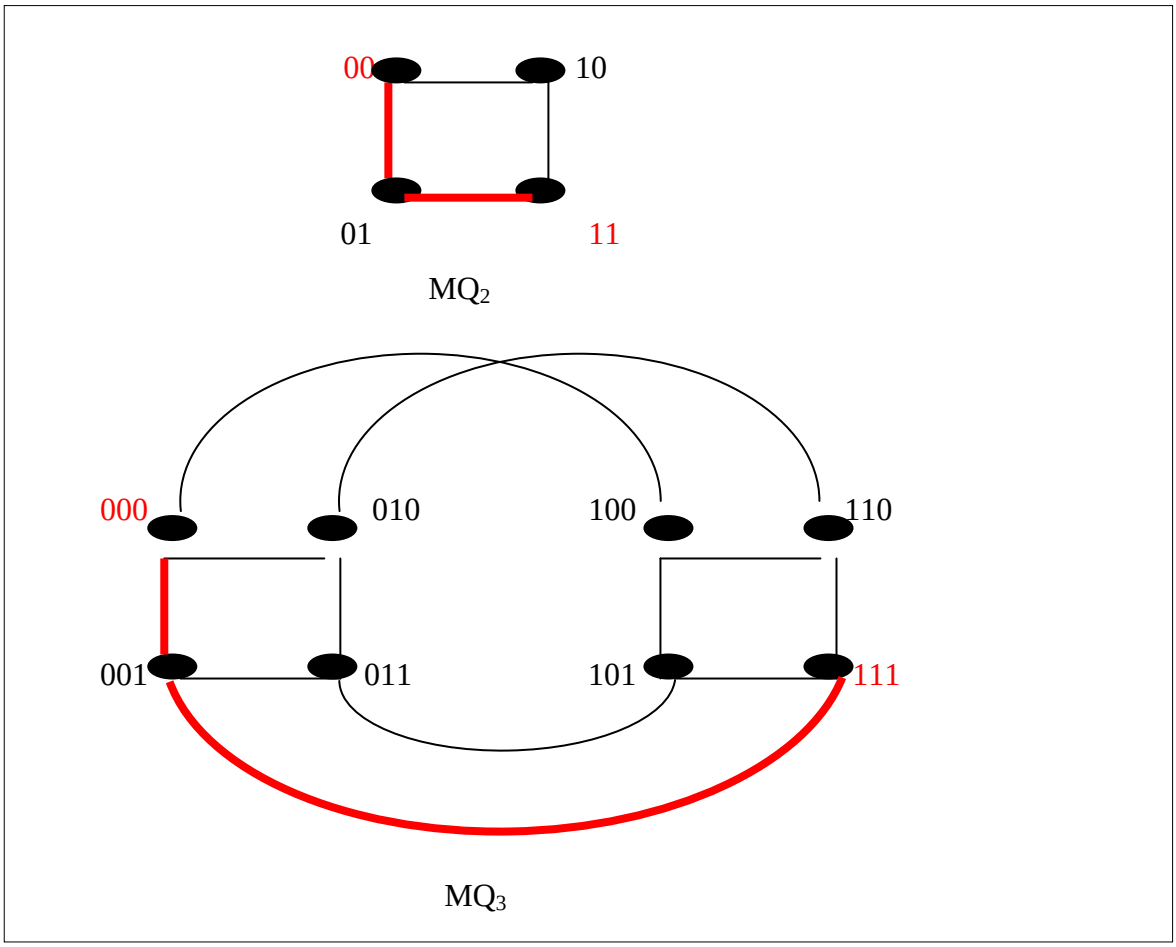
Le petit diamètre

MQ_n possède un petit diamètre égal $\lceil (n+1)/2 \rceil$.

Pour K>=1 , MQ_n et MQ_{n-1} possède même diamètre , $D(MQ_{2K})=D(M_{2K+1})= k+1$.

Exemple :

Pour K=1 , $D(MQ_2) = D(MQ_3) =2$

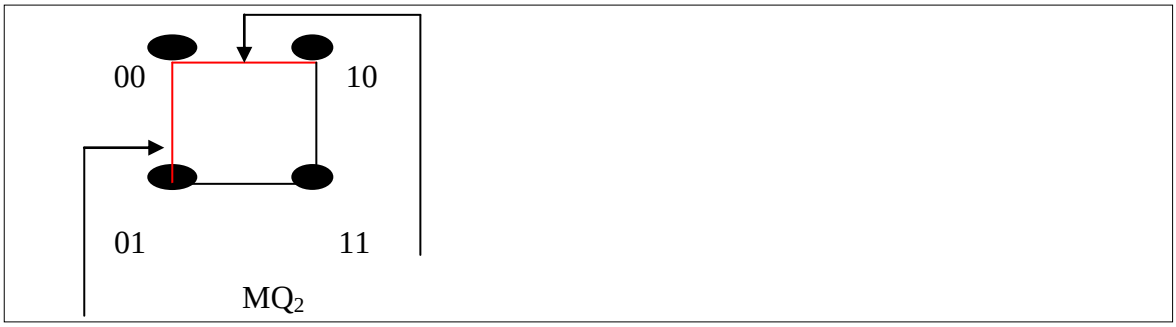


La connectivité

Un MQ_n possède une haute connectivité qui le rend plus désirable car ceci évite l'embouteillage, pour n>=1, $K(MQ_n) =n$.

Exemple:

Pour n=2 , $K(MQ_2) = 2$.



l'élimination de ces deux arêtes déconnecte le réseau (la connectivité = 2)

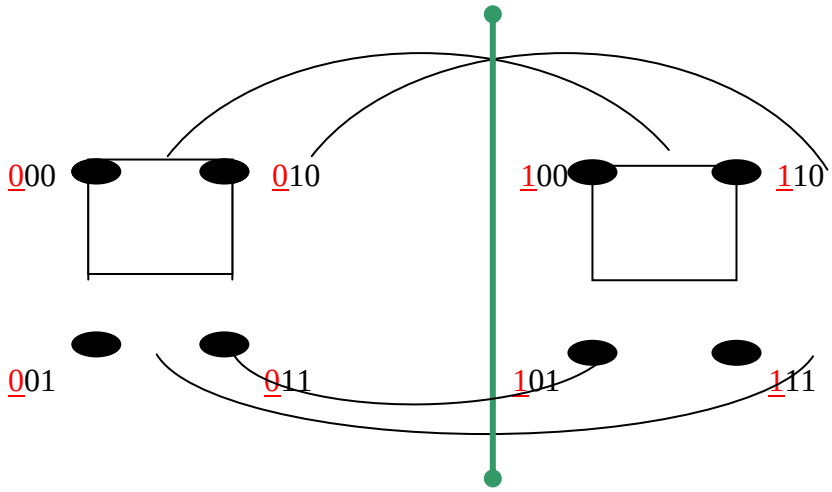
Largeur de bissection

MQ_n possède une grande largeur de bissection (N/2),(N est le nombre des nœuds) .

Exemple :

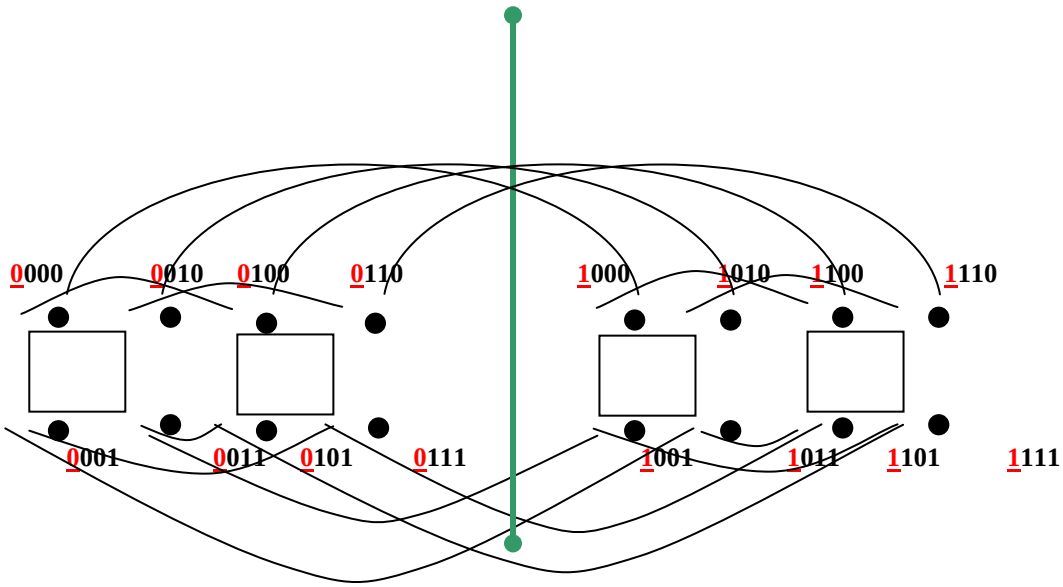
Hypercube croisé de dimension 3

Largeur de bissection=4



Hypercube croisé de dimension 4

Largeur de bissection=8



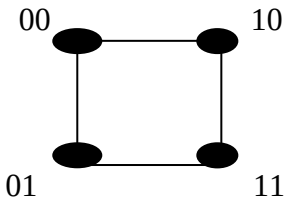
Topologie Régulière

MQ_n possède une topologie régulière, tous les nœuds ont le même degré, ce degré est égal à n.

Le cycle hamiltonien

Pour n >= 2 , tous les cycles dans MQ_n sont des cycles hamiltoniens .

Exemple : MQ_2 contient un cycle hamiltonien de quatre arêtes.



Le cycle hamiltonien est : (00,10), (10,11), (11,01), (01,00)

La symétrie

MQ_n est un nœud symétrie pour $n \leq 4$ **[BP93]**, mais pour $n \geq 5$, MQ_n n'a pas de symétrie de nœuds. Il y'a des nœuds qui sont similaires, mais ils ne sont pas tous similaires.

CQ_0, CQ_1, CQ_2, CQ_3 et CQ_4 ont tous la symétrie des nœuds. CQ_5 qui a 32 nœuds contient deux classes de nœuds, chaque classe contient 16 nœuds, les nœuds de chaque classe sont tous similaires. CQ_6 qui a 64 nœuds contient aussi deux classes de nœuds, chaque classe contient 32 nœuds, les nœuds de chaque classe sont tous similaires.

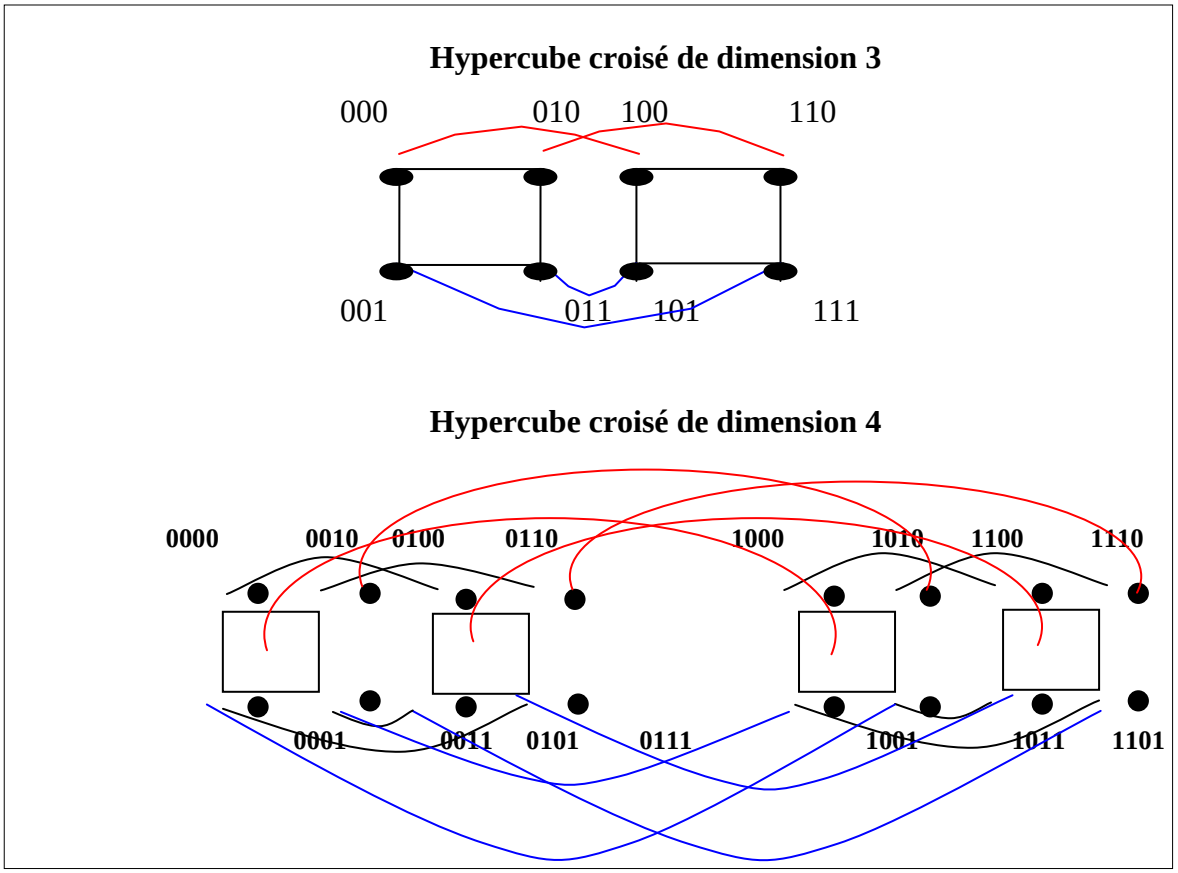
Et en général, MQ_n qui a 2^n nœuds contient aussi $2^{\lceil (n-4)/2 \rceil}$ classes de nœuds. Chaque classe contient $2^{n-\lceil (n-4)/2 \rceil}$ nœuds. Les nœuds de chaque classe sont tous similaires.

- MQ_n est un arc symétrie pour n<=3 , mais pour n>3 ce n'est pas le cas**[BP93].**

MQ_n est un graphe fortement connexe

MQ_n est un graphe fortement connexe du fait qu'il existe une chaîne qui relie tous les nœuds de MQ_n .

Exemple de construction d'hypercube croisé :



4-La communication des données

L'une des plus importantes composantes des réseaux de connexions est le mécanisme de communication des données. Dans les machines parallèles, la communication des données provoque des goulots d'étranglements qui sont dues souvent aux échanges d'informations entre les différents processeurs. Donc il est important d'obtenir une information correcte avec un temps raisonnable.

L'émission des données est une opération de communication très importante dans les réseaux de connexions. Le poids d l'arbre d'émission de données dépend du diamètre. L'hypercube croisé réduit le diamètre de moitié, ceci implique que le poids de l'arbre d'émission lui aussi est réduit de moitié. [KE92]et [SZ97] introduisent indépendamment tous les deux des algorithmes de routage et d'émission de données sur l'hypercube croisé.

5- Les opérations de base sur l'hypercube croisé.

L'hypercube a une capacité importante d'exécuter plusieurs opérations de base nécessaire à la conception des algorithmes parallèles. Ils apparaissent souvent comme sous problèmes dans

la résolution des problèmes majeurs. Citons par exemple le tri, considéré comme une tache commune exécutée souvent dans les machines parallèles. Dans le travail réalisé par [KE92], l'auteur montre que l'hypercube croisé réduit le nombres d'étapes dans la communication dans le tri par deux par rapport à l'hypercube ordinaire ainsi que le nombre d'étapes dans l'algorithme de communication d'un produit matricielle .

5.1-Le calcul associatif.

Les opérations d'association comme l'addition, la multiplication, la recherche du minimum ou du maximum et d'autres sont utilisées fréquemment et apparaissent souvent comme sous problèmes dans la résolution d'autres problèmes. Nous traitons l'opération d'addition dénotée + dans un ensemble X défini comme suit :

$\{x_0 , x_1 ,\ldots\ldots\ldots\ldots x_{k-1} \} \varepsilon X$

$y= x_0 + x_1 +\ldots\ldots\ldots\ldots\ldots\ldots + x_{k-1}$

nous supposons que chaque processeur pi $0 =< i <= 2n-1$ contient une valeur xi.

La somme y_0 après calcul est dans le processeur p_0.

Le symbole $\longleftarrow^{j}$ dénote le transfert de donnée entre deux processeurs adjacent par la liaison de dimension j.

La fonction BIT(j) retourne le $j^{iéme}$ bit de l'adresse d'un nœud.

L'algorithme suivant réalise l'opération d'addition.

ADDITION (X)

begin

for all P_i, $0 \leq i \leq 2^n - 1$, do

$y_i \longleftarrow x_i$

for j $\longleftarrow$ n **to** 1 **do**

for all Pi, $0 \leq i \leq 2^j - 1$, **do**

IF BIT (j) = 1 **then**

$temp_k \Longleftarrow^{j} y_i$, ou P_k est voisin dans la dimension j.

IF BIT (j) = 0 **then**

$y_i \longleftarrow y_i + temp_i$

end for

end for

end

5.2-L'opération de préfixage

Elle est considérée comme opération importante qui apparaît fréquemment dans la conception des algorithmes parallèles. Elle a été introduite en premier lieu par LADNER et Ficher dans la résolution des problèmes de l'addition binaire et par la suite elle a été utilisée par

plusieurs chercheurs dans la résolution des problèmes informatiques notamment dans la résolution des équations récurrentes [AG,HW92], dans les routages des paquets dans les réseaux de connexion et dans le tri rapide ou tri régulier [AD,IS89]

Soit (+) l'opération d'association binaire dans un ensemble X l'opération de préfixage est calculée d'une façon partielle en réalisant des sommes partielles

Supposons (+) est l'addition et $Y_i = x_0 (+) x_1 (+)\ldots\ldots\ldots\ldots..(+)x_i \ /0 \leq j \leq k-1$

La fonction BIT(j) et le résultat final sont définis de la même manière que dans l'algorithme précèdent.

L'algorithme de préfixage est le suivant :

BEGIN

FOR ALL p_i, $o \leq j \leq 2^n-1$ DO

$Y_i \longleftarrow x_i$

$T_i \longleftarrow x_i$

End for

FOR j = 1 to n **Do**

FOR ALL p_i, $o \leq j \leq 2^n-$ **DO**

$Temp_k$ ^{j} T_i ou p_k est voisin dans la dimension j-

$T_i \longleftarrow T_i + Temp_i$

If BIT (j) = 1 **THEN**

$Y_i \longleftarrow y_i$ (+) Tempi;

END FOR

END FOR

END

6-Conclusion

L'étude détaillée sur la variation de l'hypercube introduite dans [KE,PB,TS,WS88] appelée hypercube croisé indique que cette dernière préserve beaucoup de propriétés très attractives de l'hypercube. La plus importante est la réduction du diamètre par un facteur égal à deux. Tout cette réduction a agit sur le temps de la communication entre les différents processeurs de cette

architecture et sur le nombre d'étapes de calcul parallèle tout en réduisant respectivement le temps d'émission de données par deux et le nombre d'étapes dans l'addition et le préfixage utilisés dans le calcul scientifiques [SZ91].

L'étude nous a montré que l'hypercube croisé est tolérant aux pannes vu qu'il possède une topologie régulière , un degré de connectivité tr és haut et une symétrie des nœuds dans cinq classes ou dans ces dernières les nœuds sont tous similaires. Enfin cette étude nous a montré que l'hypercube croisé a une grande capacité de simuler d'autres architectures tel que l'arbre binaire complet, l'arbre à double racine, l'hypercube et le quad complete tree

1-Introduction

La mise en œuvre d'algorithme parallèle sur des architectures multiprocesseurs à mémoire distribuée a conduit au développement de la notion de plongement de graphe source G dans un graphe hôte H.

Bien souvent un algorithme distribué A est décrit en supposant l'existence d'une topologie logique S sur laquelle A est défini, les topologies logiques les plus utilisées, soit par exemple les arbres et les mailles.

La méthode qui permet d'implanter sur un réseau R un algorithme A et sa structure logique associés à S est la simulation de S dans R et consiste à plonger uniquement la structure de graphe source S dans le graphe hôte R.

2-Problèmes du plongement

Il existe une grande variété de problèmes de plongement des graphes. Parmi ces problèmes :

1-Organisation des processus sur des réseaux de processeurs

Dans les problèmes de l'organisation des processus sur un réseau de processeurs, nous représentons tous les processus originaux et le réseau des processeurs par des graphes : Si le processus peut être décomposable naturellement avec un ensemble de sous processus qui s'exécutent parallèlement avec des communications entre les sous processus, le graphe peut être obtenu comme suit :

Chaque processus correspond à un nœud du graphe et chaque communication entre deux sous processus par arête entre deux nœuds correspondant dans le graphe. Dans la représentation du réseau du processeur par un graphe, chaque processeur correspond à un nœud dans le graphe, et la liaison de communication entre deux processeurs correspond à une arête entre les deux nœuds correspondants. Par plongement efficace du graphe représentant le réseau des processus originaux dans le graphe représentant le réseau des processeurs, nous aurons une organisation efficace.

2-Simulation d'une architecture à base du réseau sur une autre architecture.

Dans le problème de simulation d'une architecture réseau A par une autre architecture réseau B, nous représentons les deux architectures par les graphes correspondants. Dans le calcul parallèle beaucoup de problèmes de calcul peuvent être formés par des plongements de graphe qui sont très utiles et surtout par la portabilité des algorithmes efficaces sur une variété d'architectures tels que le tri, le calcul matriciel, traitement d'image …etc. **[AE,LN,BS88]**

3-Définitions

Définition1

Soient G et H deux graphes tel que :

Un plongement de G dans H est un couple (P, R) où :

- P est une application injective V (G) dans V (H)
- R est une application injective associant à chaque arête [u, v] de G à une chaîne R (u, v) qui relie P (u) et P (v).

