USTL
FLANDRES ARTOIS

LIFL du C.N.R.S

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

N° d'ordre : 436

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

ARNAUD Patrick

PROPOSITION D'UNE MÉTHODE DE RÉPARTITION DE LA CHARGE SUR UN RÉSEAU DE PROCESSEURS : CONSÉQUENCES SUR LA TOPOLOGIE ET SIMULATION.

Thèse soutenue le 20 Novembre 1989 devant la commission d'Examen

Membres du jury

V. CORDONNIER	Président
P. BAKOWSKI	Rapporteur
A.F. MOUYART	Rapporteur
B. TOURSEL	Directeur de thèse
G. GONCALVES	Examinateur
M.P. LECOUFFE	Examinateur

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE
FLANDRES ARTOIS

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M.H. LEFEBVRE, M. PARREAU.

PROFESSEURS HONORAIRES DES ANCIENNES FACULTES DE DROIT ET SCIENCES ECONOMIQUES, DES SCIENCES ET DES LETTRES

MM. ARNOULT, BONTE, BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, DECUYPER, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, GERMAIN, GLACET, GONTIER, KOURGANOFF, LAMOTTE, LASSERRE, LELONG, LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, PEREZ, ROIG, ROSEAU, ROUELLE, SCHILTZ, SAVARD, ZAMANSKI, Mes BEAUJEU, LELONG.

PROFESSEUR EMERITE

M. A. LEBRUN

ANCIENS PRESIDENTS DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. M. PAREAU, J. LOMBARD, M. MIGEON, J. CORTOIS.

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

M. A. DUBRULLE.

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CONSTANT Eugène	Electronique
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie Appliquée

PROFESSEURS - 1ère CLASSE

M. BACCHUS Pierre	Astronomie
M. BIAYS Pierre	Géographie
M. BILLARD Jean	Physique du Solide
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean-Pierre	Chimie-Physique
M. BOSCQ Denis	Probabilitéss
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BREZINSKI Claude	Analyse Numérique

M. BRIDOUX Michel	Chimie-Physique
M. CELET Paul	Géologie Générale
M. CHAMLEY Hervé	Géotechnique
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. DAUCHET Max	Informatique
M. DEBOURSE Jean-Pierre	Gestion des Entreprises
M. DHAINAUT André	Biologie Animale
M. DOUKHAN Jean-Claude	Physique du Solide
M. DYMENT Arthur	Mécanique
M. ESCAIG Bertrand	Physique du Solide
M. FAURE Robert	Mécanique
M. FOCT Jacques	Métallurgie
M. FRONTIER Serge	Ecologie Numérique
M. GRANELLE Jean-Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. LABLACHE-COMBIER Alain	Chimie Organique
M. LACOSTE Louis	Biologie Végétale
M. LAVEINE Jean-Pierre	Paléontologie
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique Atomique et Moléculaire
M. LEROY Jean-Marie	Spectrochimie
M. LHOMME Jean	Chimie Organique Biologique
M. LOMBARD Jacques	Sociologie
M. LOUCHEUX Claude	Chimie Physique
M. LUCQUIN Michel	Chimie Physique
M. MACKE Bruno	Physique Moléculaire et Rayonnements Atmosph.
M. MIGEON Michel	E.U.D.I.L.
M. PAQUET Jacques	Géologie Générale
M. PETIT Francis	Chimie Organique
M. POUZET Pierre	Modélisation - calcul Scientifique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. SALMER Georges	Electronique
M. SCHAMPS Joel	Spectroscopie Moléculaire
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. TILLIEU Jacques	Physique Théorique
M. TOULOTTE Jean-Marc	Automatique
M. VIDAL Pierre	Automatique
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ALLAMANDO Etienne	Composants Electroniques
M. ANDRIES Jean-Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BART André	Biologie animale
M. BASSERY Louis	Génie des Procédés et Réactions Chimiques

Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique Atomique et Moléculaire
M. BERTRAND Hugues	Sciences Economiques et Sociales
M. BERZIN Robert	Analyse
M.BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie Végétale
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie Civil
M. BOIVIN Jean-Claude	Spectroscopie
M. BOUQUELET Stéphane	Biologie Appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BRASSELET Jean-Paul	Géométrie et Topologie
M. BRUYELLE Pierre	Géographie
M. CAPURON Alfred	Biologie Animale
M. CATTEAU Jean-Pierre	Chimie Organique
M. CAYATTE Jean-Louis	Sciences Economiques
M. CHAPOTON Alain	Electronique
M. CHARET Pierre	Biochimie Structurale
M. CHIVE Maurice	Composants Electroniques Optiques
M. COMYN Gérard	Informatique Théorique
M. COQUERY Jean-Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques et Sociales
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique Nucléaire et Corpusculaire
M. COUTURIER Daniel	Chimie Organique
M. CRAMPON Norbert	Tectolique Géodynamique
M. CROSNIER Yves	Electronique
M. CURGY Jean-Jacques	Biologie
Melle DACHARRY Monique	Géographie
M. DEBRABANT Pierre	Géologie Appliquée
M. DEGAUQUE Pierre	Electronique
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean-Paul	Informatique
M. DELORME Pierre	Physioliogie Animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
M. DENEL Jacques	Informatique
M. DE PARIS Jean Claude	Analyse
M. DEPREZ Gilbert	Physique du Solide - Cristallographie
M. DERIEUX Jean-Claude	Microbiologie
Melle DESSAUX Odile	Spectroscopie de la réactivité Chimique
M. DEVRAINNE Pierre	Chimie Minérale
Mme DHAINAUT Nicole	Biologie Animale
M. DHAMELINCOURT Paul	Chimie Physique
M. DORMARD Serge	Sciences Economiques
M. DUBOIS Henri	Spectroscopie Hertzienne
M. DUBRULLE Alain	Spectroscopie Hertzienne
M. DUBUS Jean-Paul	Spectrométrie des Solides
M. DUPONT Christophe	Vie de la firme (I.A.E.)
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FAUQUAMBERGUE Renaud	Composants électroniques

M. FONTAINE Hubert	Dynamique des cristaux
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie Sturcturale
M. GAMBLIN André	Géographie urbaine, industrielle et démog.
M. GLORIEUX Pierre	Physique moléculaire et rayonnements Atmos.
M. GOBLOT Rémi	Algèbre
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie Physique
M. GOURIEROUX Christian	Probabilités et Statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean-Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLOT Jean	Chimie Organique
M. GUILBAULT Pierre	Physiologie animale
M. HENRY Jean-Pierre	Génie Mécanique
M. HERMAN Maurice	Physique spatiale
M. HOUDART René	Physique atomique
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et Statistiques
M. Jean Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme (I.A.E.)
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KREMBEL Jean	Biochimie
M. LANGRAND Claude	Probabilités et statistiques
M. LATTEUX Michel	Informatique
Mme LECLERCQ Ginette	Catalyse
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
Melle LEGRAND Solange	Algèbre
M. LEGRAND Pierre	Chimie
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne
M. LE MAROIS Henri	Vie de la firme (I.A.E.)
M. LEROY Yves	Composants électroniques
M. LESENNE Jacques	Systèmes électroniques
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique théorique
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU Jean-Marie	Optique-Physique atomique
M. MAIZIERES Christian	Automatique
M. MAURISSON Patrick	Sciences Economiques et Sociales
M. MESMACQUE Gérard	Génie Mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie Organique
M. MORTREUX André	Chimie Organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
Mme MOUYART-TASSIN Annie Françoise	Informatique
M. NICOLE Jacques	Spectrochimie
M. NOTELET Françis	Systèmes électroniques
M. PARSY Fernand	Mécanique

M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. STEEN Jean-Pierre	Informatique

REMERCIEMENTS

Je tiens à remercier

Mr le Professeur Vincent Cordonnier, qui a bien voulu me faire l'honneur de présider le jury ;

Mr le Professeur Przemyslaw Bakowski, qui a accepté d'en être l'un des rapporteurs ;

Mme le Professeur Annie-Françoise Mouyart, non seulement pour sa participation au jury, en tant que rapporteur, mais aussi pour l'aide précieuse qu'elle m'a apportée tant dans l'élaboration de cette thèse que dans sa rédaction ;

Mr le Professeur Bernard Toursel, qui a été l'initiateur de ce travail et en a suivi la progression, la favorisant par ses conseils ;

Mme Marie-Paule Lecouffe et Mr Gilles Goncalves, qui m'ont fait l'amitié d'en être examinateurs.