Définition2

Un plongement du graphe G dans graphe H est défini par la donnée d'une application injective Q de l'ensemble de sommets de G dans l'ensemble de sommets de H et d'une application PQ de l'ensemble des arêtes de G dans l'ensemble des arêtes de H, qui associe à chaque arête (x, y) de G, une chaîne reliant les sommets Q(x) et Q(y) dans H. **[AE,LN,BS88]**

4-Type de plongement

Il existe trois types de plongement d'un graphe source G dans un graphe hôte H à savoir :

- ✓ Plusieurs pour un (Many by one) ;
- ✓ Un pour plusieurs (One by many) ;
- ✓ Un pour un (One by one)

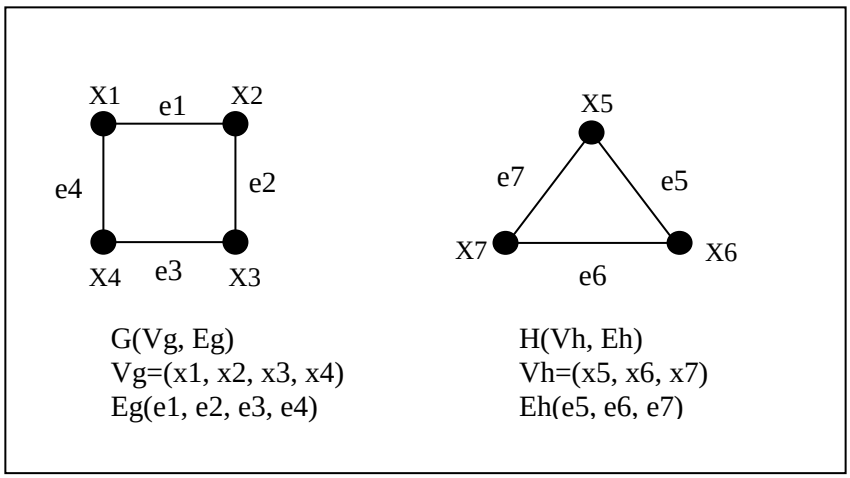
4.1-Plusieurs pour un (many by one)

Il est défini par la correspondance de plusieurs nœuds de G à un seul nœud de H. Il est généralement utilisé quand la cardinalité de G est strictement supérieure à celle de H :

|V (G)|> |V (H)|

G=(V(G), E(G)), H= (V(H), E(H))

Exemple:



Le plongement φ est défini par:

Plongement des nœuds :

φ(x1)=x5

φ(x2)=x5

φ(x3)=x6

φ(x4)=x7

Plongement des arêtes :

φ(e4)=e7

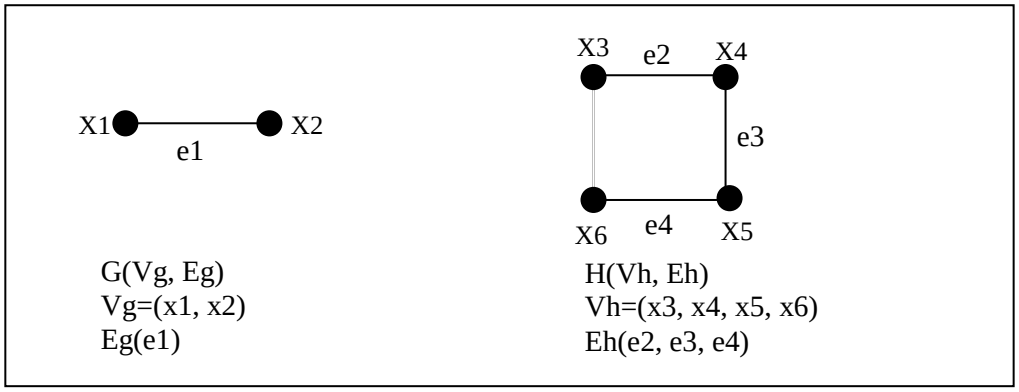
φ(e3)=e6

φ(e2, e1)=e5

4.2-Un pour plusieurs (one by many)

Il est défini dans le cas ou la cardinalité de G est strictement inférieure à celle de H : |V (G)|< |V (H)|. Dans ce type de plongement la notion d'adjacence est très importante.

Exemple :



Le plongement φ est défini par:

Plongement des nœuds :

φ(x1)=x3

φ(x2)=x6

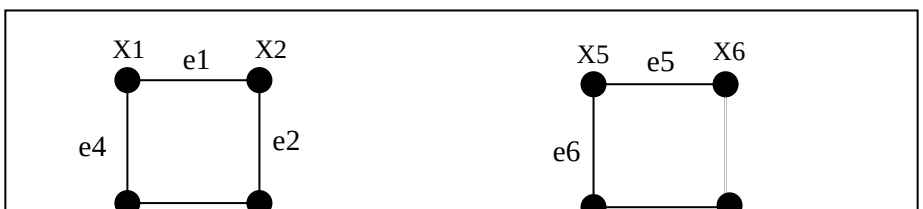
Plongement des arêtes :

φ(e1)=(e2, e3, e4)

4.3-Un pour un (one by one)

Ce type de plongement est souvent utilisé dans le cas où la cardinalité de l'ensemble des nœuds est égale ou presque à celle de H. Il fait correspondre un nœud de G à un nœud de H.

Exemple :



Le plongement φ est défini par:

Plongement des nœuds :

φ(x1)=x5

φ(x2)=x6

φ(x3)=x7

φ(x4)=x8

Plongement des arêtes :

φ(e1)=e5

φ(e3)=e7

φ(e4)=e6

φ(e2)=(e5, e6, e7)

5-Les mesures de qualité d'un plongement

Plusieurs variétés de plongement des architectures à base de réseau de connexion ont été étudiées dans la littérature**[RA,AR82], [SB,BC,MG,AR92], [SB,FC,FL,AR86], [SB,II85], [LB,SL,SD89], [SB and all92], [SB,IS89], [AG,HW92], , [JJ,SL,SD90], , [BM,IS88].** Ces différentes variétés de plongement diffèrent principalement dans l'optimisation des mesures utilisées [GH,KM,AR83] à savoir :

- ✓ Une mesure sur les délais ou la dilatation du graphe de plongement ;
- ✓ Une mesure sur l'expansion du graphe de plongement ;
- ✓ Une mesure sur le facteur de charge des nœuds du graphe de plongement ;
- ✓ Une mesure sur la congestion dans les arêtes du graphe de plongement.

5.1-La dilatation

La dilatation est une mesure de l'éloignement des images dans le graphe destination des sommets voisins dans le graphe de départ. **[DT00]** Autrement dit la dilatation d'un arc (x,y) de graphe G (graphe de départ) est le maximum des distances entre les images de (x,y) dans H

(graphe hôte). La dilatation d'un plongement Q d'un graphe G dans un graphe H, notée par dil(Q) est la longueur maximale des chaînes PQ(x,y) de H associées aux arêtes(x,y) de G. Dans le cas où l'on considère la plus courte longueur de PQ (x,y) est égale à la distance Dh (Q(x), Q(y)) et la dilatation s'exprime uniquement en fonction de Q, par : Dil (Q)=max. Dh (Q(x), Q(y)) x, y (G) **[AE,LN,BS88]**

Exemple :

La dilatation permet de mesurer le délai de communication dans le graphe de plongement concernant la communication entre deux nœuds du graphe source.

5.2-La congestion

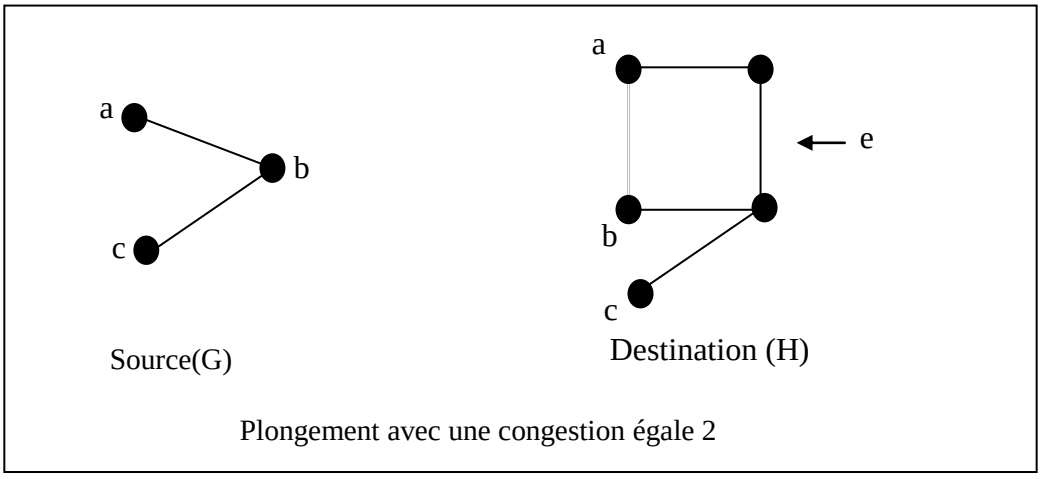
Définition1 :

La congestion d'un plongement Q d'un graphe G dans un graphe H notée par cong (Q) est le maximum (sur toutes les arêtes de H) du nombre de chaînes PQ (xy) de H qui les contient. **[DT00]**

Définition2:

La congestion d'une arête e de H est le nombre maximal des images des arêtes du graphe source G passant par une arête hôte H

Exemple :



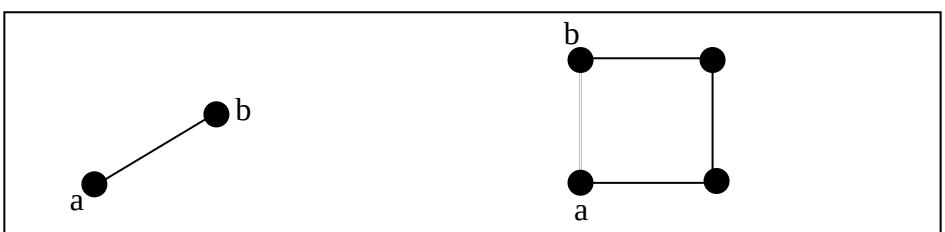
Plongement avec une congestion égale 2

5.3- L'expansion

L'expansion d'un plongement Q d'un graphe source G dans un graphe hôte H, notée par exp(Q) est le rapport du nombre de sommets de H sur le nombre de sommets de G.

exp(Q)=|X(H)|/|X(G)|,l idéal est d'avoir exp=1. **[AE,LN,BS88]**

Exemple :



Elle permet de décrire une mesure sur l'utilisation des processeurs dans un graphe hôte H.

5.4-Le facteur de charge

Définition :

Cette mesure décrit le nombre de processus dans G assignés à un seul processeur dans H, elle définit le maximum de nœuds de G correspondant à un seul nœud de H.

Si ei appartient à l'ensemble des nœuds (sommets) du graphe G.
Q l'application injective du plongement.
On définit le facteur de charge par :
Fg=max. nombre Q (ei) (quel que soit ei)

Le plongement d'un graphe source G dans un graphe hôte H s'effectue par les étapes suivantes :

1-Calcul de la cardinalité du graphe source.
2-Choix de la méthode de plongement à utiliser:
Il existe deux méthodes :

- **Le plongement par excès :**

Dans ce cas, on choisit un graphe destination dont la cardinalité est juste supérieure ou égale à celle du graphe source.

- **Le plongement par défaut :**

A l'inverse de la première méthode, le graphe destination a une cardinalité juste inférieure à celle du graphe source.

Plongement des nœuds et des arêtes :

Plongement des nœuds :

Cette étape consiste à plonger les nœuds du graphe source S dans le graphe destination tout en respectant les voisinages entre les nœuds.

Plongement des arêtes :

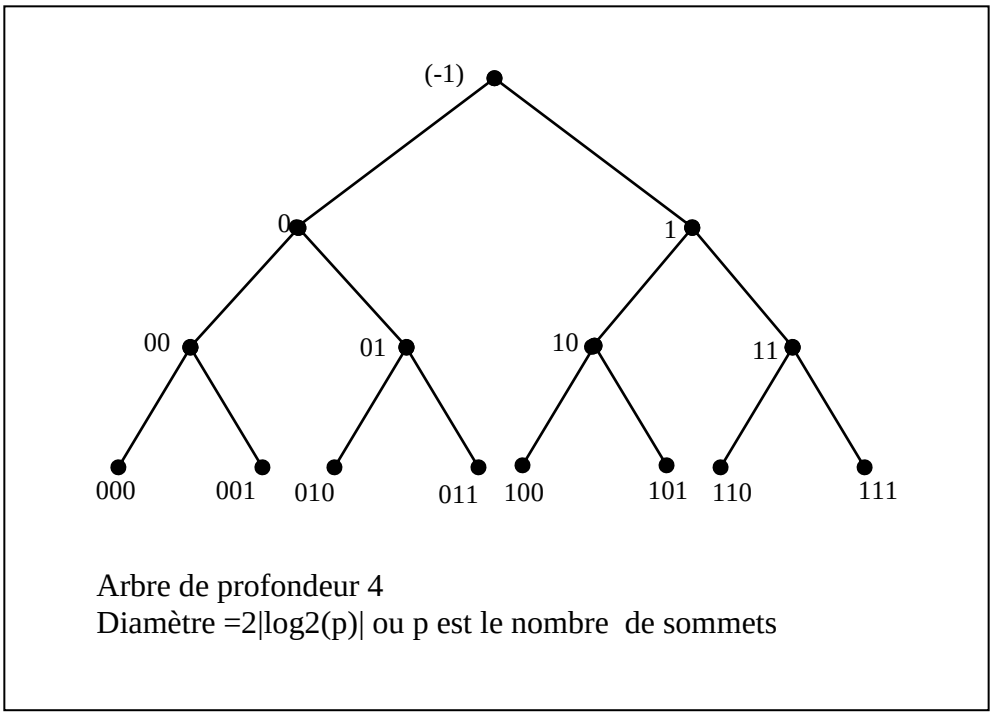
Le plongement des arêtes est basé sur le plongement des nœuds telle qu'une arête source qui peut correspondre à une arête ou un chemin dans le graphe hôte.

6-Plongement d'un arbre binaire complet dans un hypercube croisé

Le plongement se fait d'une façon telle qu'elle respecte le voisinage entre les nœuds.

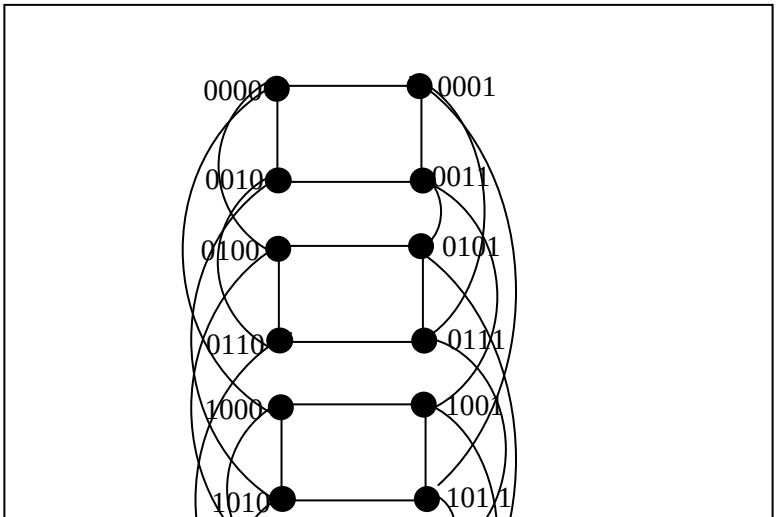
Si (ei, ej) deux nœuds voisins dans le graphe source (arbre binaire complet) seront voisins dans le graphe destination (hypercube croisé)**[EA93]**

Le schéma de graphe source est comme suit :

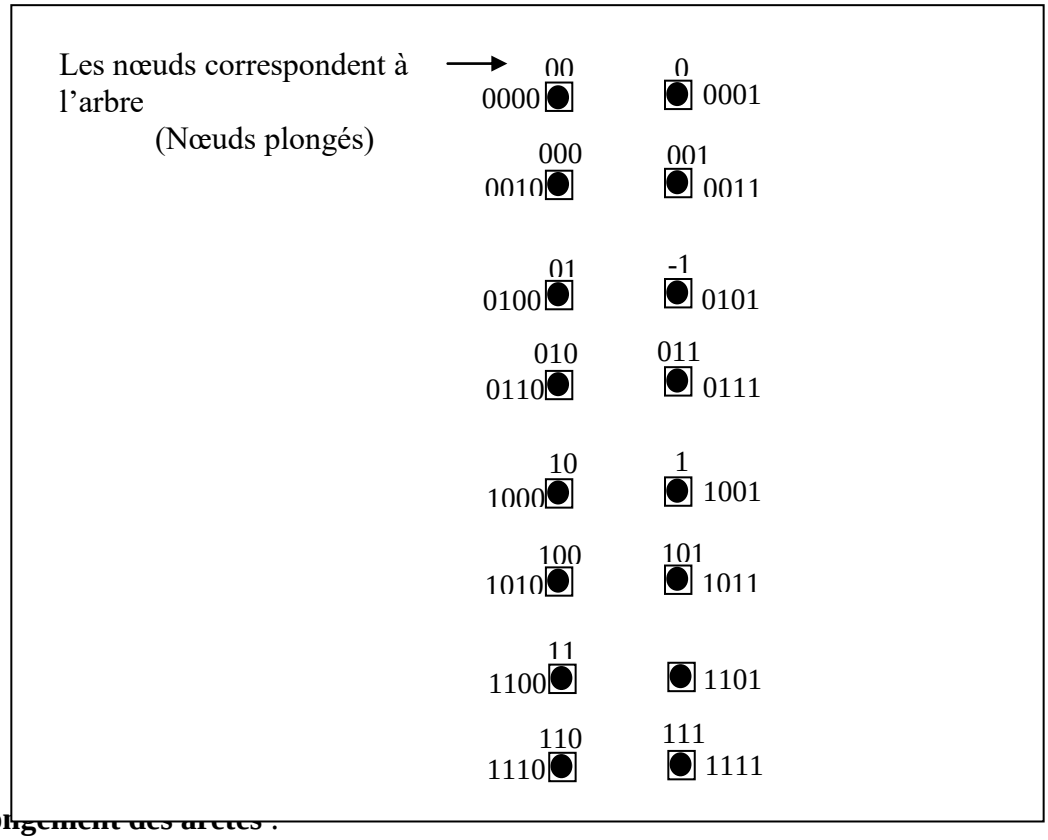


Arbre de profondeur 4
Diamètre =2|log2(p)| ou p est le nombre de sommets

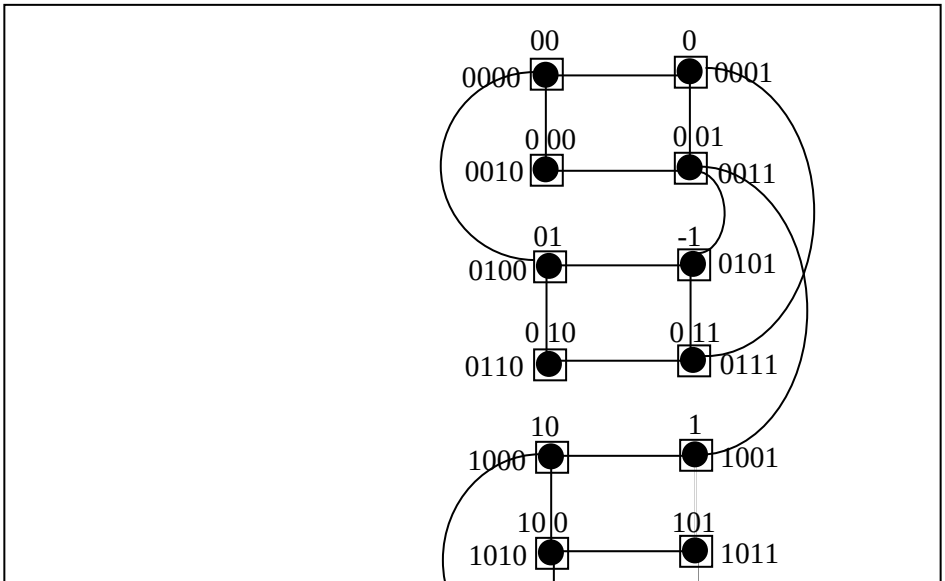
Le schéma de graphe destination (hypercube croisé)



a)Plongement des nœuds :



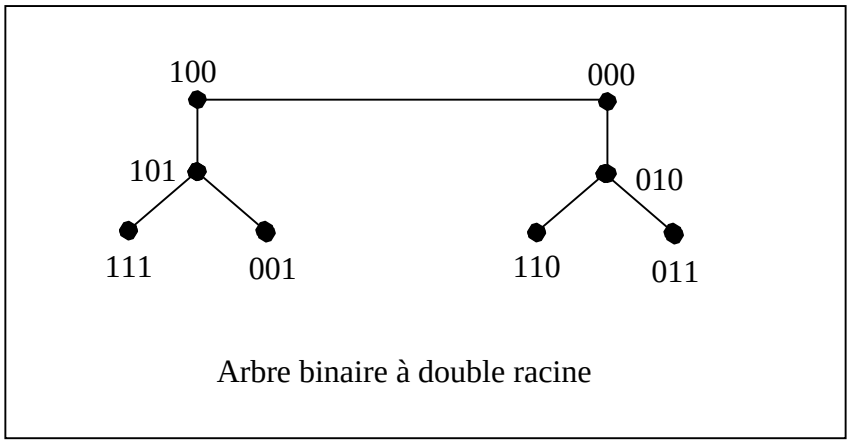
b) Plongement des arêtes :



Plongement d'un arbre binaire à double racine dans un hypercube croisé :

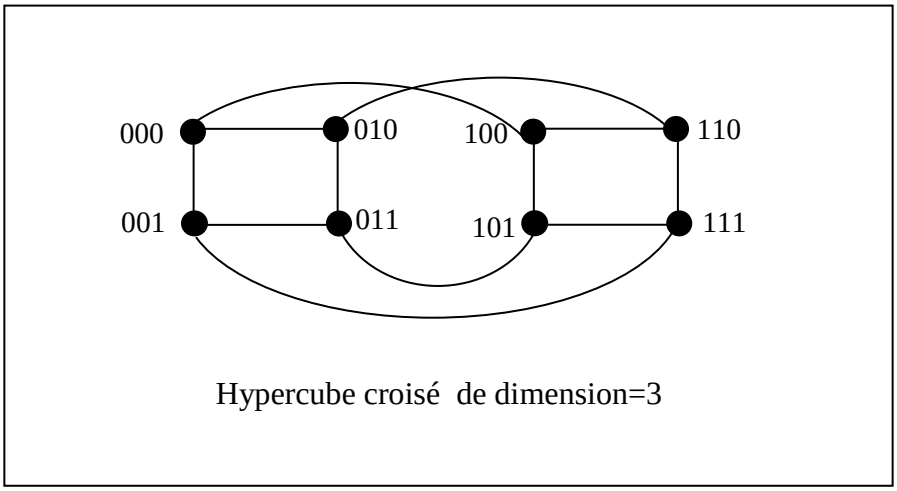
L'arbre binaire à double racine est un graphe partiel de l'hypercube croisé avec dilatation = 1et facteur de charge=2 et cogestion=1 et l'expansion=1.

Le schéma de graphe source est comme suit :



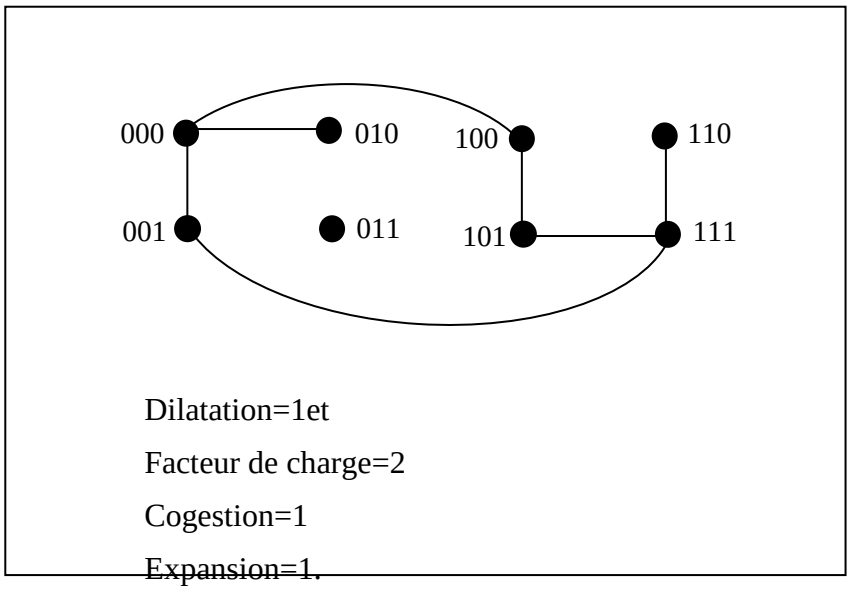
Arbre binaire à double racine

Le schéma de graphe destination (hypercube croisé)



Hypercube croisé de dimension=3

Le schéma de plongement :



7-CONCLUSION

Un ensemble de processeurs interconnectés peuvent être facilement modélisés par un graphe, dont chaque nœud représente un processeur unique , et chaque arête reliant deux nœuds si leurs processeurs sont connectés.

Lorsque on a besoin d'effectuer le passage d'un graphe à un autre pour satisfaire nos nouvelles demandes, on a besoin d'une application précise qui permet de garder les propriétés de base de chaque graphe et de gagner plus de qualité. Cette application est appelée le plongement d'un graphe source dans un autre graphe destination. Elle consiste d'appliquer les nœuds du graphe source dans les nœuds du graphe destination, ainsi que les arêtes de graphe source ont un simple chemin dans le graphe destination. Dans ce chapitre nous avons présenté une définition des différents types de plongement, et de ses mesures de

qualités. Egalement, nous avons cité quelques graphes qui définissent des structures sources différentes comme les vecteurs, les anneaux, les grilles et les arbres, et aussi la manière dont ils peuvent être facilement plongé dans un autre graphe destination hypercube croisé avec une qualité optimale.

Enfin nous pouvons dire que le plongement permet d'assurer la portabilité d'une architecture source à une autre architecture plus efficace si l'architecture source ne satisfait plus les besoins, tout en préservant les propriétés de base, et en ajoutant encore d'autres performances et avantages.

1-Introduction

La maille est l'une des architectures à réseau d'interconnexion simple. Beaucoup d'algorithmes ont été conçus sur cette architecture tels que : le calcul matriciel, traitement d'images, …etc.

Cependant, la maille présente un ensemble d'inconvénients qui agissent sur l'efficacité des algorithmes (temps d'exécution), parmi lesquels :

1- Un grand diamètre qui est égal à (N+M-2), où N est le nombre de lignes de la maille et M est le nombre de colonnes de la maille.

2- Une petite largeur de bissection qui égale à :

$$\begin{cases} \text{Min (N, M)} & \text{si Max (N, M) paire.} \\ & \text{ou} \\ \text{Min (N, M) + 1} & \text{si Max (N, M) impaire.} \end{cases}$$

C'est-à-dire, si on supprime de la maille N ou M arêtes, tout le réseau tombe en panne.

L' hypercube croisé est une architecture récente très attractive et populaire car elle offre les avantages (qui répondent pratiquement aux inconvénients de la maille) suivants:

1. Un petit diamètre qui est égal à $\lceil (d+1)/2 \rceil$,où d est la dimension de l'hypercube croisé.
2. Une grande largeur de bissection qui est égale à (N/2), où N est le nombre de nœuds dans l'hypercube croisé ($N = 2^d$).

Vu ces besoins et l'inexistence des deux architectures parallèles (la maille et l' hypercube croisé), notre système nécessite la simulation de chaque architecture (source et hôte).

Il faut, en premier lieu simuler l'architecture source (la maille), et en second lieu l'architecture hôte (l'hypercube croisé), et enfin le plongement de la première dans la deuxième.

Alors, il faut décomposer notre système en un ensemble d'objets en interaction.

L'approche qui répond à nos besoins est la conception orientée objet dans laquelle le système est vu comme une collection d'objets en interaction. A chaque objet est associé un ensemble de méthodes autorisées manipulant les données de ce même objet.

Dans l'approche de conception orientée objet il y a la notion d'encapsulation, c'est-à-dire que les données d'un objet ne sont accessibles que par leurs méthodes.

Notre système se repose sur deux sous systèmes :

- Le premier sous système est le plongement « One by One ».
- Le deuxième sous système est le plongement « Many by One » .

Chaque sous système est décomposé en un certain nombre d'objets.

La spécification de chaque sous système consistera à définir les méthodes de ses objets.

Il existe trois approches de simulation, qui sont :

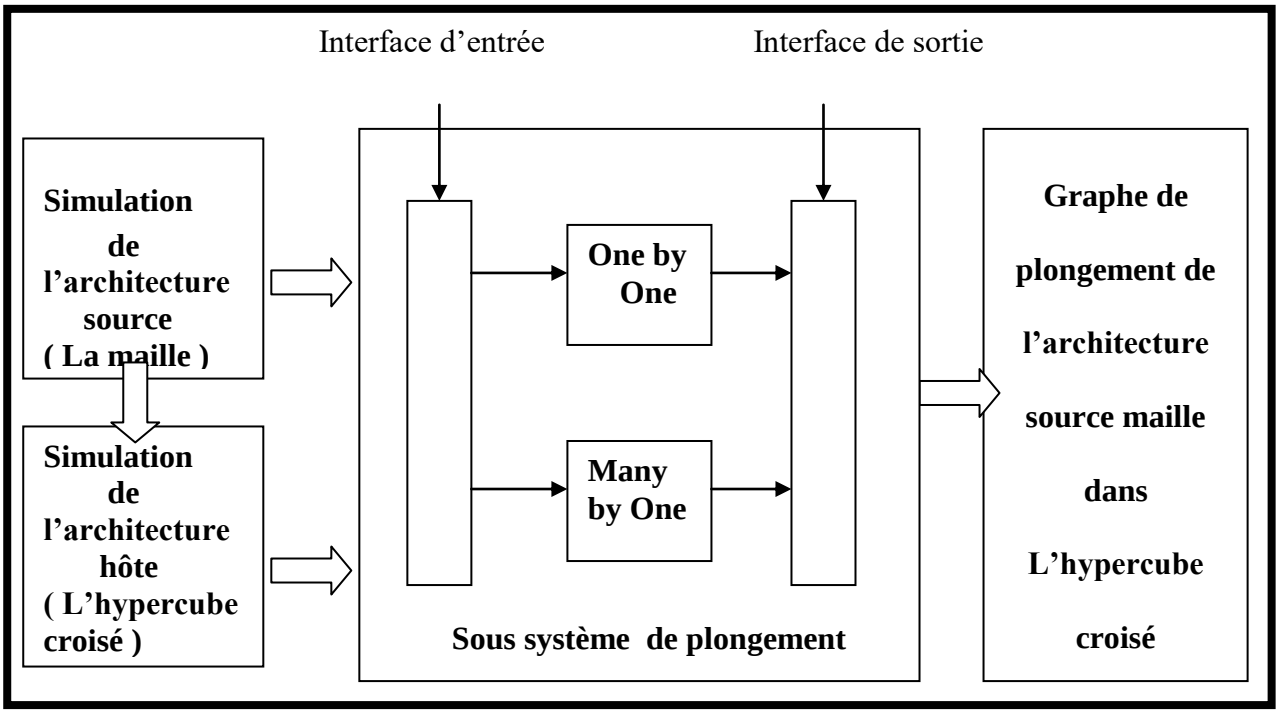
- Simulation par évènement.
- Simulation par processus.
-. Simulation par activité.

Nous avons choisi cette dernière pour représenter l'architecture de notre système de simulation.

2-Architecture du système

Notre système est composé de trois objets essentiels :

a. Simulation de l' architecture source (la maille).
b. Simulation de l' architecture hôte (l' hypercube croisé).
c. Emulation du plongement de la maille dans l'hypercube croisé.

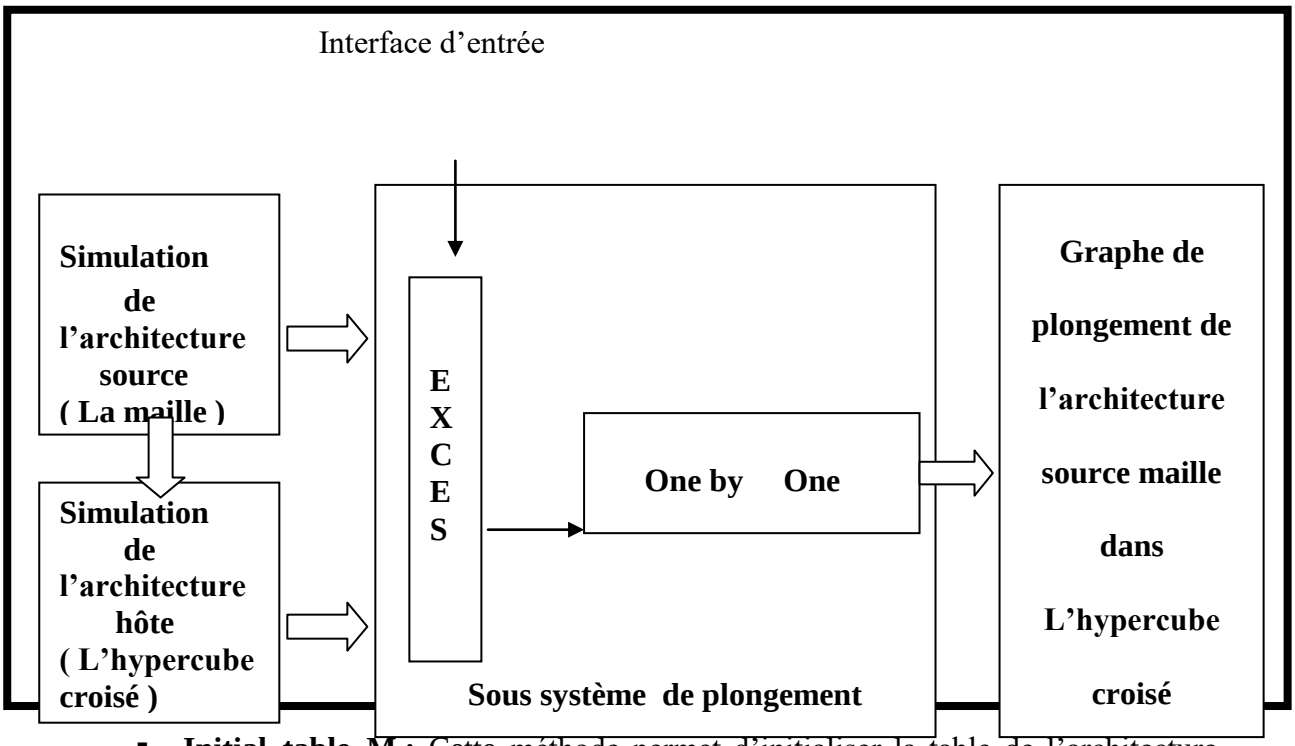


2.1-Le sous système : « Plongement One by One »

Il est composé de trois objets à savoir :

- ✓ Simulation de la maille
- ✓ Simulation de l'hypercube croisé ;
- ✓ Emulation du plongement « One by one » de la maille dans l'hypercube croisé.

2.1.1- Architecture du sous système



- **Initial_table_M :** Cette méthode permet d'initialiser la table de l'architecture source par une valeur entière (différente de 0 et 1).

2.1.1.1-Simulation de la maille

L'activité principale de cet objet est l'interprétation structurelle de l'architecture source (la maille).

Les paramètres d'entrée :

Deux paramètres d'entrée qui sont :

N : le nombre de lignes de la maille.

M : le nombre de colonnes de la maille.

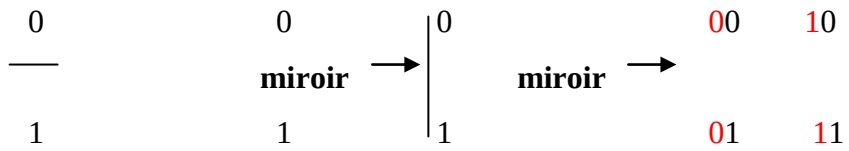
N et M : deux valeurs entières comprises entre (1 et 128) .

Le résultat de cette activité : la construction de la table de l'architecture source.

La structure de la table :

Création Nœuds_M : Elle permet de créer les nœuds de la maille en appliquant l'effet miroir, qui donne à chaque application deux copies, on préfixe l'une par « 0 » et l'autre par « 1 ».

Exemple :



Création_Voisin_M : Elle permet de créer les voisins de chaque nœud à partir la table des nœuds de la maille.

Test : permet de vérifier à partir des valeurs de N et M le type de la maille qui doit être créer.

Si (N < M) alors, il faut créer une maille rectangulaire horizontale.

Si (N > M) alors, il faut créer une maille rectangulaire verticale.

Si (N = M) alors, il faut créer une maille carrée.

2.1.1.2- Simulation de l'hypercube croisé

Activité1 : le choix entre :

- la création de l'hypercube croisé correspondant par excès.
- la création de l'hypercube croisé correspondant par défaut.

On utilise la création de hyper cube croisé correspondant par excès .

Activité2 :

C'est l'activité qui permet l'interprétation structurelle de l'architecture hôte (l'hypercube croisé).

La contrainte d'entrée est la validation de la construction de la table de l'architecture source (la maille).

Le résultat de cette activité est la construction de la table de l'hypercube croisé.

La structure de la table :

C'est une matrice de (2^d) lignes et (d+1) colonnes, où d est la dimension de l'hypercube croisé. Chaque ligne contient un nœud avec ses d-voisins.

La table a la structure suivante :

Nœud	**Voisin1**	**Voisin2**	**…**	**Voisin d**

Les méthodes :

Un test de création de la table de l'architecture source est généré.

Calcul_dimension_H : C'est une fonction qui permet de retourner la dimension par excès de l'hypercube croisé, calculée à partir de la dimension de la maille.

Initial_table_H : Elle permet d'initialiser la table de l'architecture hôte par une valeur entière (différente de 0 et 1).

Création_Nœuds_H : Elle permet de créer les nœuds de l'hypercube croisé en utilisant l'effet miroir, qui donne à chaque application deux copies, on préfixe l'une par « 0 » et l'autre par « 1 ».

Création_Voisin_H : Elle permet de créer les voisins de chaque nœud de l'hypercube croisé selon le principe indiqué dans la règle donnée par :

T.Shiau et ***al*** (définie dans le chapitre de l'hypercube croisé).

Exemple :

Le nœud 0 : « 000 » possède trois voisins « 001, 010, 100 ».

Le nœud 1 : « 001 » possède trois voisins « 000, 011, 111 ».

Cette méthode donne comme résultat la table de l'architecture hôte.

2.1.1.3-Emulation de plongement One by One

Activité1 : Interprétation structurelle du plongement.

Activité2 : Interprétation graphique du plongement.

Activité3 : Les mesures de qualités du plongement.

Interprétation structurelle du plongement :

Un test de validation de la construction de la table de l'hypercube croisé est généré.

Cette activité est composée de trois sous activités :

-Plongement des nœuds.

-Plongement des arêtes.

-Plongement des nœuds et des arêtes.

Le résultat de cette activité est :

1. La construction de la table de plongement des nœuds
2. La construction de la table de plongement des arêtes.
3. La construction de la table de plongement des nœuds et la table de plongement des arêtes.

La structure de la table de plongement des nœuds :

C'est une matrice de (N*M) lignes et de 2 colonnes, chaque ligne représente la correspondance entre un nœud de la maille et un nœud de l'hypercube croisé, et chaque colonne représente l'adresse binaire du nœud dans les deux architectures.

La structure de la table est la suivante :

Nœud de la maille	**Nœud de l'hypercube croisé**

La structure de la table de plongement des arêtes :

C'est une matrice de W lignes, où W = 2*((N-1)*M+(M-1)*N) et de 3 colonnes.

Chaque ligne représente une arête de la maille, le chemin correspondant de l'hypercube croisé et la dilatation.

La structure de la table est la suivante :

Arête_maille	**Chemin_hypercube_croisé**	**Dilatation**

Les méthodes :

Plongement des nœuds :

Initial_table_nœuds : Elle permet d'initialiser la table de plongement des nœuds par une valeur entière (différente de 0 et 1).

Plonger_vecteur : Elle permet de plonger un vecteur ligne de la maille dans l'hypercube croisé.

Plong_gauche : Elle fait appel à la procédure précédente pour plonger les (N/2) premiers vecteurs lignes de la maille , si N est paire, ou ((N/2)+1) premiers vecteurs lignes, si N est impaire dans la partie gauche de l'hypercube croisé.

Plongement des arêtes :

Initial_table_arêtes : Elle permet d'initialiser la table de plongement des arêtes par une valeur entière (différente de 0 et 1).

Nœud_départ_final : Elle permet de créer une table contenant toutes les arêtes de la maille, et les nœuds de l'hypercube croisé correspondants à chaque extrémité initiale et finale.

Cherche_chemin : fait appel à la procédure précédente et cherche le plus court chemin entre l'extrémité initiale et l'extrémité finale de toutes les arêtes dans la table de l'hypercube croisé, et calcule la dilatation de chaque arête.

Interprétation graphique du plongement :

interprétation graphique de la maille :

On utilise la classe **Tcanvas** pour dessiner les ellipses (nœuds) et les arêtes.

-interprétation graphique de l'hypercube croisé :

On utilise la classe **Tcanvas** et **TPoint** pour dessiner les ellipses (nœuds) et les arêtes.

-interprétation graphique de plogement des noeuds :

On utilise la classe **Tcanvas** et **TPoint** et une image pour représenter les nœuds des deux réseaux. Cette image est animée par le composant : **Timer.**

<u>Interprétation graphique du plongement des arêtes One</u>

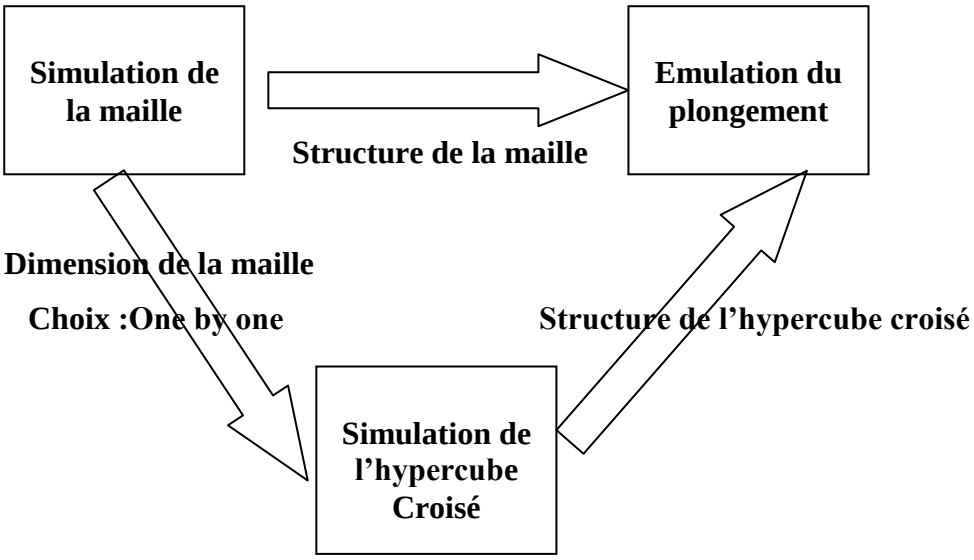
On utilise la classe Tcanvas et Tpoint pour représenter les arêtes élémentaires dans le graphe hôte.

Interprétation graphique du plongement des arêtes Two

On utilise la classe Tcanvas et Tpoint pour représenter les chemins de longueur deux(02) dans le graphe hôte.

Les interactions entre les objets

Organisation des interaction entre objets

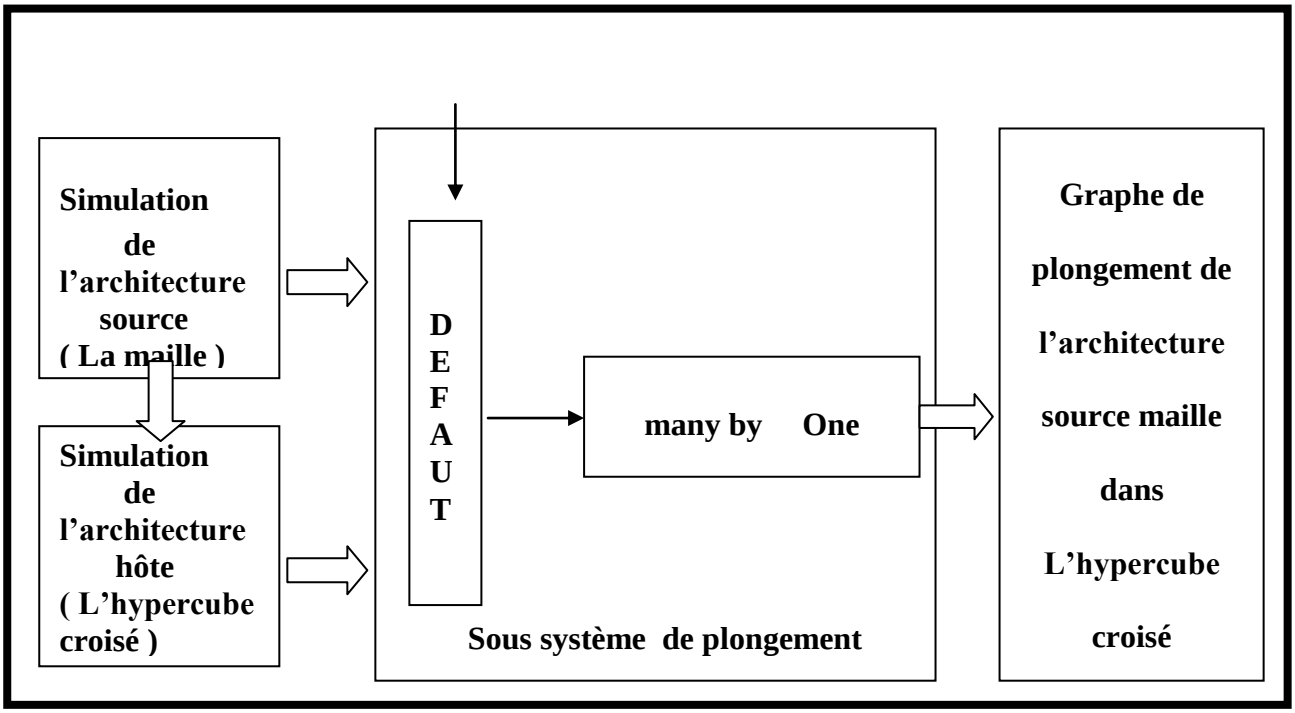


-Le deuxième objet « Simulation de l'hypercube croisé » nécessite une dimension, cette dernière est calculée à partir de N et M du premier objet « Simulation de la maille » et du choix du plongement par excès autrement dit « One by one ».

- Le troisième objet « émulation du plongement » nécessite la table de la maille résultante du premier objet ainsi que la table de l'hypercube croisé résultante du deuxième objet.