Je remercie également les membres de l'équipe N-ARCH avec qui il a été très agréable de travailler, ainsi que mes collègues du département informatique de l'IUT A de Lille, pour leur aide et leur patience lors de la frappe de cette thèse.

Merci enfin à Mr Glanc qui en a assuré la reproduction.

INTRODUCTION

0.1 Limites des machines classiques

Les ordinateurs "classiques", c'est-à-dire basés sur les principes définis par von Neumann, eux-mêmes issus de la fameuse Machine de Turing, semblent se rapprocher inexorablement de leurs limites d'un point de vue technologique.

En effet, depuis leur apparition (ENIAC), ces machines répondaient aux besoins croissants en vitesse d'exécution et en performance grâce aux progrès accomplis en technologie des circuits logiques. La vitesse de commutation est ainsi passée d'une valeur de l'ordre du 10^{e} de seconde à une valeur de l'ordre de la nanoseconde. L'intégration des composants a, elle aussi, fait un bond quantitatif impressionnant : après les temps archéologiques du relais électromécanique, puis du tube à vide, on est passé au transistor puis aux circuits faiblement intégrés (MLSI) comportant plusieurs centaines de transistors, puis aux circuits largement intégrés (LSI) contenant plusieurs milliers de transistors ; puis aux circuits à très haute intégration (VLSI) : plusieurs centaines de milliers de transistors. On en arrive à envisager maintenant des circuits à ultra-haute intégration : ULSI (des millions de transistors par composant) et même GLSI (des milliards de transistors par composant) !

Mais ces courses à la vitesse de commutation et à l'intégration vont devoir s'arrêter, ne serait-ce que pour des raisons physiques.

On voit donc que l'accélération de la vitesse d'une machine va devenir de plus en plus difficile si l'on s'en tient aux seules améliorations technologiques.

0.2 Surpasser les limites

Une première solution pour accroître, à technologie égale, les possibilités des ordinateurs a été le recours à des machines à plus ou moins haut degré de parallélisme, dont la réalisation bénéficie évidemment de la "course à l'intégration" exposée ci-dessus.

Cette réponse a toutefois amené un nouveau problème : celui de la programmation de telle machines. Des transformations ou extensions en vue de "paralléliser" des langages traditionnels (procéduraux) préexistants (voire préhistoriques, cf FORTRAN) ont été utilisées mais ces langages restent trop lourdement hantés par l'esprit von Neumann : la base théorique des machines von Neumann est la machine de Turing, séquentielle par définition!

D'autre part, essayer de conserver des modes de fonctionnement traditionnels (mémoires partagées,...) impose un frein à la mise en oeuvre de parallélisme à grande échelle.

Pour résoudre ces problèmes, les recherches se sont orientées suivant deux axes non nécessairement distincts :

(1) vers la conception de machines dites non von Neumann - machines dataflow, machines de réduction, etc...-

et/ou

(2) vers l'utilisation de langages déclaratifs - langages fonctionnels, logiques ...-.

C'est dans cette double optique que se situe le projet N-ARCH, pour Nouvelle ARCHitecture, dans le cadre duquel ont été effectués les travaux présentés ici.

La base de l'architecture envisagée est un réseau de processeurs pour lequel les besoins de répartition des tâches et des objets manipulés ont amené à définir un procédé de hachage et une topologie de réseau particulières.

Après un premier chapitre consacré à la distribution des objets et de la charge en système fortement couplé, puis un deuxième où sera présenté le projet N-ARCH, nous nous intéresserons, dans le chapitre 3, au hachage appliqué au contexte particulier que représente une architecture complètement distribuée, ce qui nous amènera à définir une technique de hachage originale.

Le chapitre 4 sera consacré à l'étude du réseau, pour lequel on verra que les topologies généralement utilisées ne conviennent pas, et à la présentation des réseaux retenus.

Le chapitre 5 présente les résultats de simulation obtenus, dans divers cas, avec les réseaux définis au chapitre 4.

Enfin, dans le chapitre 6, on évoquera d'autres travaux approchants réalisés dans le cadre du projet N-ARCH.