2.2-Le sous système de plongement « Many by One »

2.2.1- Architecture du sous système



Il est composé de trois objets :

- La simulation de la maille.
- La simulation de l'hypercube croisé.
- L' émulation du plongement.

2.2.1.1- Simulation de la maille

Cet objet est hérité du premier sous système (plongement One by One).

2.2.1.2-Simulation de l'hypercube croisé

Activité1 : le choix entre :

- la création de l'hypercube croisé correspondant par excès.
- la création de l'hypercube croisé correspondant par défaut.
- dans cette activité on utilise la création de l'hypercube croisé correspondant par défaut .

Activité2 : déjà définie.

La structure de la table : déjà définie.

Les méthodes :

Un test de création de la table de l'architecturesource est généré.Calcul_dimension_H : C'est une fonction qui permet de retourner la dimension par défaut de l'hypercube croisé, calculée à partir de la dimension de la maille.

Initial_table_H : Cette méthode est héritée du premier sous système.

Création_Nœuds_H : Cette méthode est héritée du premier sous système.

Création_Voisin_H : Cette méthode est héritée du premier sous système.

2.2.1.3- Emulation de plongement Many by One

Activité1 : Interprétation structurelle du plongement.

Activité2 : Interprétation graphique du plongement.

Activité3 : Les mesures de qualités du plongement.

2.2.1.4-Interprétation structurelle du plongement

Cette activité est définie dans le premier sous système.

La structure de la table de plongement des nœuds :

C'est une matrice de (N*M) lignes et de 3 colonnes, chaque ligne représente la correspondance entre un ou deux nœuds de la maille et un nœud de l'hypercube croisé et avec son facteur de charge et chaque colonne représente l'adresse binaire du nœud dans les deux architectures.

La structure de la table est la suivante :

Nœuds de la maille	Nœud de l'hypercube croisé	Facteur de charge

La structure de la table de plongement des arêtes :

Déjà définie dans le premier sous système.

Les méthodes :

Plongement des nœuds :

Initial_table_nœuds : cette méthode est héritée du premier sous système.

Plonger_vecteur : Elle est héritée du premier sous système.

Plong_gauche : Elle est héritée du premier sous système.

Plongement des arêtes :

Initial_table_arêtes : Héritée du premier sous système.

Nœud_départ_final : Héritée du premier sous système.

Cherche_chemin : Héritée du premier sous système.

2.2.1.5-Interprétation graphique du plongement

On réalise l'interprétation graphique du plongement par :

- ✓ L'interprétation graphique de la maille ;
- ✓ L'interprétation graphique de l'hypercube croisé ;
- ✓ L'interprétation graphique du plongement des nœuds ;
- ✓ L'interprétation graphique des arêtes de type one ;
- ✓ L'interprétation graphique des arêtes de type two.

2.2.1.6-Interprétation graphique de la maille

On utilise la classe **Tcanvas** pour dessiner les ellipses (nœuds) et les arêtes.

2.2.1.7-Interprétation graphique de l'hypercube croisé:

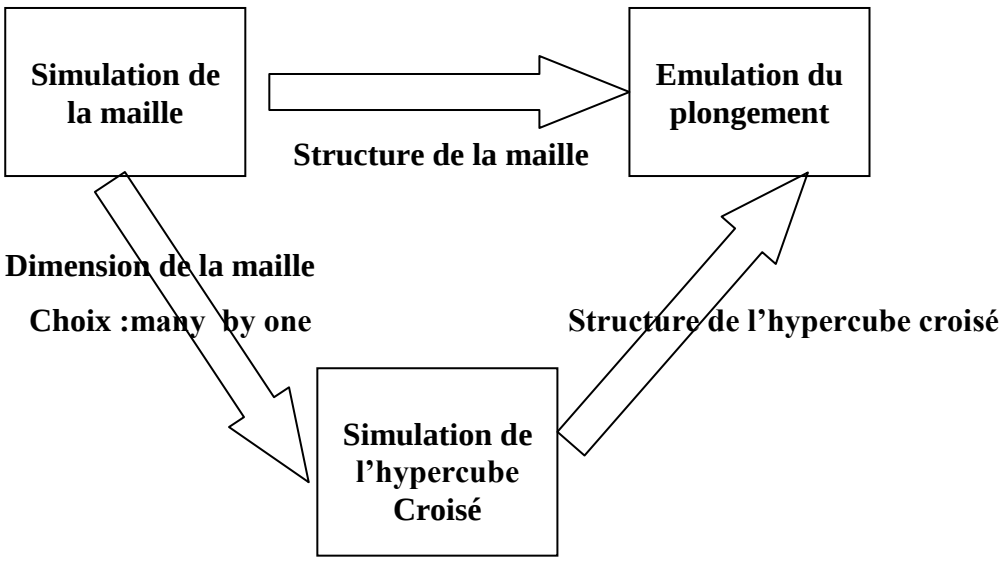
On utilise la classe **Tcanvas** et **TPoint** pour dessiner les ellipses (nœuds) et les arêtes.

2.2.1.8- interprétation graphique de plongement des nœuds

On utilise la classe **Tcanvas** et **TPoint** et une image pour représenter les nœuds des deux réseaux. Cette image est animée par le composant :**Timer.**

Les interactions entre les objets

Organisation des interaction entre objets



-Le deuxième objet « Simulation de l'hypercube croisé » nécessite une dimension, cette dernière est calculée à partir de N et M du premier objet « Simulation de la maille » et du choix du plongement par défaut autrement dit « many by one ».

- Le troisième objet « émulation du plongement » nécessite la table de la maille résultante du premier objet ainsi que la table de l'hypercube croisé résultante du deuxième objet.

3-Conclusion

Les besoins spécifiés par l'étude théorique nous ont incités à concevoir un système composé d'un ensemble de sous systèmes à savoir :

- ✓ Un sous système de simulation du graphe source maille
- ✓ Un sous système de simulation du graphe hôte hypercube croisé
- ✓ Un sous système d'émulation du plongement du graphe source maille dans un graphe hôte hypercube croisé.

Chaque sous système est conçu selon l'approche objet puisqu'elle répond naturellement aux besoins informationnels, fonctionnels et structurels du présent problème. Ce système répond aux besoins fonctionnels de deux façons :

- ✓ la première dite plusieurs pour un (many by one) répondant ainsi à la contrainte du nombre de nœud du graphe hôte (inférieur au du nombre de nœud du graphe source) ;
- ✓ la seconde est celle de un pour un utilisé quand le nombre de nœud et supérieur ou égal au nombre de nœud du graphe hôte.

1- Introduction

La solution proposée à été conçue par l'approche objet, ceci nous à conduit à choisir un langage de programmation adéquat aux de details de cette approche qui est le langage C++ bulder ce dernier ; dispose d'une bibliographie très riche. Nous avons utilisé un environnement physique nécessaire à l'implémentation de la solution composée d'un micro-ordinateur dont les caractéristiques sont les suivantes :

- ✓ Une mémoire à accès aléatoire de 1 giga bytes ;
- ✓ Un CPU dont la fréquence est de 3 giga hertz ;
- ✓ Une carte graphique de 32 méga bytes ;
- ✓ Un bus de fréquence 800 méga hertz ;

2-: La Simulation de la maille »

Cet objet est composé d'un ensemble d'informations binaires représentant la structure de l'architecture maille selon la dimension introduite N*M : N : nombre de lignes , M : nombre de colonnes. Il utilise les méthodes permettant la construction de la maille sur la base des règles prédéfinies.

2.1-La construction de la table de la maille

Structure maille
début
t[14] : entier // *C'est un tableau qui contient l'adresse binaire de chaque nœud dans la maille.*
Fin.
M[128][128] ; // *Matrice de 128 lignes et 128 colonnes.*
Structure Voisin_maille
début
N[14], V1[14], V2[14], V3[14], V4[14] : entier ;
Fin.
MM[16384] ;
// N : *tableau contenant l'adresse binaire de chaque nœud.*
V_i : i = {1, 2}, ou {1. 2. 3}, ou {1. 2. 3. 4} :
C'est un tableau contenant l'adresse binaire du $i^{ième}$ *voisin de chaque nœud.*

2.1.1-Fonction d'analyse

Cette fonction entière permet d'analyser la parité et l'imparité de la dimension du graphe source maille et initialise les tables des nœuds des voisins.

```
Fonction Analyse ( a : entier) : entier
var c, x, t[100], ta[100], e, i, dim : entier ;
    x=a ;
    TQ (d != 1) faire
     Début
          d = x / 2 ;
          e = x mod 2 ;
          t[i] = d;
          ta[i] = e;
          x = d;
          i = i+1;
      FinTQ;
      c = i;  i=0;
      TQ (( ta[i] =0 ) et ( i < c)) faire
          i = i + 1 ;
      FINTQ
      Si (i ≥ c) Alors    dim = c ;   Sinon   dim = c+1 ;FinSi.
      Retourner ( dim ) ;
Fin.
```

2.1.2- Fonction de calcul de la dimension de la maille

Elle permet de calculer la dimension de la maille.

```
Fonction Calcul_dimension( ) : entier  // calcule la dimension de la maille à partir de N
                                                et M.
var h, hh, dim : entier ;
début
    h = Analyse(N) ; // N : nombre de lignes de la maille ;
    hh = Analyse(M) ; // M :nombre de colonnes de la maille;
    dim = h+hh ;
    Retourner ( dim );
Fin.
```

2.1.3- Fonction puissance :

Elle permet de calculer le nombre de nœuds du graphe source.

```
Fonction Puissance ( d : entier) : entier // calcule le nombre des nœuds.
```

var i, x : entier ;

début

x = 1;

pour (i = 1 à d) **faire** x = x*2 ;

Retourner (x) ;

Fin.

2.1.4- Procédure de la création de la maille

Elle permet de construire la structure de la maille carrée. Selon N et M la procédure utilise des générateurs selon la forme carrée.

Procédure création_maille_carrée ( )

Var i, j, x, y, p, sav, s, k, pp, c, save, z, h : entier ;

début

i=0; j=0;

di = Calcul_dimension ( );

p= di-1 ; x = j+1, y = i ;

c = Analyse(b) ;

z = puissance(c) ;

M[i][j].t[p] = 0 ; *// création d'un vecteur par le générateur vertical ;*

M[i+1][j].t[p] = 1 ;

k = 0 ;

TQ (k < z) **faire**

sav = puissance(y) - 1 ;

save = puissance(x) – 1 ;

k = sav + 1 ;

TQ (sav ≥ 0) **faire** // *l'application d'un générateur horizontal* ;

début

s = 0 ;

TQ (s ≤ save) **faire**

début

p=0 ;

TQ (M[s][sav].t[p] = 6) **faire** // *la matrice M est initialisée par 6 ;*

pp= p ;

pour (p = pp à di-1) **faire**

```
                    M[s][k].t[p] = M[s][sav].t[p] ;
                    M[s][sav].t[pp-1] = 0 ; // préfixation par 0 ;
                    M[s][k].t[pp-1] = 1 ; // préfixation par 1 ;
                     s = s + 1 ;
                  finpour ;
                  sav = sav-1;
                  k = k + 1;
            finTQ;
            TQ (( save ≥ 0) et ( s<z))faire
                début
                   h=0 ;
                  TQ(h<sav)faire
                     début
                       P=0 ;
                       TQ(M[save][h].t[p]=6) faire p=p+1 ;
                       pp=p ;
                      Pour (p=pp à di1) faire
                         M[s][h].t[p]=  M[save][h].t[p] ;
                         M[save][h].t[pp-1]=0 ;   / /préfixation par
0.

                         M[s][h].t[pp-1]=1 ;  / /préfixation par 1.
                         h=h+1 ;
                      Finpour;
                      s=s+1;       save=save-1;
                  FinTQ.
                  x=x+1; y=y+1 ;
          Fin TQ ;
      Fin TQ ;
Fin.
```

2.1.5 - Procédure de création de maille horizontale

Elle permet de construire la structure de la maille horizontale. Lorsque le nombre de lignes est inférieur au nombre de colonnes la procédure utilise des générateurs selon la forme rectangulaire horizontale.

Procédure création_maille_horizontale( ) :

```
Var i, j, x, y, cp, cpt , z1,z2, sav, save, v, w, p, pp :entier ;
 début
       i=0 ; j=0 ;
        di=dimension() ;
         p=di-1 ;  x=i+1 ; y=j ;
          cp= Analyse(N) ;   z1=puissance(cp) ;
           cpt= Analyse(M) ;   z2 =puissance(cpt) ;
           M[i][j].t[p] = 0 ;
           M[i+1][j].t[p] = 1 ;
           sav = puissance( y) - 1  ;
           save = puissance(x) – 1 ;
           w = save + 1 ;
          TQ (w <z1)faire
              début
                p=0 ;
               TQ(M[save][sav].t[p]=6) faire p=p+1 ;
               pp= p ;
               TQ(save≥ 0) faire
                     Pour(p=pp à di-1) faire
                        M[w][sav].t[p]=M[save][sav].t[p] ;
                        Préfixation par 0 et par 1 ;
                       w= w+1 ; save=save-1 ;
                     FinPour
                Fin TQ;
                save=w-1;

           Fin TQ
           v=sav+1 ;
          TQ(v < z2) faire
              TQ(sav ≥ 0) faire
                  début
                  p=0 ; save=0 ;
                  TQ(M[save][sav].t[p]=6) faire p=p+1 ;
                  pp=p ;
                  TQ(sav < z1) faire
```

```
            début
                Pour (p=pp à di-1) faire
                    M[save][v].t[p]=M[save][sav].t[p];
                    Préfixation par 0 et  par 1 ;
                    save=save+1 ;
                FinPour
            Fin TQ ;
            v=v+1 ; sav=sav-1 ;
        Fin TQ;
        sav=v-1;
    Fin TQ ;
Fin TQ ;
Fin .
```

2.1.6 - Procédure de création de maille verticale

Elle permet de construire la structure de la maille verticale. Lorsque le nombre de lignes est inférieur au nombre de colonnes la procédure utilise des générateurs selon la forme rectangulaire verticale.

```
Procédure création_maille_verticale( ) :
var i, j, x, y, cp, cpt , z1,z2, sav, save, v, w, p, pp :entier ;
 début
    i=0 ; j=0 ;
    di=dimension() ;
     p=di-1 ;  x=i ; y=j+1 ;
      cp= Analyse(N) ;   z1=puissance(cp) ;
      cpt= Analyse(M) ;   z2 =puissance(cpt) ;
    M[i][j].t[p] = 0 ;
    M[i][j+1].t[p] = 1 ;
    sav = puissance( y) - 1  ;
    save = puissance(x) – 1 ;
    v= save + 1 ;
    TQ (v < z2) faire
        début
          p=0 ;
          TQ(M[save][sav].t[p]=6) faire p=p+1 ;
          pp= p ;
```

```
            TQ(save≥ 0) faire
                Pour(p=pp à di-1) faire
                    M[save][v].t[p]=M[save][sav].t[p] ;
                    Préfixation par 0 et par 1 ;
                     v= v+1 ; sav=sav-1 ;
            Fin TQ;
             sav=v-1;
        Fin TQ ;
         w=save+1 ;
          TQ(w< z1) faire
                TQ(save ≥ 0) faire
                   début
                     p=0 ; sav=0 ;
                    TQ(M[save][sav].t[p]=6) faire p=p+1 ;
                     pp=p ;
                     TQ(sav < z2) faire
                         début
                            Pour (p=pp à di-1) faire
                                M[w][sav].t[p]=M[save][sav].t[p];
                                 Préfixation par 0 et  par 1 ;
                                 sav=sav+1 ;
                    Fin TQ ;
                    w=w+1 ; save=save-1 ;
            Fin TQ;
             save=w-1;
         Fin TQ ;
    Fin TQ ;
Fin .
```

2.1.7 - Procédure voisins maille

Elle permet de construire les voisins de chaque nœud de la maille selon sa dimension et sa forme.

```
Procédure voisin_maille( )
Var  i, j, k, p :entier ;
        k = 0;
```

```
MM[k].N=M[i][j].t ;        //  le  nœud situé dans le coin (0,0)
MM[k].V1=M[i][j+1].t ; //  le voisin droit.
k = k + 1 ;
MM[k].V2=M[i+1][j].t ;  //  le voisin bas.
MM[k].N=M[i][j].t ;        //  le  nœud situé dans le coin (N-1,0)
MM[k].V1=M[i][j+1].t ; //  le voisin droit.
MM[k].V2=M[i-1][j].t ;  //  le voisin haut.
 k = k+1 ;
 MM[k].N=M[i][j].t ;        //  le  nœud situé dans le coin (0,M-1)
 MM[k].V1=M[i][j-1].t ; //  le voisin gauche.
 MM[k].V2=M[i+1][j].t ;  //  le voisin bas.
  k = k+1 ;
 MM[k].N=M[i][j].t ;        //  le  nœud situé dans le coin (N-1,M-1)
 MM[k].V1=M[i][j-1].t ; //  le voisin gauche
 MM[k].V2=M[i-1][j].t ;  //  le voisin haut.
  k = k+1 ;
 MM[k].N=M[i][j].t ;                //  les  nœuds situé dans la première ligne
  MM[k].V1=M[i][j+1].t ; //  le voisin droit.
  MM[k].V2=M[i][j-1].t ; //  le voisin gauche
   MM[k].V3=M[i+1][j].t ;  //  le voisin bas.
  k = k+1 ;
  MM[k].N=M[i][j].t ;                //  les  nœuds situés dans la dernière ligne
 MM[k].V1=M[i][j+1].t ; //  le voisin droit.
 MM[k].V2=M[i][j-1].t ; //  le voisin gauche
  MM[k].V3=M[i-1][j].t ;  //  le voisin haut.
  k = k+1 ;
  MM[k].N=M[i][j].t ;                //  les  nœuds situés dans la première colonne
  MM[k].V1=M[i-1][j].t ; //  le voisin haut.
  MM[k].V2=M[i][j-1].t ; //  le voisin gauche
  MM[k].V3=M[i+1][j].t ;  //  le voisin bas.
   k = k+1 ;
  MM[k].N=M[i][j].t ;                //  les  nœuds situés dans la dernière colonne
  MM[k].V1=M[i][j+1].t ; //  le voisin droit.
   MM[k].V2=M[i-1][j].t ; //  le voisin haut
   MM[k].V3=M[i+1][j].t ;  //  le voisin bas.
```

k = k+1 ;

MM[k].N=M[i][j].t ; // *les nœuds internes*

MM[k].V1=M[i][j+1].t ; // *le voisin droit.*

MM[k].V2=M[i][j-1].t ; // *le voisin gauche*

MM[k].V3=M[i+1][j].t ; // *le voisin bas.*

MM[k].V4=M[i-1][j].t ; // *le voisin haut*

Fin.

3-. Simulation de l'hypercube croisé

Cet objet dispose de la structure de l'hypercube croisé autrement dit les adresses binaires de chaque nœud ainsi que celles de leurs voisins. Il utilise les méthodes de la construction des nœuds, des nœuds voisins et des arêtes les reliant selon les règles architecturales selon le principe de Schiau et Al prédéfinies.

3.1-La construction de la table de l'hypercube croisé

Structure hyper_croisé

début

t[14] : entier // *C'est un tableau qui contient l'adresse binaire de chaque nœud de l'hypercube croisé*

Fin.

HC[16384][15] ; // *Matrice de* 16384 (c.à.d : 2^{14}) *et* 15 *colonnes, contient l'adresse binaire de chaque nœud de l'hypercube croisé avec ses voisins*.

3.1.1- Procédure de création des nœuds de l'hypercube croisé

Elle permet de construire la structure des nœuds de l'hypercube croisé. Selon le critère du type du plongement « many by one » ou « one by one ».

Procédure creation_noeuds( )

Var i, j, sav, l, k, kk: entier;

```
i=0;j=0; k=di-1;
HC[i][j].t[k]=0;  i=i+1;
HC[i][j].t[k]=1;   i=i+ 1;
n = puissance(di) ; // n est le nombre de noeuds de l'hypercube croisé;
TQ ( i < n ) faire
    début
        sav=i-1;
        l=0;
    TQ (l <= sav) faire
```

```
        début
            k=0;
            TQ (HC[l][j].t[k]=3 )  faire   k=k+1;
                kk=k;
                pour (k=0 à di-1 ) faire
                    HC[i][j].t[k]=HC[l][j].t[k];
                    HC[i][j].t[kk-1] = 1;
                    HC[l][j].t[kk-1] = 0;
                FinPour
                l++;
            FinTQ
                    j++;
        FinTQ
Fin.
```

3.1.2- Procédure de création des nœuds voisins de l'hypercube croisé

Elle permet de construire la structure des nœuds voisins de l'hypercube croisé. Selon la dimension définie par le critère du type du plongement « many by one » ou « one by one ».

```
Procédure création_voisin( )
Var  i, j, k, s, w, tb[14], T[14]: entier
début
    i=0;
    TQ (i < n ) faire
        début
            pour (k=0 à di-1)
                tb[k]=HC[i][0].t[k];
                s=d-1;    j=1;
                TQ(s >= (d-2))  faire //La création des deux premiers voisins pour
                                                                chaque  nœud.
                    début
                        Si (tb[s]=0) Alors  tb[s]=1;
                                    Sinon   tb[s]=0;
                        Pour (k=0 à di-1)  faire
                            HC[i][j].t[k]=tb[k];   j++;  s--;
                        Finpour
```

```
                    Pour (k=0 à di-1) faire
                                tb[k]=HC[i][0].t[k];
                    finpour
              finTQ;
             TQ((s >= 0) et (i  mod 2) = 0))  faire  // La création des autres voisins
        des
                                                              nœuds paires
                    début
                        Si (tb[s]=0) Alors   tb[s]=1;
                                    Sinon   tb[s]=0;
                        Pour (k=0 à di-1) faire
                            HC[i][j].t[k]=tb[k];  j++;  s--;
                        Finpour
                    Pour (k=0 à di-1) faire
                        tb[k]=HC[i][0].t[k];
                    Finpour
finTQ;
    s=d-2;
     j=3;
    TQ ((s >= 0 ) et  ((i mod 2) != 0)) faire // La création des autres voisins des
                                                    nœuds impaires.
       début
          Si (tb[s]=0)  Alors tb[s]=1;
              Sinon   tb[s]=0;
          s--;
           pour (k=0 à di-1 ) Faire T[k]=tb[k];
           w=s;
          TQ (w >= 0) faire
                début
                 Si (tb[w]=0) Alors  tb[w]=1;
                            Sinon   tb[w]=0;
                Pour (k=0 à di-1) faire   HC[i][j].t[k]=tb[k];
                j= j+1 ;  w = w-1 ;
                pour (k=0 à di-1) faire  tb[k]=T[k];
           finTQ
```

finTQ

i = i+1 ;

finTQ

Fin.

4- Emulation du plongement »

Cet émulateur est objet composé d'un ensemble d'informations binaires qui sont les adresses des nœuds, des nœuds voisins et des différents liens entre ces nœuds dans les deux structures à savoir :

- ✓ la structure source (maille)
- ✓ La structure hypercube croisé.

Il dispose d'un ensemble de méthodes permettant le plongement de la structure source dans la structure hôte à savoir :

- ✓ Plongement des nœuds ;
- ✓ Plongement des arêtes.

4.1-La construction de la table de plongement des noeuds

Structure table_plong_noeuds

Début

t[15] : entier // *tableau contenant l'adresse binaire des nœuds de la maille et l'adresse binaire des noeuds de l'hypercube croisé.*

Fin

Pl[16384][2] // *c'est une matrice de* 2^{14} *lignes et 2 colonnes, chaque ligne représente un noeud de la maille et le noeud correspondant de l'hypercube croisé.*

4.1.1- Fonction procedure1

Elle permet d'établir la correspondance entre les noeuds de la première ligne de la maille dans la partie la plus à gauche nord gauche de la structure de l'hypercube croisé.

Fonction procédure1(x, i, k : entier) : entier

Structure maille_plongée // *cette matrice représente la position de chaque noeud de la maille plongé dans l'hypercube croisé*

début

t[15] :entier

ma[16384][2] ;

```
Var  j, p;
     j=0;
     pour (p=0 à di-1) faire  ma[i][j].t[p]=M[x][k].t[p]; k++; // le plongement du premier
                                                    nœud du premier vecteur ligne de la maille.
     Si (k < M) Alors  // s'il y a encore des nœuds dans cette ligne.
          Pour (p=0 à di-1)  faire  ma[i+1][j].t[p]=M[x][k].t[p];
          k=k+1;  //  passer au prochain noeud.
     FinSi ;
     Si(k < M) Alors
          Pour (p=0 à di-1) faire  ma[i+3][j].t[p]=M[x][k].t[p];
          k++;
     finSi;

     Si (k < M) Alors
          Pour (p=0 à di-1) faire  ma[i+2][j].t[p]=M[x][k].t[p];
     FinSi ;
     k++;
     Retourner (k);
Fin.
```

4.1.2- Fonction procedure2

Elle permet d'établir la correspondance entre les noeuds de la première colonne de la maille dans la partie la plus à gauche nord droite de la structure de l'hypercube croisé.

```
Fonction  procédure2 ( x, i, k: entier) : entier
début
 var  j, p;
   j=0;
   pour (p=0 à di-1) faire  ma[i][j].t[p]=M[x][k].t[p]; k++;
   Si (k < M) Alors  // s'il y a encore des nœuds dans cette ligne.
        Pour (p=0 à di-1)  faire  ma[i+1][j].t[p]=M[x][k].t[p];
        k=k+1;  //  passer au prochain noeud.
   FinSi ;
   Si(k < M) Alors
```

```
        Pour (p=0 à di-1) faire  ma[i-1][j].t[p]=M[x][k].t[p];
        k++;
    finSi;
    Si (k < M) Alors
        Pour (p=0 à di-1) faire  ma[i-2][j].t[p]=M[x][k].t[p];
    FinSi ;
    k++;
    Retourner (k);
Fin.
```

4.1.3-Procédure de plongement des vecteurs lignes

Elle fait appel aux deux fonctions précédentes pour plonger un vecteur ligne dans la partie gauche de l'hypercube croisé, où x est un entier représente le numéro de situation d'un vecteur dans la maille, et i est un entier qui représente l'emplacement du premier nœud d'un vecteur ligne dans l'hypercube croisé.

```
Procédure plonger_vecteur_ligne_g (x, i : entier)
Var k, ii, emp : entier ;
    ii = i ;
    kk= procédure1(x, ii, k);
    TQ (kk < M) faire   // tant qu'il y a des nœuds dans le vecteur ligne.
        début
            ii = ii+ emp ;
            kk = procédure2(x, ii, kk);
      finTQ;
Fin.
```

4.1.4-Procédure de plongement la partie haute de la maille

Elle permet de plonger la moitié des vecteurs lignes (de la partie haute de la maille) dans la partie gauche de l'hypercube croisé.

```
Procédure plong_gauche ( )
Var f, ff, i, emp_v, moit : entier ;// moit est le nombre de vecteurs lignes qui doivent être
                                    plongés  dans la partie gauche de l'hypercube croisé.
```

```
                                    // emp_v est une variable indiquant l'emplacement d'un
                                       vecteur ligne dans la maille ;
        Si (N mod 2 =0) Alors    moit = N/2 ;
        Sinon   moit = (N-1)/2 ;  moit = moit + 1
        finSi ;
        ff = 0 ; x = 0 ; i = 0 ;
        plonger_vecteur_ligne_d(x, i);
        x = x+ 1 ; // passer au prochain vecteur ligne ;
        si (x < moit) Alors
                f = Analyse(M) ;
                ff = puissance(f); i = ff;
                plonger_vecteur_ligne_d(x, i) ;
       finSi ;
        x =x+emp_v ;  i = i+ff ;
       TQ (x < moit) faire
              plonger_vecteur_ligne_d(x, i) ;
              x =x+emp_v ;  i = i+ff ;
        finTQ;
   Fin.
```

4.1.5- Fonction procedure3

Elle permet d'établir la correspondance entre les noeuds à partir de la dernière ligne de la maille dans la partie la plus à droite nord gauche de la structure de l'hypercube croisé.

```
Fonction  procédure3( x, i, k : entier) : entier
   Var  j, p;
        j=1;
       pour (p=0 à di-1) faire  ma[i][j].t[p]=M[x][k].t[p]; k++;
      Si (k < M) Alors .
          Pour (p=0 à di-1)  faire  ma[i+1][j].t[p]=M[x][k].t[p];
           k=k+1;  //  passer au prochain noeud.
     FinSi ;
     Si(k < M) Alors
          Pour (p=0 à di-1) faire  ma[i+2][j].t[p]=M[x][k].t[p];
```

```
        k=k+1;
    finSi;
    Si (k < M) Alors
        Pour (p=0 à di-1) faire  ma[i+3][j].t[p]=M[x][k].t[p];
    FinSi ;
    k=k+1;
    Retourner (k);
Fin.
```

4.1.6- Fonction procedure4

Elle permet d'établir la correspondance entre les noeuds à partir de la dernière ligne de la maille dans la partie la plus à droite sud gauche de la structure de l'hypercube croisé.

```
Fonction  procédure4( x, i, k : entier) : entier
  Var  j, p;
    j=1;
    pour (p=0 à di-1) faire  ma[i][j].t[p]=M[x][k].t[p]; k++;
    Si (k < M) Alors
        Pour (p=0 à di-1)  faire  ma[i+1][j].t[p]=M[x][k].t[p];
        k=k+1;  //  passer au prochain noeud.
    FinSi ;
    Si(k < M) Alors
        Pour (p=0 à di-1) faire  ma[i-2][j].t[p]=M[x][k].t[p];
        k=k+1;
    finSi;
    Si (k < M) Alors
        Pour (p=0 à di-1) faire  ma[i-1][j].t[p]=M[x][k].t[p];
    FinSi ;
    k=k+1;
    Retourner (k);
Fin.
```

4.1.7- Procédure de plongement des vecteurs lignes dans la partie droite de l'hypercube croisé

Elle fait appel aux deux fonctions précédentes pour plonger un vecteur ligne dans la partie droite de l'hypercube croisé, où x est un entier représente le numéro de situation d'un vecteur dans la maille, et i est un entier qui représente l'emplacement du premier nœud d'un vecteur ligne dans l'hypercube croisé.

```
Procédure plonger_vecteur_ligne_d (x, i : entier)
Var k, ii, emp : entier ;
    ii = i ;
     kk= procédure3(x, ii, k);
     TQ (kk < M) faire   // tant qu'il y a des nœuds dans le vecteur ligne.
         début
            ii = ii+ emp ;
            kk = procédure4(x, ii, kk);
     finTQ;
Fin.
```

4.1.8- Procédure de plongement des vecteurs de la base de la maille de l'hypercube croisé

Elle permet de plonger la moitié des vecteurs lignes (de la partie basse de la maille) dans la partie droite de l'hypercube croisé.

```
Procédure plong_droite ( )
Var f, ff, i, emp_v, moit : entier ;// moit est le nombre de vecteurs lignes qui doivent être
                                       plongés  dans la partie droite de l'hypercube croisé.
                                    // emp_v est une variable indiquant l'emplacement d'un
                                       vecteur ligne dans la maille ;
    Si (N mod 2 =0) Alors    moit = N/2 ;
     Sinon   moit = (N-1)/2 ; moit = moit + 1
     finSi ;
     ff = 0 ; x = N-1; i = 0 ;
      plonger_vecteur_ligne_d(x, i);
      x = x - 1 ;
```

```
Si (x < moit) Alors
        f = Analyse(M) ;
        ff = puissance(f);  i = ff;
        plonger_vecteur_ligne_d(x, i) ;
  finSi ;
  x =x + emp_v ;  i = i + ff ;
  TQ (x < moit) faire
     plonger_vecteur_ligne_d (x, i) ;
     x =x + emp_v ;  i = i + ff ;
finTQ;
Fin.
```

5-Interprétation graphique du système

Pour que la simulation du plongement de la maille dans l'hypercube croisé soit plus lisible et apparente, nous avons interprété graphiquement les structures de la maille et de l'hypercube croisé par un ensemble de méthodes utilisant des fonctions graphiques.

5.1- interprétation graphique de la maille

```
procédure dessiner_graphe_maille ( ) // cette procédure permet de dessiner : le
                                        graphe de la maille.
    début
 var  i , j ,p, dx,dy, xo,yo:entier;
        R: entier //  C'est le rayon de l'ellipse représentant un nœud de la maille.
        AnsiString s,s2;
    PaintBox1->Canvas->FillRect(PaintBox1->ClientRect);
pour (i=0 à  bb) faire          s="";
    PaintBox1->Canvas->Ellipse(xo, yo + dy * i, xo + R, yo + R + dy * i);
    PaintBox1->Canvas->MoveTo(xo + R / 2, yo + dy * i + R / 2);
    PaintBox1->Canvas->LineTo(xo + R / 2 + (b - 1) * dx, yo + dy * i + R / 2);
    Fin pour;
   pour (j=0 à b) faire          s="";
   PaintBox1->Canvas->LineTo(xo + dx * j + R / 2, yo + dy * (bb - 1) + R / 2)
   pour (j=0 à b) faire   s2.printf("%d ",M[i][j].t[p]);   s=s+s2;   fin pour
     PaintBox1->Canvas->Brush->Style = bsSolid;
     PaintBox1->Canvas->Ellipse(xo + dx * j, yo + dy * i,xo + R + dx * j, yo + R +
     dy * i);  // pour dessiner un ellipse
```

PaintBox1->Canvas->Brush->Style = bsClear;

PaintBox1->Canvas->TextOutA(((xo + dx * j)-15), yo + dy * i - R / 2, s);

// **TextOut** *est utilisée pour écrire les étiquettes des nœuds de la maille*

Fin pour

Fin.

5.2- interprétation graphique de l'hypercube croisé

Procédure dessiner_graphe_hyper ( ) // *cette procédure permet de dessiner : le graphe de l'hypercube croisé*

début

var i , j ,p, dx,dy, xo,yo, a, k, pp:entier;

pour (int i = 0 à n)

pour (int j = 1 à d+1)

k = bin2int(i, j); //la conversion d'un entier en binaire .

si ((j == 1) | (j == 2))

dessiner_ligne(i, k); *// C'est une procédure permettant de dessiner une arête droite de l'hypercube croisé en utilisant la classe Tcanvas.*

sinon

dessiner_arc(i, k);); *// C'est une procédure permettant de dessiner un arc de l'hypercube croisé en utilisant la classe Tcanvas.*

fin pour

fin pour

Fin.

5.3- interprétation graphique du plongement

procédure get_position // *permet à un nœud de la maille de trouver son emplacement dans le réseau de destination (l'hypercube croisé) en utilisant la classe Tpoint.*

Procédure Graph_par_Defaut ( )

d = puissance(di1); // di1 c'est la dimension par défaut.

Hypercube1->definir_hyperc(di1);

Form1->definir_maille();

AnsiString s, s1, s2, s3;

TPoint p11, p12, p2;

s1 = "";

p11 = getpos(maille_s, s1, b * bb, 1, 1);

p12 = TPoint(0, 0);

```
    PaintBox2->Canvas->Brush->Color = clRed;
    PaintBox1->Canvas->Ellipse(p11.x, p11.y, p11.x + 21, p11.y + 21);
  PaintBox1->Canvas->Ellipse(p12.x, p12.y, p12.x + 21, p12.y + 21);
  PaintBox2->Canvas->Ellipse(p2.x - 6, p2.y - 6, p2.x + 6, p2.y + 6);
  PaintBox1->Canvas->Brush->Color = clWhite;
```

Fin.

Procédure Graph_par_excès ( )

début

```
   Hypercube1->definir_hyperc(di);
   d = puissance(di);
   Form1->definir_maille();
   AnsiString s, s1, s2;
   TPoint p1, p2;
   s1 = "";
   p1 = getpos(maille_s, s1, b * bb, 1, 1);
   p2 = getpos(hyperc_d, s2, d, 2, 1, s1);
   PaintBox2->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->Ellipse(p1.x, p1.y, p1.x + 21, p1.y + 21);
   PaintBox2->Canvas->Ellipse(p2.x - 6, p2.y - 6, p2.x + 6, p2.y + 6);
   PaintBox1->Canvas->Brush->Color = clWhite;

    Image1->Left = PaintBox1->Left + p1.x;
    Image1->Top = PaintBox1->Top + p1.y;
    Adresse1->Left = Image1->Left;
   Adresse1->Top = Image1->Top + Image1->Height;
```

Fin.

procédure deplacer_proc(TPoint ps, TPoint pa, AnsiString adresse)

// cette procédure permet de déplacer chaque nœud de la maille vers son emplacement dans l'hypercube croisé, elle utilise la procédure suivante.

procédure Timer1Timer(TObject *Sender)

Timer1->Enabled = false; // **Timer** : *un composant permettant l'animation des images, des formes et d'autres composants de manière synchronisée.*

si (Image1->Left <= ScrollBox1->Width) *//Déplacement Horizontal*

```
  Image1->Left += 20;
```

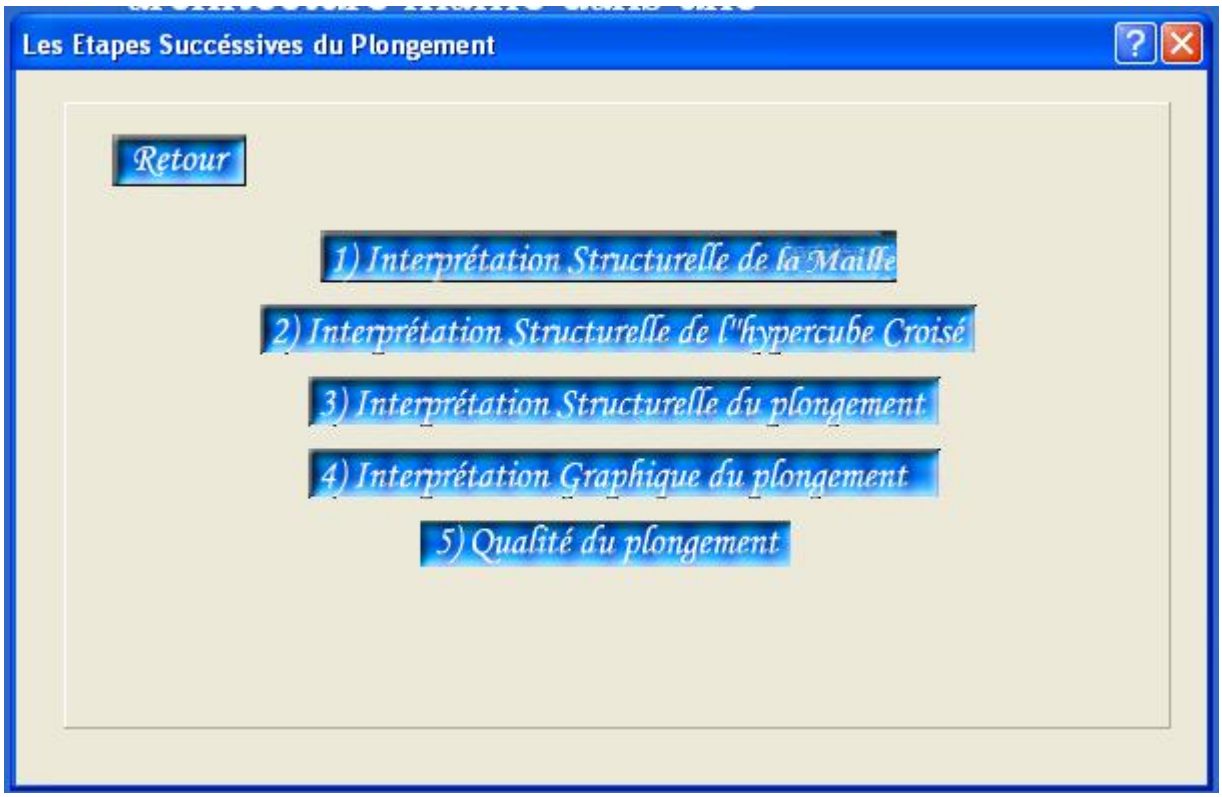
```
    Adresse1->Left = Image1->Left;
  Sinon                                   //Déplacement Vertical
        Image2->Top += 20;
        Adresse2->Top = Image2->Top + Image2->Height;
Fin.
```

6-Tests et résultats

6.1- Test sur le plongement many by one

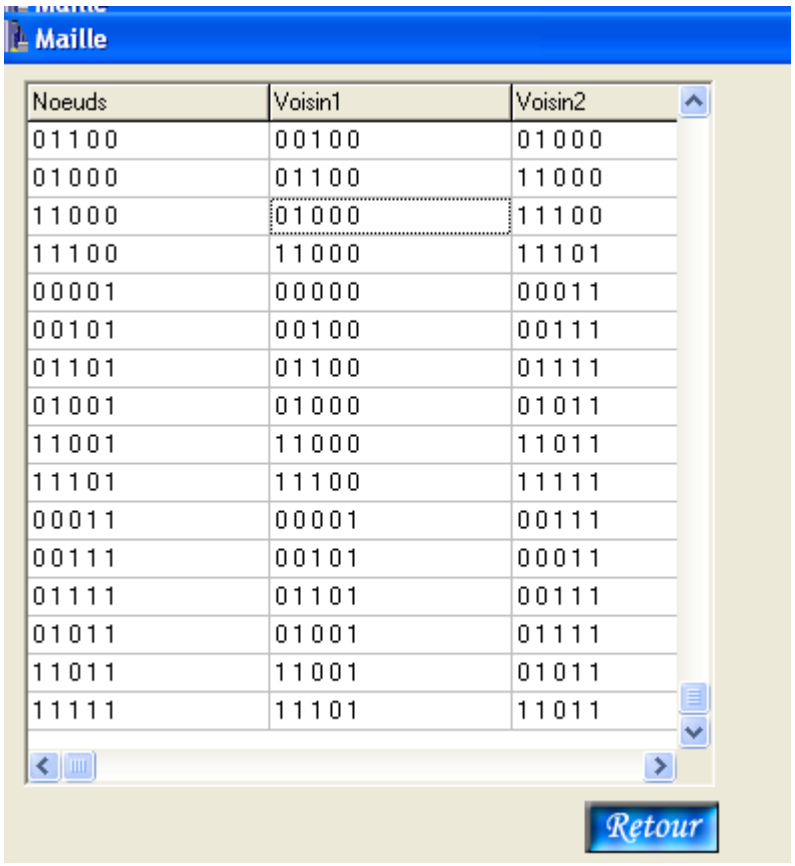
Cette fiche permet d'introduire la dimension de la maille en nombre de et en nombre de colonne. Elle permet aussi de lancer les étapes du plongement en deux modes à savoir:

1. le mode pas a pas.
2. le mode direct.

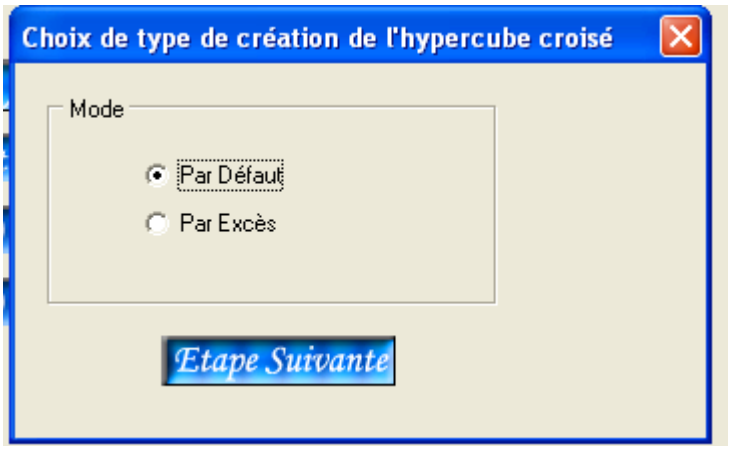


La fiche étapes successives du plongement contient un menu composé d'une succession d'activités :

1. Interprétation structurelle de la maille
2. Interprétation structurelle de l'hypercube croisé.
3. Interprétation structurelle du plongement.
4. Interprétation graphique du plongement.
5. La qualité du plongement.

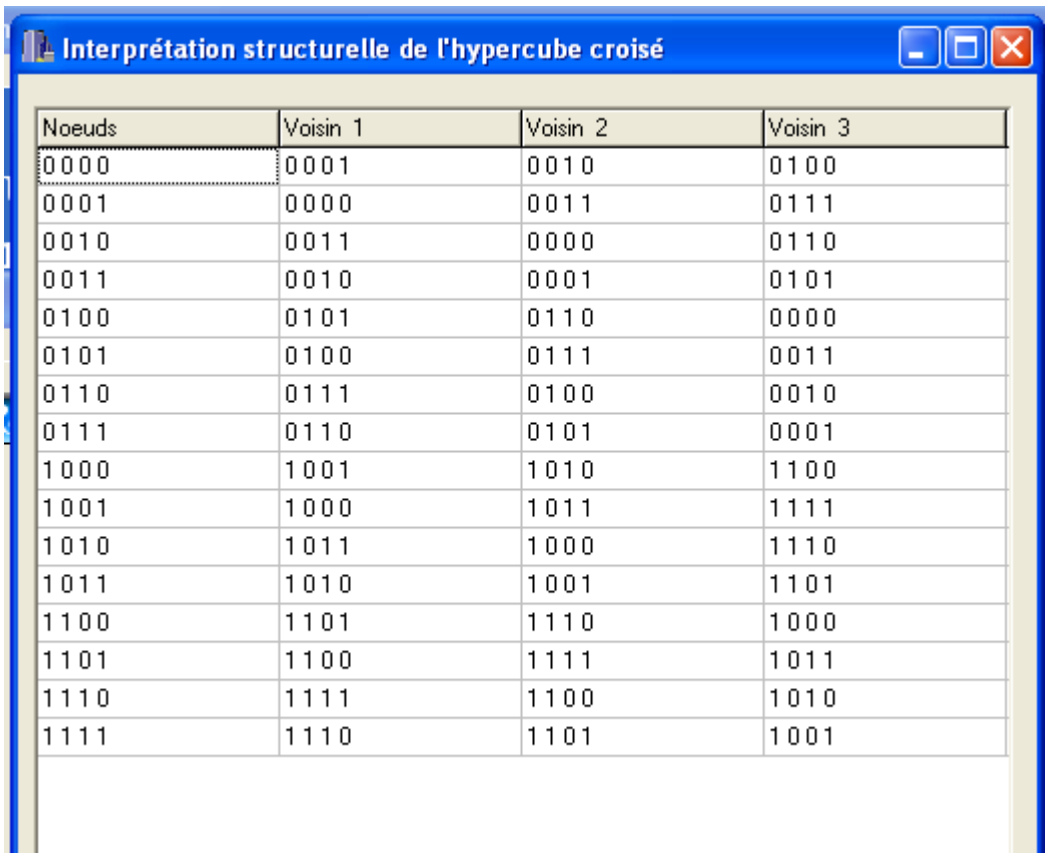


Après avoir choisi l'interprétation structurelle de la maille. Cette fiche fait apparaître le résultat de l'exercice de l'activité principale de la simulation de la maille autrement dit les noeuds et les voisins des nœuds selon les règles architecturaux de la maille.