CHAPITRE 1

DISTRIBUTION EN SYSTEME FORTEMENT COUPLE.

1.1 Répartition de charge dans les systèmes distribués parallèles

Dans les systèmes distribués l'un des problèmes les plus importants à résoudre est celui de l'allocation des données et des tâches. On peut choisir de favoriser la localité des traitements pour minimiser les temps et coûts de communication : il faudra alors opérer de nombreuses duplications de données, et même éventuellement d'actions sur ces données, sur divers processeurs élémentaires. A l'opposé, on peut décider de dupliquer le moins possible d'objets (au sens large : un objet peut consister en un opérande pour une opération, une instruction, un segment de code ... etc ...) et, dans ce cas, la quantité de communications va être accrue.

Ces problèmes d'allocation ont fait l'objet de nombreuses études et les méthodes employées dans la plupart des cas sont spécifiques. De nombreux travaux se concentrent uniquement sur la distribution de tâches indépendantes (régulation de charge) ou sur les bases de données distribuées (distribution de données). On trouve cependant des articles traitant à la fois de répartition de charge et de distribution des données comme par exemple [MARC81]. Cette étude concerne à la fois les réseaux locaux, les réseaux géographiques et les machines multiprocesseurs. L'allocation y est abordée sous deux aspects : **l'allocation de données** sert à déterminer les capacités mémoires à mettre en oeuvre, tandis que **l'allocation de modules** aux processeurs doit permettre un équilibrage de la charge et une minimisation des communications. Les auteurs présentent 2 modèles généraux d'allocation utilisant amplement la recopie. L'allocation de données est abordée en fait sous forme d'allocation de fichiers. L'allocation de modules est faite par l'intermédiaire d'un découpage en modules le plus indépendants possible pour minimiser les communications. C'est la taille des modules qui détermine la charge des processeurs.

Généralement, indépendamment de la répartition des données, lorsqu'une tâche doit être exécutée dans un univers distribué et plus particulièrement sur une machine multiprocesseurs, cette tâche, ou plus exactement l'algorithme la représentant, est subdivisée en sous tâches pouvant tourner de manière concurrente

(1) en échangeant des données,

(2) avec certaines "précédences" c'est-à-dire que certaines sous-tâches ne peuvent démarrer qu'après l'achèvement de certaines autres.

Le problème est : comment attribuer ces sous-tâches de manière optimale aux processeurs ? Poser ce problème amène à en soulever un autre : quel sens attribuer à "de manière optimale" ? Les critères utilisés vont rejoindre ceux évoqués plus haut : ce sera

tantôt la minimisation du coût des communications interprocesseurs, tantôt la maximisation du rendement global, tantôt la minimisation du temps de réponse.

Un autre point de vue sépare les différents procédés : l'allocation, voire la subdivision de tâche en sous-tâches, peuvent être faite de façon statique ou dynamique.

Dans le cas d'une allocation statique, le traitement à effectuer est généralement représenté par un graphe (ce graphe étant un DAG : Directed Acyclic Graph), dans lequel les noeuds représentent les sous-tâches et les arcs les contraintes de précédence à prendre en compte et/ou les échanges de données.

Dans [POLY86], les auteurs s'intéressent à la distribution de programme parallèles sur des multiprocesseurs à mémoire partagée. Le critère d'optimisation est le temps d'exécution. Dans les programmes considérés, le parallélisme est explicitement spécifié sous formes de tâches qui sont en fait des boucles parallèles. L'allocation est faite de manière statique par l'intermédiaire d'un DAG construit avant l'exécution.

Le problème consiste à affecter les sous-tâches aux processeurs de manière optimale en fonction du critère de jugement utilisé. Si, par exemple, le critère retenu est la minimisation des communications, il faudra, autant que possible, affecter à un même processeur une sous-tâche donnée t_0 et toutes les sous-tâches dont t_0 dépend. On aboutit donc à une perte de parallélisme, à un accroissement de la charge de certains processeurs et par suite à une certaine hétérogénéité de la répartition de la charge.

Dans le cas d'une allocation dynamique, celle-ci est faite au cours du traitement et peut être réorganisée à tout instant. Plusieurs possibilités sont envisageables.