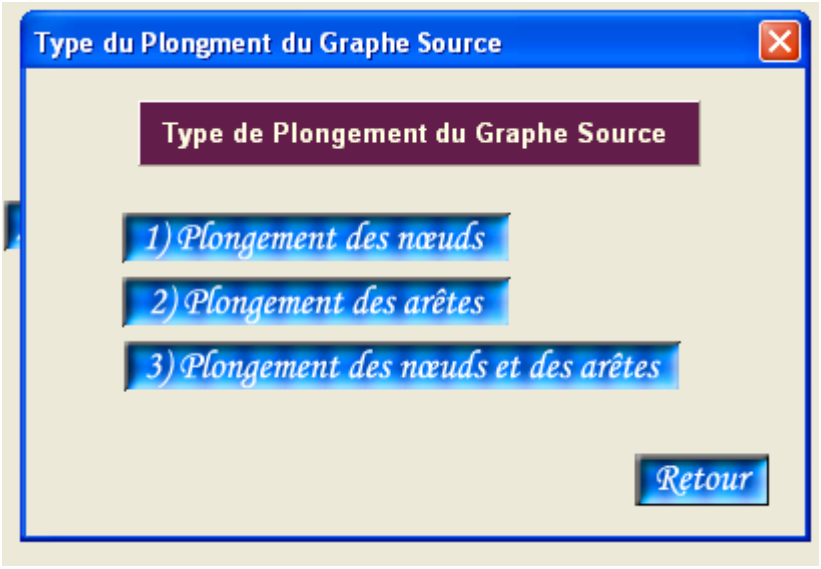


Un choix est fait sur le type de plongement a savoir:

1. par défaut, elle exprime le plongement many by one.
2. par excès elle exprime le plongement one by one.

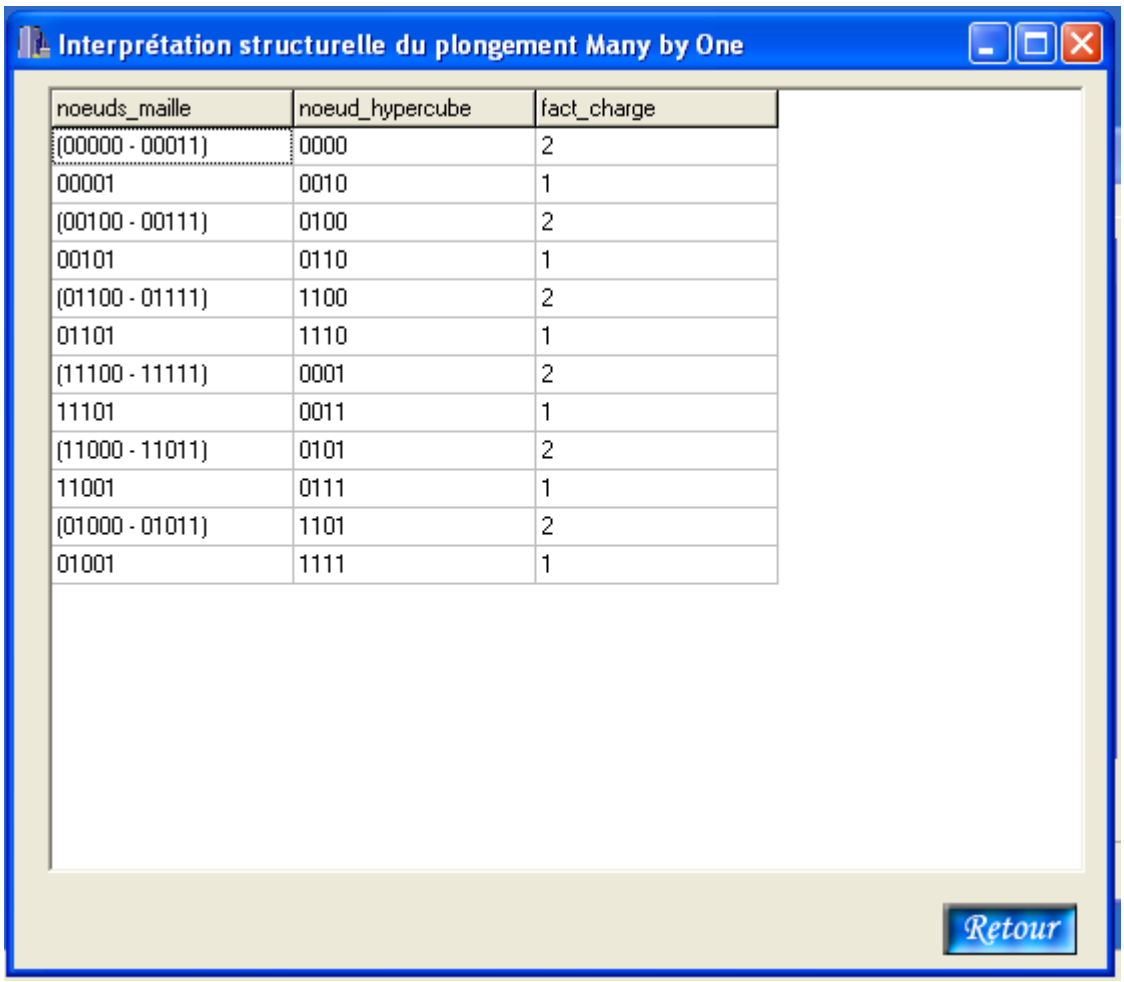


Après avoir sélectionné par défaut, autrement dit un plongement many by one. L'activité de construction des de l'hypercube croisé est exercé. Les résultats apparaîtront dans cette fiche. Ces résultats sont les nœuds et leurs voisins i tel que i est la dimension.



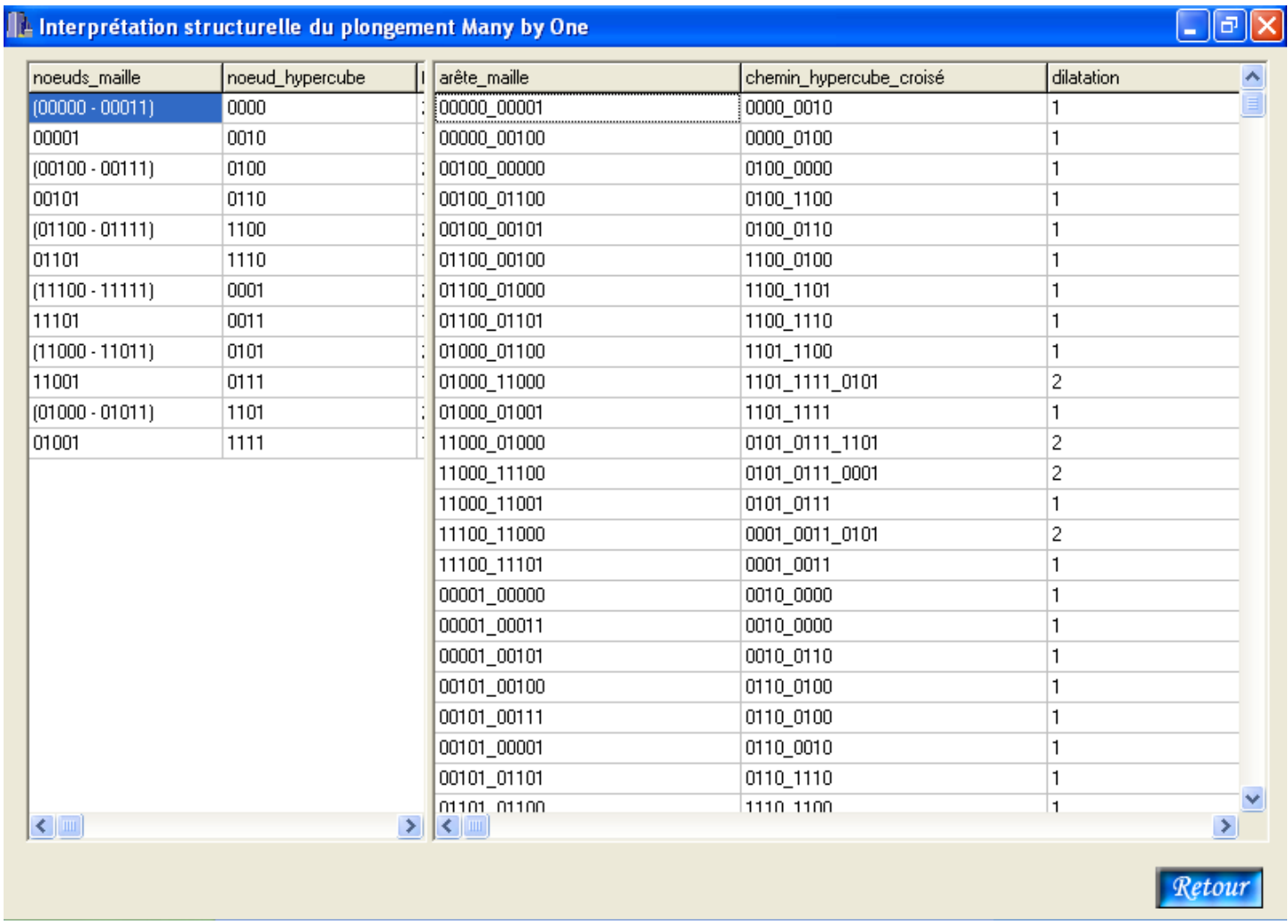
Une fois les structures de la maille et celle de l'hypercube sont construites autrement dit simulés. On active l'émulateur de plongement qui contient les activités suivantes :

1. Le plongement des nœuds.
2. Le plongement des arêtes.
3. le plongement des nœuds et des arêtes.



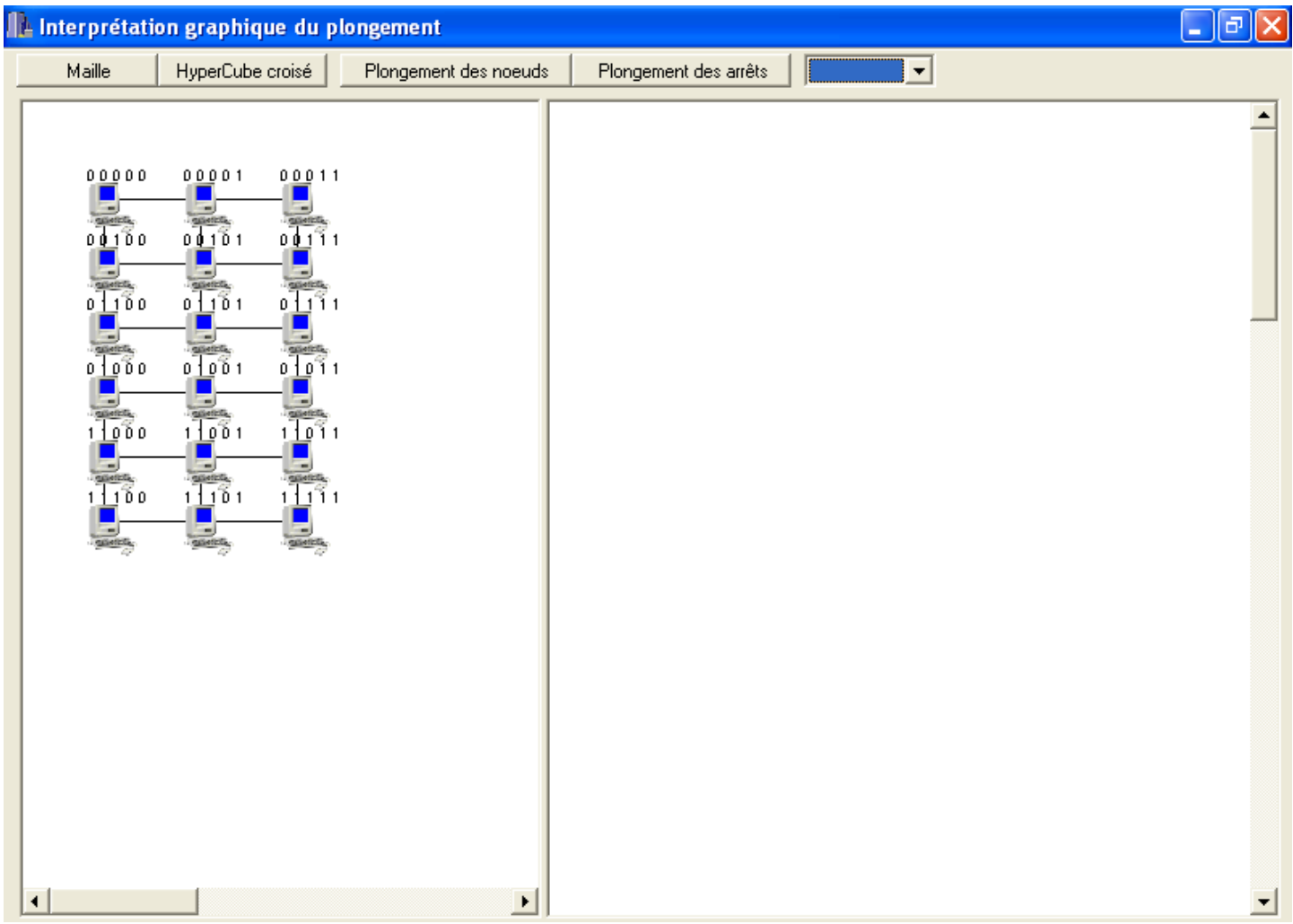
Après avoir sélectionné l'activité plongement des nœuds. Cette fiche apparaît et affiche les résultats du plongement des nœuds ainsi que leur facteur de charge par exemple les nœuds étiquetés par 00000,00011 correspondent à un seul nœud étiqueté 0000 dans l'hypercube avec un facteur de charge égal a deux (2), par contre le nœud 00001 correspond à 0010 dans l'hypercube croisé avec un facteur de charge égal à un

Après avoir sélectionné le plongement des arêtes cet fiche apparaît et montre les résultats autrement dit une arête de la maille peut correspondre soit à une arête élémentaire ou à un chemin dans l'hypercube croisé. Par exemple l'arête 00000-00001 correspond à 0000-0010 et l'arête 01000-11000 correspond à un chemin 1101-1111-0101. Cette fiche fait apparaître aussi la dilatation.



Cette fiche montre quand l'activité plongement des nœuds et des arêtes est activée le résultat du plongement à savoir :

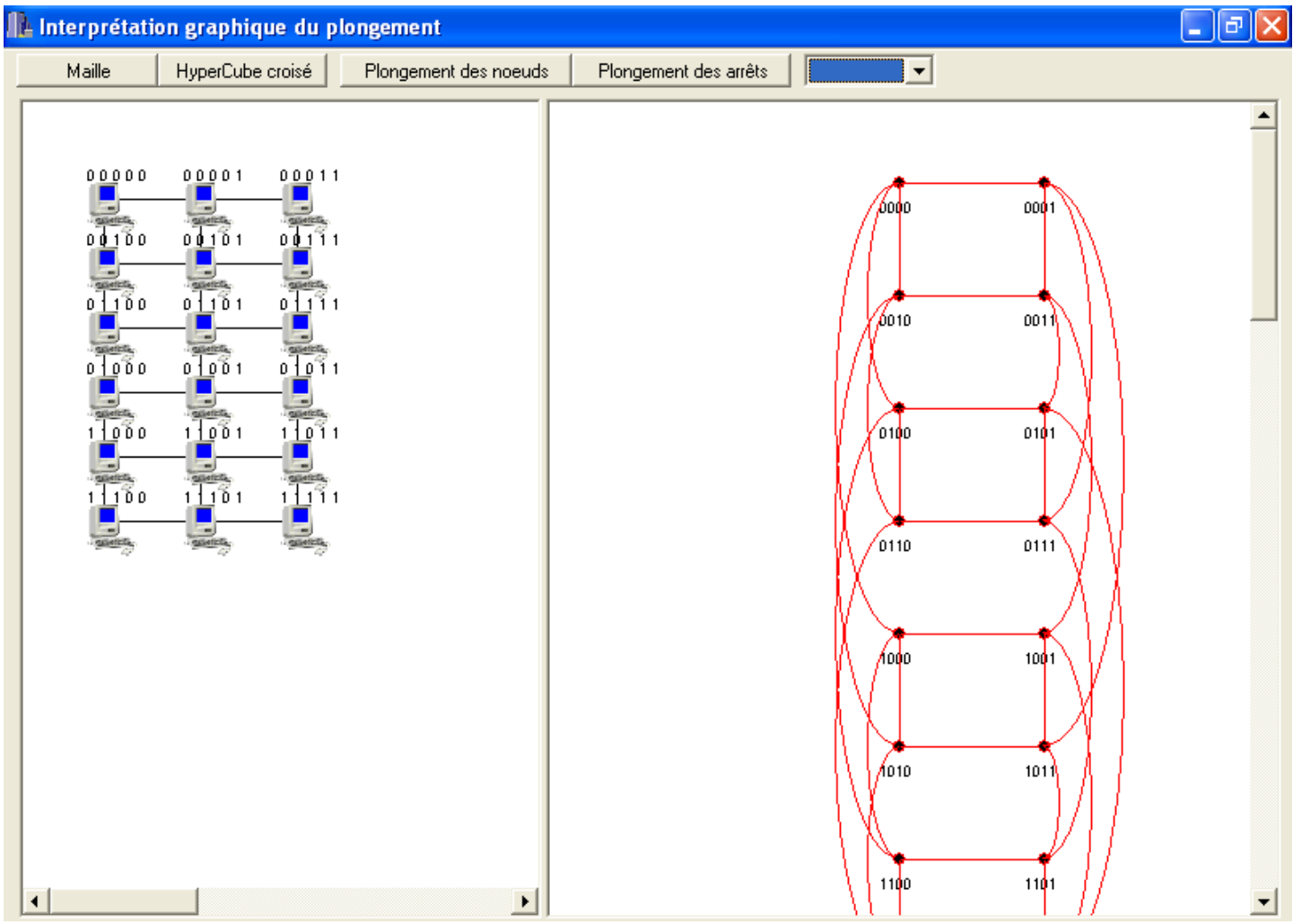
1. la correspondance entre les nœuds de la maille et celles des nœuds de l'hypercube croisé.
2. la correspondance entre les arêtes de la maille est celles de l'hypercube croisé qui peuvent êtres soit des arêtes élémentaires ou des chemins.



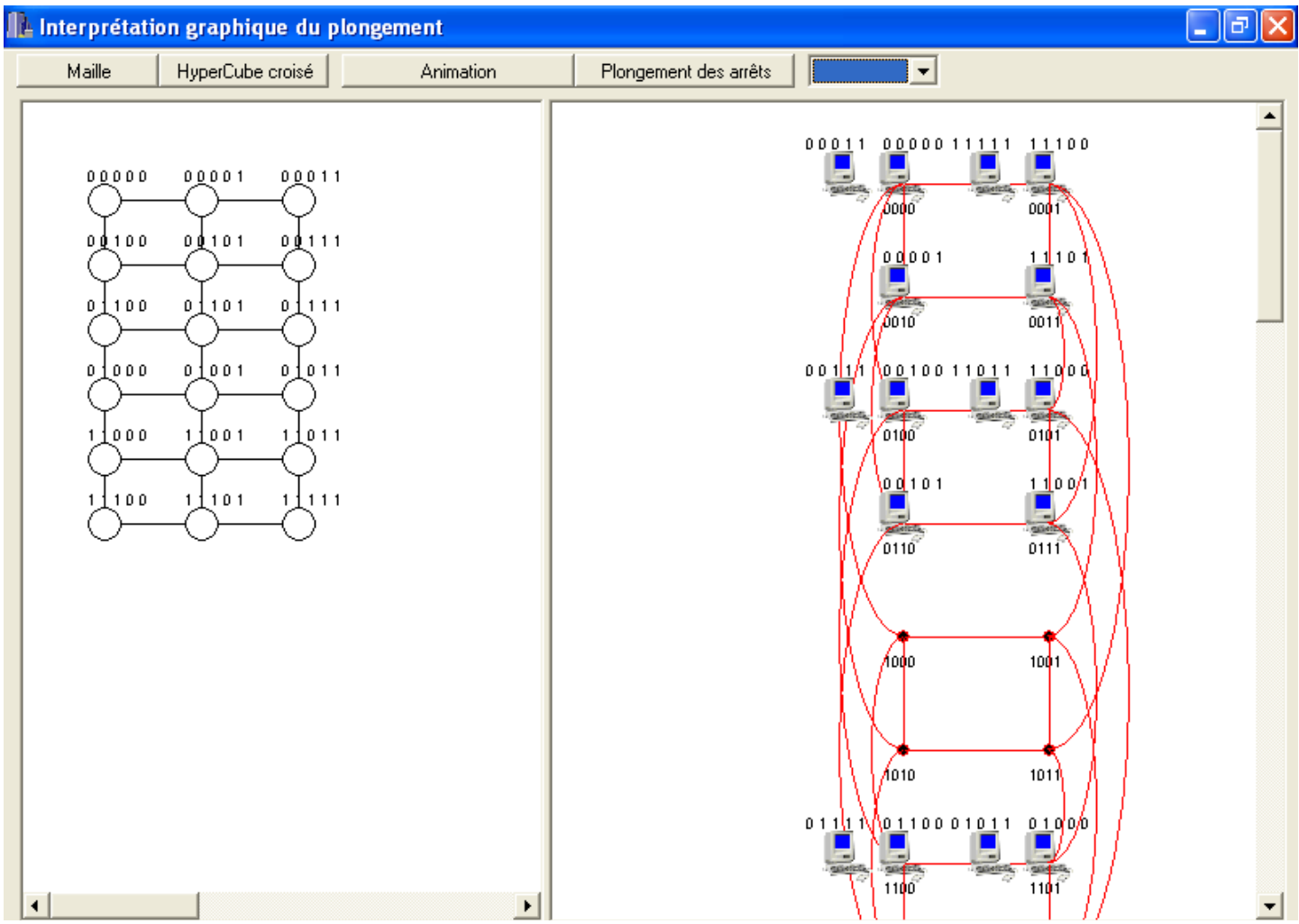
Cette fiche contient les différentes étapes de la simulation du plongement de la maille dans l'hypercube croisé d'une façon graphique à savoir :

1. la maille.
2. l'hypercube croisé.
3. plongement des nœuds.
4. plongement des arêtes.

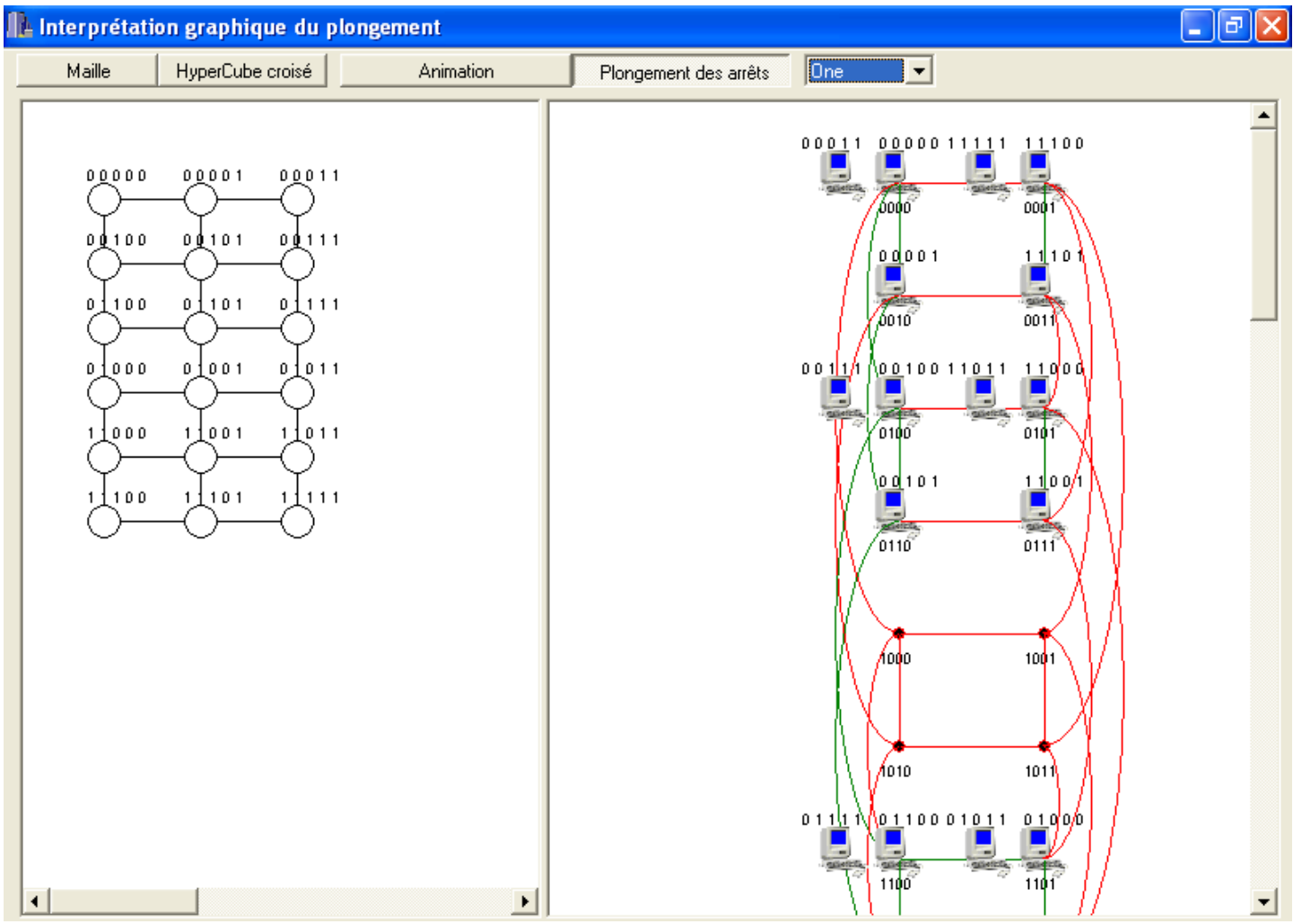
Après avoir séléctionné la maille. Un graphe interprétant la maille par exemple de dimension 6*3 apparaît.



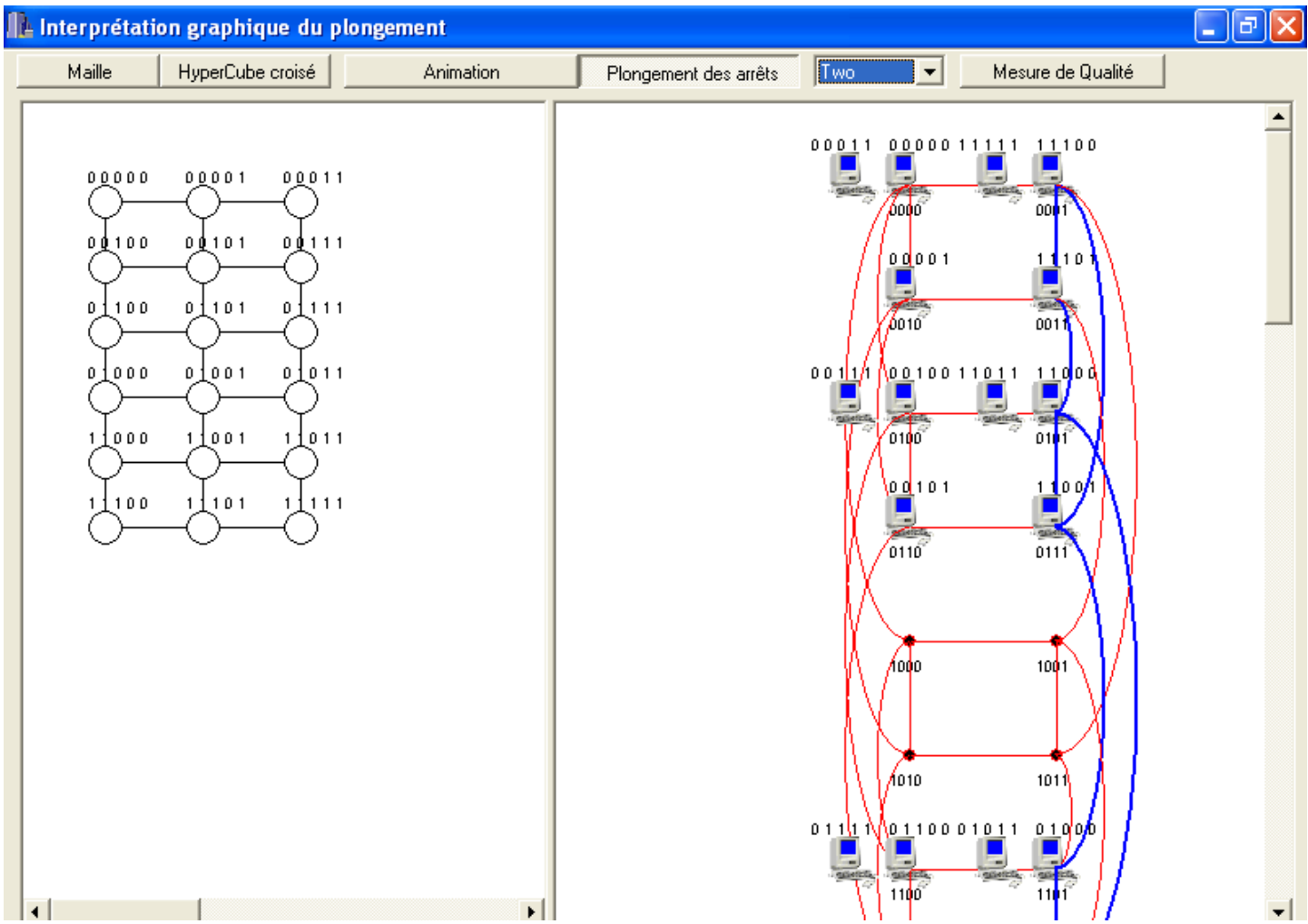
cette fichez montre l'interprétation graphique de la maille et celle de l'hypercube croisé simulées.



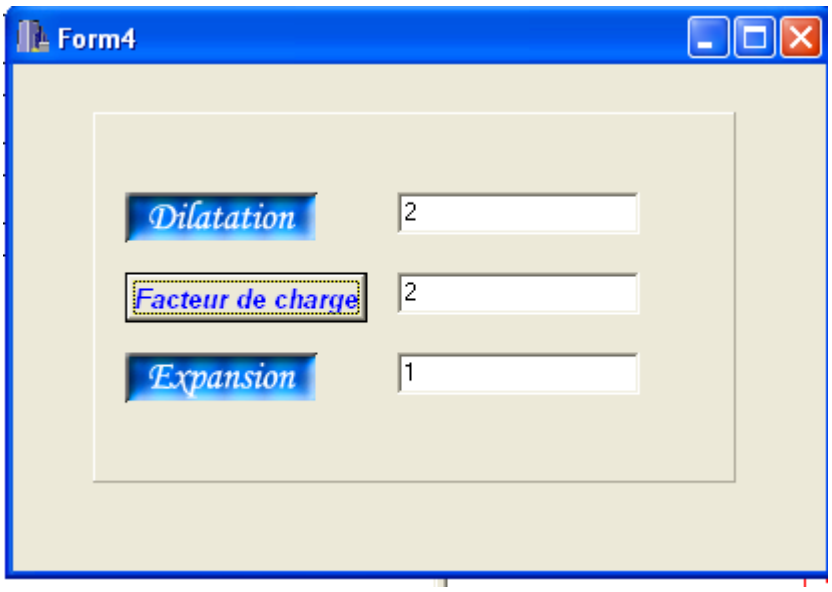
Cette fiche montre le plongement des nœuds de la maille dans l'hypercube croisé dune façon graphique.



Cette fiche montre le plongement many by one des arêtes de longueur égale à un de la maille dans l'hypercube croisé dune façon graphique en premier lieu. (Coloration en vert à gauche)

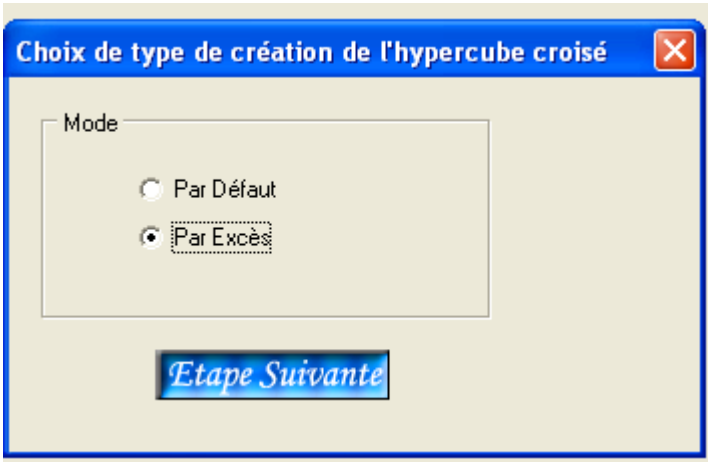


Cette fiche montre le plongement many by one des arêtes de longueur égale à un de la maille à des chemins de longueur égale à deux dans l'hypercube croisé dune façon graphique en premier lieu. (Coloration en bleu à droite)

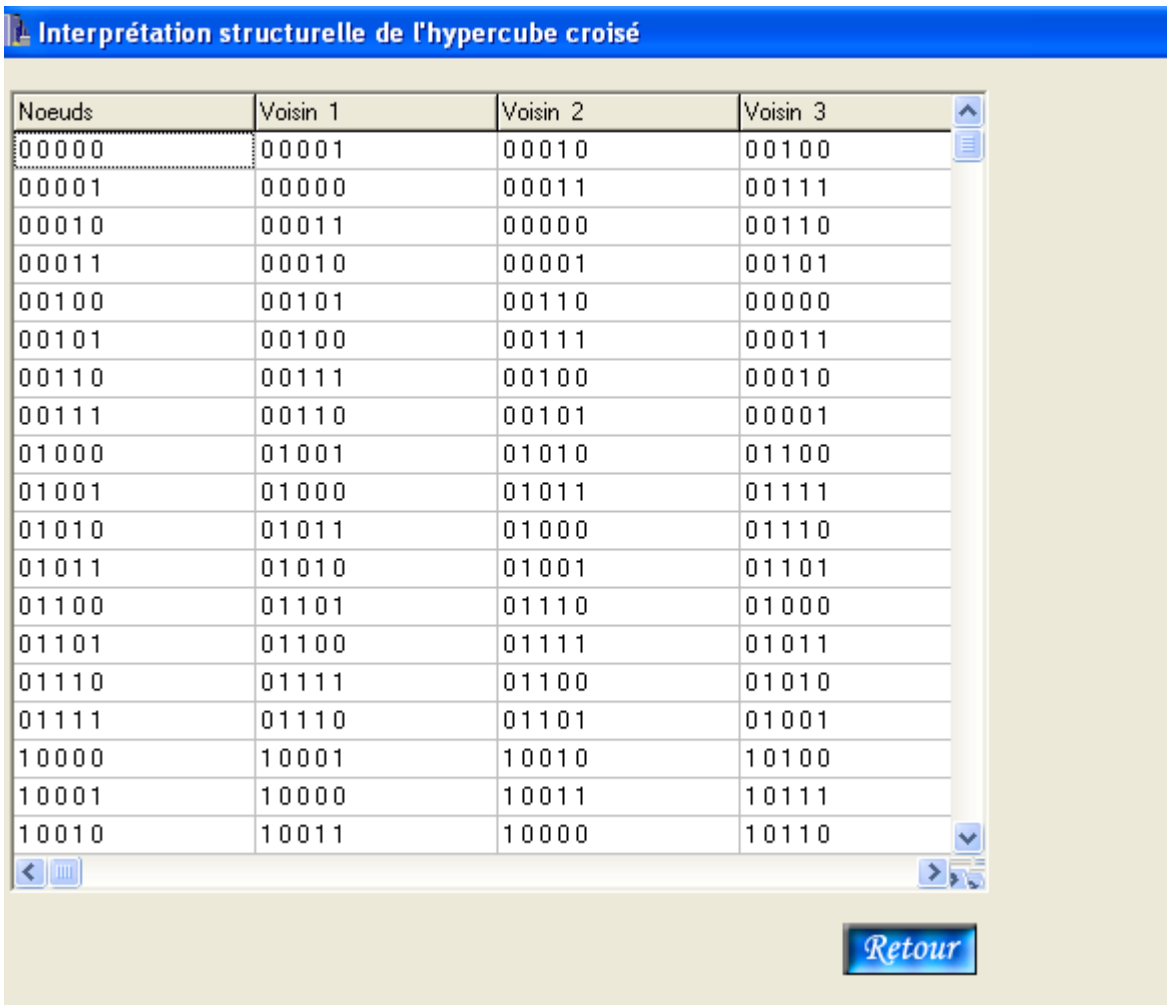


cette fiche apparaît quand on séléctionne la mesure de qualité du plongement. Elle fait apparaître les valeur de la dilatatio,l'expansion et le facteur de charge.

6.2-Test sur le plongement One by One



A prés avoir simuler la structure du graphe source. On sélectionne les plongement one by one en utilisant cette fiche avec un choix par excès



Les résultats de la simulation de l'hypercube croisé de dimension égale à 5 autrement par excès sur l'exemple choisi apparaissent dans cette fiche.

Interprétation structurelle du plongeme

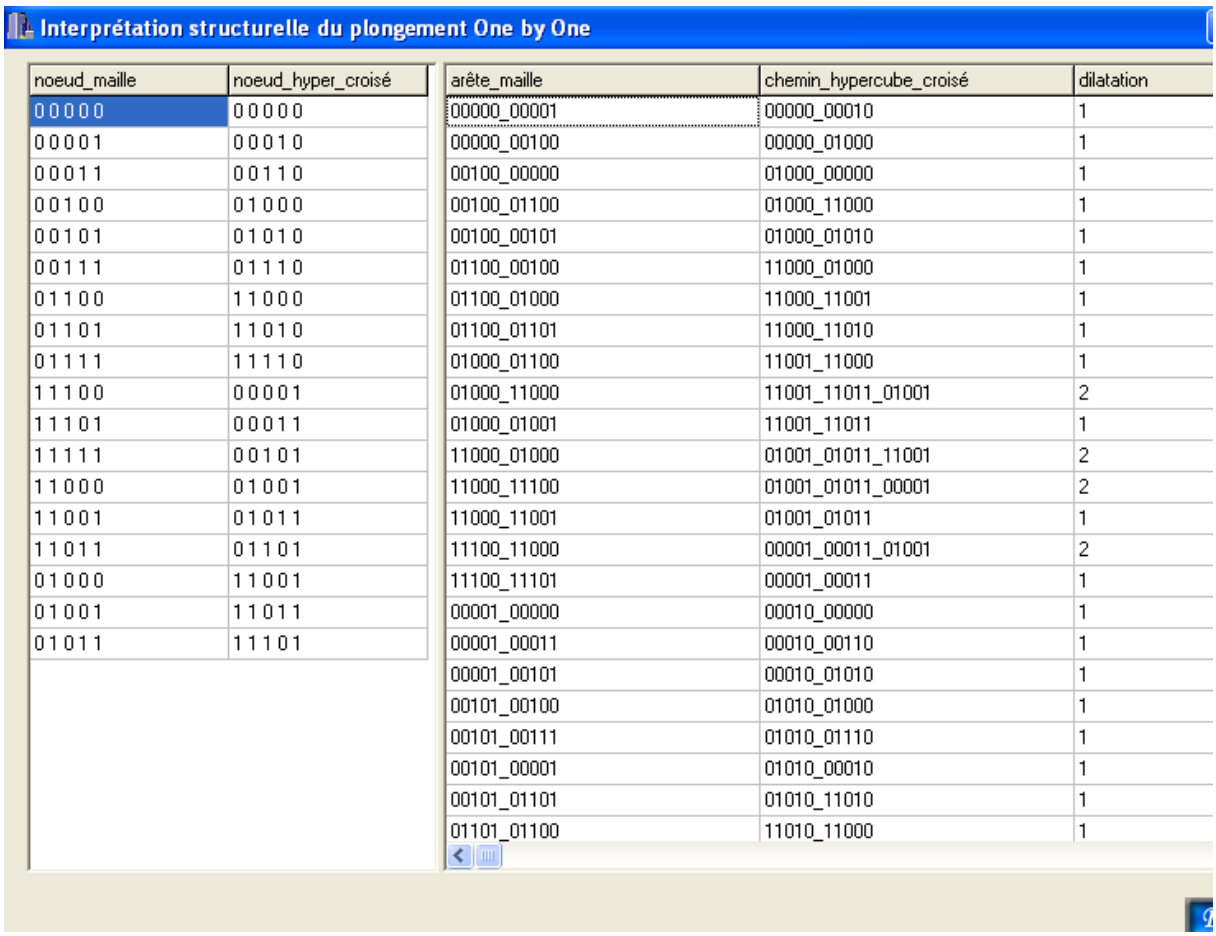
noeud_maille	noeud_hyper_croisé
00000	00000
00001	00010
00011	00110
00100	01000
00101	01010
00111	01110
01100	11000
01101	11010
01111	11110
11100	00001
11101	00011
11111	00101
11000	01001
11001	01011
11011	01101
01000	11001
01001	11011
01011	11101

Cette fiche nous montre le plongement des différents nœuds de la maille dans l'hypercube croisé en utilisant le plongement one by one par exemple le nœud 01111 de la maille correspond à 11110 dans l'hypercube croisé

Interprétation structurelle du plongement One by One

arête_maille	chemin_hypercube_croisé	dilatation
00000_00001	00000_00010	1
00000_00100	00000_01000	1
00100_00000	01000_00000	1
00100_01100	01000_11000	1
00100_00101	01000_01010	1
01100_00100	11000_01000	1
01100_01000	11000_11001	1
01100_01101	11000_11010	1
01000_01100	11001_11000	1
01000_11000	11001_11011_01001	2
01000_01001	11001_11011	1
11000_01000	01001_01011_11001	2
11000_11100	01001_01011_00001	2
11000_11001	01001_01011	1
11100_11000	00001_00011_01001	2
11100_11101	00001_00011	1
00001_00000	00010_00000	1
00001_00011	00010_00110	1
00001_00101	00010_01010	1
00101_00100	01010_01000	1
00101_00111	01010_01110	1
00101_00001	01010_00010	1
00101_01101	01010_11010	1
01101_01100	11010_11000	1
01101_01111	11010_11110	1

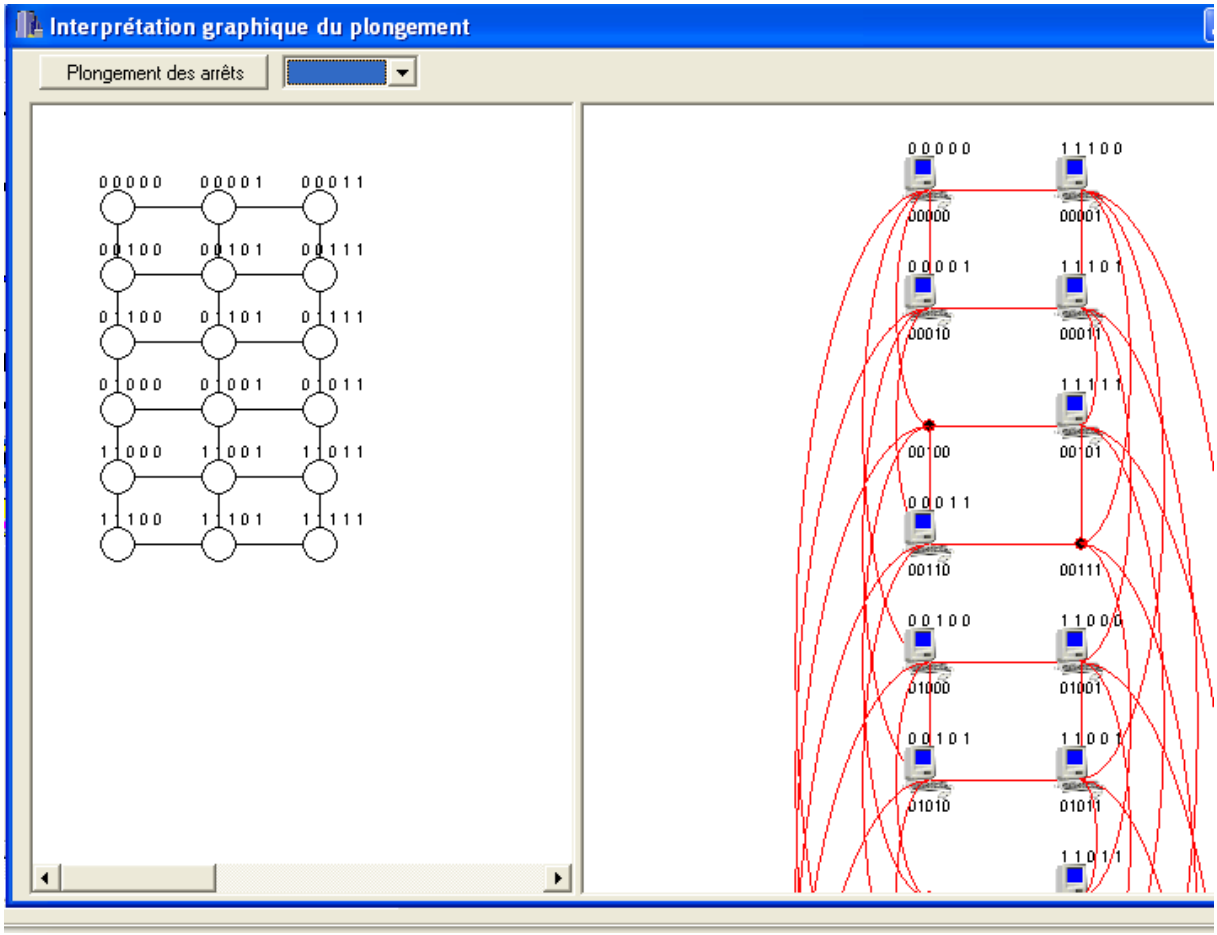
Cette fiche nous montre les résultats de l'exercice du plongement one by one des différentes arêtes de la maille dans des chemins de longueurs égale à deux dans l'hypercube croisé. Par exemple 11000 – 11100 correspond à 01001-01011-00001 la dilatation est égale à deux.