Première possibilité : un site "maître" scrute la charge de chacun des processeurs et détecte les processeurs/sites inactifs ou peu actifs ou, au contraire, ceux dont la charge est trop importante. Une redistribution de la charge peut à tout instant être décidée.

Cette méthode est par exemple utilisée pour l'architecture décrite dans [CRAM85]. Il s'agit d'une architecture multiprocesseurs pour laquelle l'allocation des processus aux processeurs est faite par un "scheduler" tournant sur un processeur dédié. Pour cette architecture le problème de la distribution des données ne se pose pas, du fait de l'utilisation d'une mémoire globale accessible par tous les processeurs (ceux-ci disposant tout de même de mémoire locale utilisée en particulier pour l'unification, le langage utilisé étant PARLOG).

Cette méthode n'a pas lieu d'être dans la machine N-ARCH puisque, comme on le verra, le contrôle y est complètement distribué.

Autre possibilité : tout processeur peut déléguer une partie de sa tâche lorsque celle-ci devient trop importante. Ici aussi on a recours à des DAG comme vu plus haut à propos de l'allocation statique. Le problème est alors de savoir à qui confier les tâches en "sous-traitance". Là encore, plusieurs solutions sont envisageables :

--> envoi "à la cantonade" : la tâche est déposée sur le réseau et le parcourt jusqu'à sa prise en charge par un processeur inactif ou peu actif. La mise en oeuvre d'un tel système semble assez ardue.

--> "dialogue" entre les ou des processeurs : le(s) processeur(s) surchargé ou saturé se renseigne sur la charge des autres processeurs ou seulement de certains autres (ses voisins immédiats, par exemple) et délègue une partie de son travail à l'un ou plusieurs d'entre eux.

Ceci est équivalent à la première possibilité, mais sans site maître.

--> il existe un algorithme, connu de tous les processeurs, permettant d'effectuer la délégation des sous-tâches. Cette solution semble restreinte à certains cas particuliers.

Dans tous les cas, l'allocation dynamique, pour être plus efficace, peut être précédée d'une allocation statique.

Pour être efficace, l'une et l'autre méthode doivent éviter de trop ralentir le traitement proprement dit.

Solution N-ARCH : Le parti pris est celui de répartir le plus possible toutes les entités manipulées dans le traitement : données, fonctions travaillant sur ces données ... etc ... A chaque entité est associé un nom d'objet qui identifie à la fois l'objet et son contexte. La répartition est faite par un procédé de hachage qui distribue tous les objets de manière uniforme sur les différents processeurs. C'est de l'uniformité de cette répartition que proviendront la bonne distribution de la charge, des objets et un taux de parallélisme élevé, au prix d'une quantité de communications elle aussi élevée.

1.2 Relations avec la topologie des réseaux

Pour le projet N-ARCH, une approche originale réside dans le fait que la topologie du réseau a été choisie en fonction de la technique de hachage particulière utilisée pour la répartition de la charge et des données.

Dans la littérature, la démarche générale est plutôt inverse : à partir d'une architecture existante, trouver la méthode de distribution qui sera optimale pour cette architecture, souvent même en fonction du type de programme ou de langage que l'on prévoit d'exécuter sur cette architecture.

On trouve par exemple dans [IQBA86] une comparaison de plusieurs méthodes de distribution de charge, mais dans un cas très restrictif : les programmes à répartir sont décomposables en modules formant une chaîne, c'est-à-dire que si on appelle M_0, M_1, ..., M_n les modules, $\forall\, i \in [1, n-1]$ le module M_i n'aura de dépendances qu'avec les modules M_{i-1} et M_{i+1} . Il s'agit donc d'un cadre très orienté pipeline.

On trouve toutefois dans [RAVI87] la description d'une machine de réduction dont la topologie est "dictée" par la répartition de charge. Pour cette machine, dont la topologie est basée sur les graphes de de Bruijn dont nous reparlerons au chapitre 4, on obtient une répartition de charge optimale lorsque la tâche à exécuter peut être représentée par un arbre binaire complet. Cette répartition de charge est statique.

CHAPITRE 2
PRESENTATION DU PROJET N-ARCH.

Le but du projet N-ARCH est de réaliser une machine fortement parallèle adaptée à l'exécution de programmes déclaratifs.

La base de la machine est un réseau de processeurs communiquant par messages. Son fonctionnement fait appel à des modes de contrôle non-von Neumann : contrôle par nécessité, par disponibilité. Chaque processeur dispose de mémoires associatives, l'ensemble des processeurs étant vu de manière globale comme une grande mémoire associative distribuée, gérée grâce à une technique de hachage particulière, que nous verrons dans le prochain chapitre.

Dans les paragraphes qui suivent, nous allons revenir plus en détail sur ces différentes caractéristiques.

2.1 Fonctionnement et architecture

2.1.1 Finesse du parallélisme

Le degré de parallélisme d'une machine est généralement évalué en termes de *granularité*. Cette dernière représente en quelque sorte le niveau de parallélisation du traitement auquel fonctionne la machine : niveau de l'instruction, du bloc d'instructions, du sous-programme ... Intuitivement, on peut aisément associer une granularité faible à des machines de type CRAY, ETA, alors que les machines dataflow, par exemple, seront créditées d'une granularité plus élevée.

En toute rigueur, on peut définir la granularité comme étant la mesure inverse du temps pendant lequel une tâche peut se dérouler sans rencontrer un point de communication ou de synchronisation. Les structures de grain fin (à granularité élevée) risquent de se voir engorgées par de (trop) nombreux processus de communication et/ou de synchronisation interne, alors que les structures à gros grain ont des tâches qui tournent plus longtemps sur un même processeur.

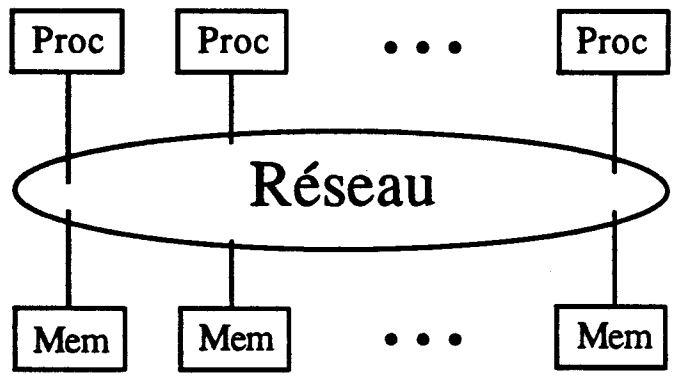
La granularité envisagée dans le projet N-ARCH est élevée. On peut donc d'ores et déjà prévoir que les communications et le réseau inter-processeurs y revêtiront une grande importance.

2.1.2 Réseau d'interconnexion

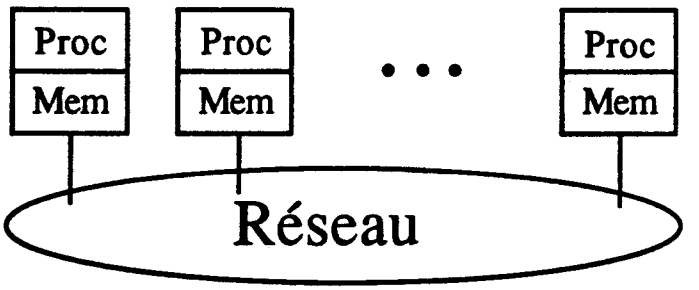
2.1.2.1 Types d'interconnexion

On peut distinguer deux catégories de réseaux pour ce qui est du mode d'interconnexion :

- l'interconnexion indirecte, dans laquelle le réseau d'interconnexion constitue une entité physique à part entière, par l'intermédiaire de laquelle communiquent soit les processeurs d'une part et les mémoires d'autre part (dans ce cas ces dernières sont partagées) (voir schéma figure 2.1.a), soit les processeurs (munis de mémoires propres) entre eux (voir schéma figure 2.1.b).



.a



.b

figure 2.1

Exemples : réseaux étagés, banyans, bénès, gamma, omega ...etc ...

- l'interconnexion directe, pour laquelle le réseau peut être modélisé par un graphe dans lequel les noeuds représentent les processeurs et les arcs les liens de communications.

La machine N-ARCH appartient à cette catégorie.