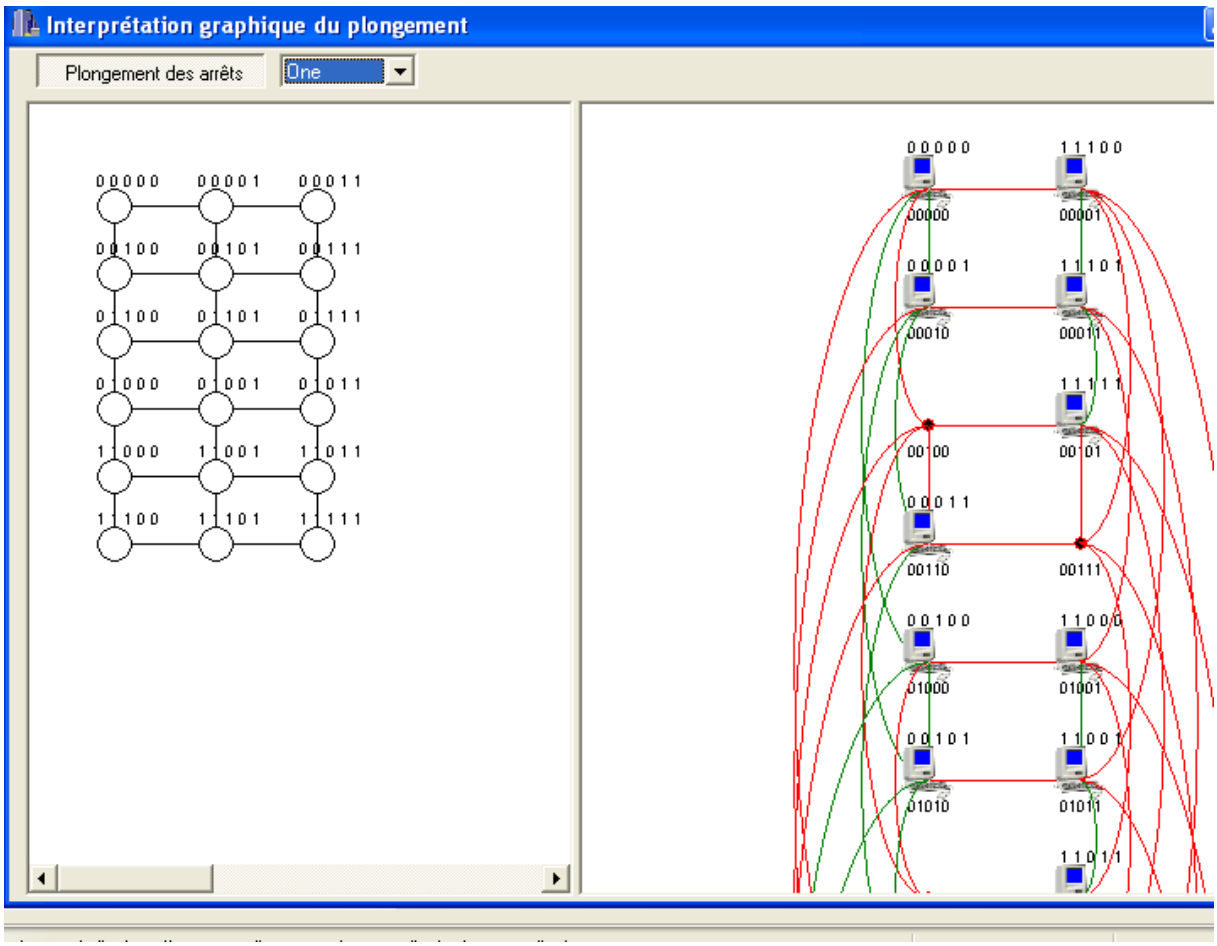
noeud_maille	noeud_hyper_croisé
00000	00000
00001	00010
00011	00110
00100	01000
00101	01010
00111	01110
01100	11000
01101	11010
01111	11110
11100	00001
11101	00011
11111	00101
11000	01001
11001	01011
11011	01101
01000	11001
01001	11011
01011	11101

arête_maille	chemin_hypercube_croisé	dilatation
00000_00001	00000_00010	1
00000_00100	00000_01000	1
00100_00000	01000_00000	1
00100_01100	01000_11000	1
00100_00101	01000_01010	1
01100_00100	11000_01000	1
01100_01000	11000_11001	1
01100_01101	11000_11010	1
01000_01100	11001_11000	1
01000_11000	11001_11011_01001	2
01000_01001	11001_11011	1
11000_01000	01001_01011_11001	2
11000_11100	01001_01011_00001	2
11000_11001	01001_01011	1
11100_11000	00001_00011_01001	2
11100_11101	00001_00011	1
00001_00000	00010_00000	1
00001_00011	00010_00110	1
00001_00101	00010_01010	1
00101_00100	01010_01000	1
00101_00111	01010_01110	1
00101_00001	01010_00010	1
00101_01101	01010_11010	1
01101_01100	11010_11000	1

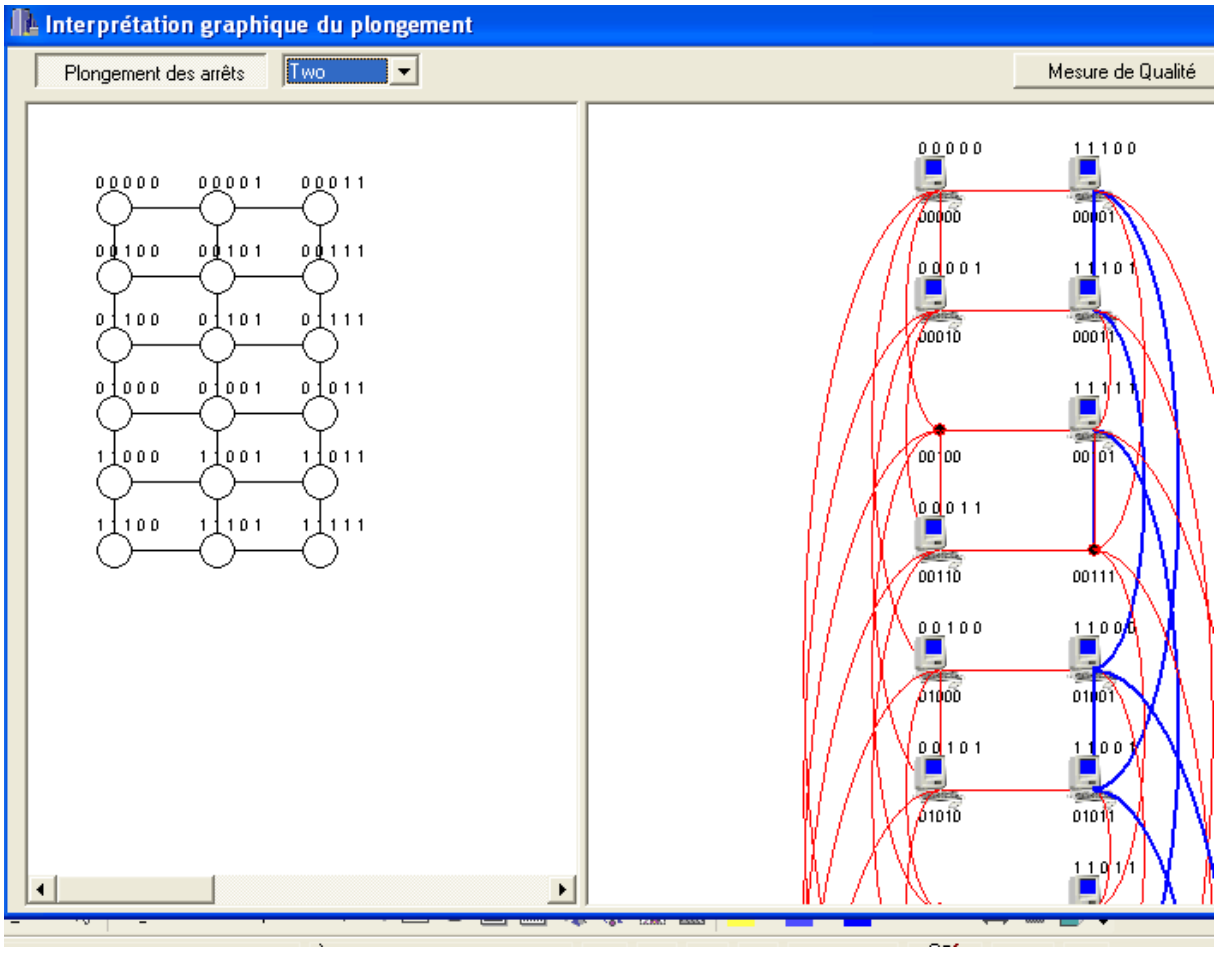
Cette fiche nous montre le graphe de plongement de la maille dans l'hypercube croisé autrement dit celui des nœuds et des arêtes ainsi que le dilatation.



Cette fiche nous montre graphiquement le plongement des nœuds de la maille dans l'hypercube croisé.



Cette fiche interprète graphiquement le plongement des arêtes de longueur une de la maille dans l'hypercube croisé.


Interprétation graphique du plongement
Plongement des arrêts
Two
Mesure de Qualité
00000
00001
00011
00100
00101
00111
01100
01101
01111
01000
01001
01011
11000
11001
11011
11100
11101
11111
00000
00001
00010
00011
00100
00101
00110
00111
01000
01001
01010
01011
11000
11001
11011

Cette fiche interprète graphiquement le plongement des arêtes de longueur une de la maille dans des chemins de longueur à deux dans l'hypercube croisé.

Elle interprète les mesures de qualité du plongement one by one à savoir :

- ✓ La dilatation = 2
- ✓ L'expansion = 1
- ✓ La congestion = 2

7-Conclusion

La solution proposée à été conçue par l'approche objet ceci nous à conduit à choisir un langage de programmation adéquat aux de détails de cette approche qui est le langage C++ builder ce dernier ; dispose d'une bibliographie très riche.

La solution aussi réalisée nous a amenés au résultats attendus et répondant aux objectifs fixés à chaque sous système à savoir la simulation exacte de l'architecture maille dans toutes ses forme (Rectangulaire horizontal Rectangulaire verticale et carrés)

- ✓ La simulation exacte de l'architecture hypercube croisé
- ✓ Une émulation de bonne qualité pour ses formes fonctionnelles à savoir :
  - Le plongement many by one avec une mesure sur la dilatation qui est égale à deux, l'expansion optimale égale à un et un facteur de change égal a deux.
  - Le plongement de one by one à été réalisé correctement avec une mesure sur la dilatation égale deux , une expansion égale à presque a un et une mesure de la congestion égale a deux.

Conclusion générale

L'un des plus importants facteurs qui gouvernent la performance dans les machines parallèles est la communication dans les réseaux d'interconnexion. Plusieurs réseaux d'interconnexion sont introduits dans la littérature. Les plus importantes propriétés de ces réseaux dans le futur sont : Le diamètre, le degré des nœuds et la tolérance aux pannes. L'hypercube a obtenu un gain de cause vu qu'il offre des propriétés très attractives. L'hypercube croisé préserve ces propriétés et réduit le diamètre par un facteur égal à deux (02).

Aussi, à travers cette recherche, nous avons montré la relation et les différentes transformations entre l'hypercube croisé et les autres architectures. Il s'est avéré également nécessaire de présenter dans ce travail le plongement de l'architecture maille dans l'architecture de l'hypercube croisé sous deux formes de schémas :

- ✓ Le premier, consiste à plonger une architecture maille dans l'hypercube croisé en utilisant une application injective many by one ou une arête élémentaire dans la maille a une image dans l'hypercube croisé égale à un chemin dont la distance est égale au maximum à deux arêtes, et deux nœuds du graphe source maille correspondent à un seul nœud dans l'hypercube croisé. Cette application de plongement optimise l'expansion du graphe à un et réalise un facteur de charge dans les nœuds du graphe de plongement à deux.
- ✓ Dans le deuxième schémas, nous avons réalisé un plongement de la maille dans l'hypercube croisé en utilisant une application bijective One by one ou une arête élémentaire assurant la connexion de deux nœuds dans la maille est appliqué en un chemin de longueur égale à deux dans l'hypercube croisé, une expansion égale à un (optimal) et une congestion d'arêtes égale à deux.

Ces deux schémas ont été implémentés en utilisant la technique de simulation par activité vu l'absence de machines parallèles maille et hypercube croisé. Notre émulateur de plongement a été testé et a montré la capacité de simulation d'une architecture maille dans une architecture hypercube croisé avec une qualité optimale d'expansion , un très bon délai de communication entre les différents processus assignés au processeur du graphe de plongement qui est égale à deux (dilatation) et une bonne congestion des arêtes du graphe de plongement égale à deux.

Perspectives :

Dans les futures recherches, nos intentions seront certainement sur d'autres questions aussi importantes notamment :

- ✓ Le plongement d'autres architectures sources dans l'hypercube croisé ;
- ✓ L'étude de plongement dans un hypercube croisé présentant des pannes ;
- ✓ L'amélioration de la qualité des différents plongements existants dans l'hypercube croisé présentant des pannes ;

References Bibliographiques

[AD,IS89]	A. Dingle and I.Sudborough, 'Simulating Binary trees and X-trees on pyramid networks", Proc 11 ST IEEE Symposium on parallel and distributed processing 1989
[AE,LN,BS88]	A. Esfahanian, L.Ni, B. Sagan « On enhancing hypercube multiprocessors » Proc 1988 international Parallel processing 86-89.
[AG,HW92]	A.Gupta and H.Wong "Optimal Embeddings of ternary trees into Boolean hypercubes" Proc 44th IEEE Symoposum on parallel and distributed processing, 1992
[AW,RC,EM90]	A. Wang, R.Cypher and E. Maye "Embedding binary trees in faculty hypercubes", Proc 3 rd IEEE symposium on parallel and distributed processing 1990
[BM,IS88]	B. Monien and I.Sudbourough,"Simulating Binary trees on hypercubes" Proc 3 rd Aegean Workshop on computing lecture notes in computer sciences, 1988
[BP94]	S Bettayeb, K. Priyalal "On the multiply-twisted hypercube" Proc of ISCA int.conf on design, simulations and analysis,pp 1-4,1994
[CB73]	Claud Berge : « Graphe et hyperGraphe (deuxièmeEditions BORDAS 1973).
[CT,HK77]	C. Thompson, H. Kung "Sorting on a mesh-connected parallel computer" communication of the ACM, 20(4) : pp 263-271, April 1977
[DN,SS80]	D. Nassimi, S Sahni "Finding connected components and connected ones on a mesh-parallel computer" SIAM Journal on computing, 9(4) : pp 744-757, 1980
[DT00]	Daniel Tabak : Les multi-processuers , IntelEditions PARIS 2000
[EA93]	Emadeddin Abuelrub: "Interconnection Networks Embeddings And Efficient Parallel Computations "(May 1993).LSU USA
[EU,AW87]	E.Upfal ,A.Wigderson "How to share memory in distributed system" Journal of ACM, 34(1):pp116-127,january 1987
[EU82]	E.Upfal "Efficient schemes for parallel communication" in ACM SIGACT-SIGOPS Symposium on principles of distributed computing :pp55-59,August 1982
[FP,JV81]	F. Preparata and J.Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," Communication of the ACM, Vol 24,no 5,pp. 300-309, May 1981.

[FP,RM89]	F. Provost and Melhem, " Distributed fault-tolerant embedding of binary trees and rings in hypercubes" Proc International workshop on defect and fault-tolerance in VLSI systems 1989
[GA,AG89]	G. Almasi, A. Gottlieb "Highly parallel computing" Benjamin Cumming, red wood city, CA,1989
[GA,EJ75]	G;Anderson and E. Jensen 'Computer interconnection structure taxonomy, characteristics, and examples" Computing surveys, vol 7 pp 197-213, December 1975
[GD01]	Gilles Deghilage ,"Architectures et programmation parrallèles Approche pratique en environnement scientifique sur multiprocesseurs SILICON GRAPHICS" 2001
[GH,KM,AR83]	G;Hong, K. Mehlhorn , A.Rosenberg „Coste Trade-Offs in graphe Embeddings with application" Journal of the association computer machinary, vol 30, n° 4, pp 709-728, october 1983.
[HA,TH,KM,FP87]	H.Alt, T.Hagerup, K.Mehlhorn, F.Preparata "Deterministic, simulation of idealised parallel computers on more realistic ones", SIAM journal on computing,16(5) : pp808-835, October 1987
[JH,FL,MN87]	J. Hastad, F. Leighton, and M Newman , "Fast Computation Using Faulty Hyercubes," Proc. 19th Annual ACMSTOC, 1987
[JJ,SL,SD90]	J. Jow, S.Lakshmivarahan and S.Dhall "Embedding of cycles and grids in star graphs" Proc 2 nd IEEE symposium on parallel and distributed processing 1990
[JPS02]	Jean-Paul SanSonnet : « Informatique parallèle et Système multiprocesseurs ». InterEdition Paris (2002)
[KE 92]	K.Efe "The crossed cube architecture for parallel computation" IEEE transaction on parallel and distributed systems, vol 3, n° 5, pp 513-524, September 1992
[KE,PB,TS,WS88]	K. Efe, P. Blackwell, T. Shiau and Slough, "A reduced Diameter Interconnection Network," Technical Report, Department of Computer Science, university of Missouri, 1988.
[LB,SL,SD89]	L. Barasch, S. Lakshmivarahan and S.Dhall "Embedding arbitrary meshes and complete binary trees in generalized hypercubes" Proc 1st IEEE symposium on parallel and distributed processing 1989
[LB,SL,SD89]	L. Barasch, S. Lakshmivarahan and S.Dhall "Embedding arbitrary meshes and complete binary trees in generalized hypercubes" Proc 1st IEEE symposium on parallel and distributed processing 1989
[LEI92]	T. Leighton, " Introduction to parallel algorithms and architectures : arrays, trees, hypercubes" Mogan Kaufman

	publisher inc) 1992
[LV82]	L; Valiant "A scheme for fast parallel communication" SIAM journal on computing,11(2):pp350-361, mai 1982
[MC,SL91]	M. Chan and S. Lee, "Distributed Fault tolerance Embeddings of Rings into Hypercubes, "Jornal of parallel and Distributed computing, no 11.pp 63-711, 1991.
[MC93]	Michel Cosnard et Denis Trystram : « Algorithme et achitecture parallèle »(IntelEditions, Paris 1993).
[MK01]	Mohamed Kobeissi : « Plongement de graphes dans l'hypercube » Université de Joseph Fourier-Grenoble1 U.F.R.IMA , le 12 octobre2001).
[MQ,ND84]	M. Quinn, N. Deo "Parallel Graph Algorithm" ACM computing surveys ,16(3): pp 319-348,september 1984
[RA,AR82]	R. Aleliunas and A.Rosenberg "On embedding Rectangular grids in square grids" IEEE Transaction on computers Vol C-31,pp 907-913 1982
[SB,and all92]	S. Bettayeb, Z. Miller and I.Sudborough "Embedding grid in to hypercube" journal of computer and system sciences, vol 45 N° 03,pp 345-366, December 1992
[SB,BC and all92]	S. Bettayeb, I.Sudborough, B. Cong, M. Gerou "Embedding simulations networks into hypercubes", Latin 1992
[SB,BC,MG,AR92]	S. Bettayeb, B. Cong, M. Girou, And I. Sudborough, "Ebedding Simulation Networks into Hypercubes," LATIN 92, 11992.
[SB,FC and all86]	S.Bhatt, F; Chung,F Leighton, A.Rosenberg „ Optimal simulations of tree machines" Proc 27 th annual IEEE Foundation of computer science conference.
[SB,FC,FL,AR86]	S.Bhatt, F; Chung,F Leighton, A.Rosenberg „ Optimal simulations of tree machines" Proc 27 th annual IEEE Foundation of computer science conference.
[SB,II85]	S.Bhatt and I.Ispen : "How to embeb trees in hypercubes",Techical report,Departement of computer science, Yale university 1985
[SB,IS,MG,AR92]	S. Bettayeb, I.Sudborough, B. Cong, M. Girou "Embedding simulations networks into hypercubes", Latin 1992
[SB,IS89]	S. Bettayeb, I.Sudborough "Grid embedding into ternary

	hypercubes" Proc 1989 ACM South central regional conference, 1989
[SB,ZM,IS92]	S. Bettayeb, Z. Miller and I.Sudborough "Embedding grid in to hypercube" journal of computer and system sciences, vol 45 N° 03,pp 345-366, December 1992
[SB02]	Said Bettayeb: Parallel Processing (2002, Louisiana State University, Department of computer Science).
[SB94]	Said Bettayeb, Priyalal Kulasinghe: "Symmetry of the Crossed Cube" (november 1994, Louisiana State University, Department of computer Science).
[SL,AE188]	S.Latifi and A. Al-Amawy, "On embedding binary trees in 3-D meshe ayyay" Proc 1st IEEE symposium on parallel and distributed processing 1989
[SL,AE88]	S.Latifi and A. Al-Amawy, "On Folded-Hypercubes" Proc International conference on parallel processing 1989
[ST,PB,KE88]	Shiau, T., P. Blackwell, and K. Efe .(1988), " Multipy-twisted n-cubes for parallel computing ", Proc. Of 20th Symp. On the Interface : Compuing Science and Statistics, to appear.
[SZ91]	S. zheng, "SIMD Data Communication Algorithms for Multiply-Twisted Hypercubes," Proc 5th Internationnal Parallel Processing Symposium, 1991
[TL,HE92]	T.Lewis and H El-Rewini, "Introduction to parallel computing" Prentice Hall,1992
[YS,MS88]	Y. Saad and M Schultz "Topoligical proprieties of the hypercube" IEEE Transaction on computers vol C-37, n° 7, pp 867-872, july 1988