Exemples :

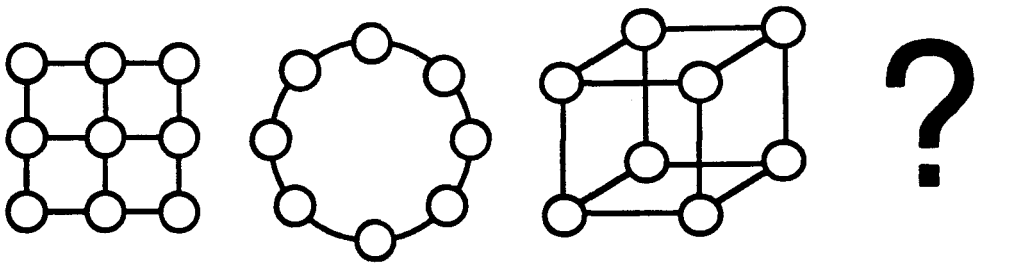


figure 2.2

2.1.2.2 Topologie

La technique de hachage évoquée au chapitre 1 sera développée dans le chapitre 3. Disons simplement ici qu'il s'agit d'un hachage à deux niveaux : un premier hachage, dit global, désigne un noeud du réseau ; en cas de collision, un deuxième hachage, dit local, désigne un autre noeud, physiquement rattaché au précédent. Le hachage local peut être réitéré à plusieurs reprises. La mise en place d'une telle technique a amené à définir une topologie particulière du réseau qui doit pouvoir être recouvert, à partir de n'importe quel noeud, par un arbre. Une (suite de) collision(s) amènera à parcourir une des branches de cet arbre de recouvrement. La définition de cette topologie fera l'objet du chapitre 5.

2.1.3 Modes de contrôle

La machine N-ARCH doit être à même de supporter des modes de contrôle de deux types : par disponibilité, par nécessité.

Le contrôle par nécessité est aussi appelé **contrôle par réduction** (de chaîne ou de graphe). Dans ce mode de contrôle, le programme est constitué de deux ensembles : un ensemble de définitions et un ensemble d'expressions à calculer ou à évaluer. L'exécution consiste en la transformation des expressions au moyen des définitions jusqu'à l'obtention d'opérandes atomiques, dont le remplacement par leurs valeurs respectives permet la réduction des expressions.

Deux approches différentes dans la manipulation des définitions conduisent à deux méthodes distinctes d'évaluation :

* dans la **réduction de chaînes**, l'évaluation d'expressions utilise les définitions comme des règles de réécriture, ce qui amène à effectuer de nombreuses copies ;

* dans la **réduction de graphe**, les définitions sont manipulées par l'intermédiaire de pointeurs. Au lieu de copier la définition, on "traverse" sa référence, on réduit la définition et la valeur équivalente est retournée.

Dès qu'une expression a été calculée, sa définition est remplacée par sa valeur. Par la suite, toute nouvelle demande de l'expression fournira ainsi directement sa valeur sans calcul.

Dans le **contrôle par disponibilité**, connu aussi sous le nom de contrôle dataflow ou dirigé par les données, on considère qu'une instruction est exécutable dès que ses opérandes sont disponibles. Généralement, un paquet, contenant l'instruction et le nom des consommateurs de son résultat, est alors envoyé à l'unité de traitement qui l'exécute et renvoie un ou plusieurs paquets résultat aux consommateurs associés à l'instruction.

2.1.4 Objets manipulés

Suivant le mode de contrôle et/ou le langage utilisés, les objets manipulés pourront être de plusieurs types que l'on peut classer en deux catégories :

* les objets statiques (par exemple générés par un éventuel compilateur) qui peuvent être :
  - des définitions d'expressions ou de fonctions (programmes en langage fonctionnel),
  - des instructions (code opération + opérandes) (programme en langage à assignation unique),
  - des constantes ou des données structurées de type arbres, listes ....

* les objets dynamiques, qui sont produits à l'exécution et permettent la mise en oeuvre de l'un des modes de contrôle décrits plus haut :
  - par disponibilité : paquets-résultat,
  - par nécessité : paquets-demande-d'évaluation d'expression et paquets-résultat.

Comme cela a été évoqué plus haut (§ 2.1.2.2), les informations manipulées sont distribuées sur l'ensemble des processeurs du réseau à l'aide d'une fonction de hachage portant sur le nom des objets (le nom d'une information est une entité dynamique identifiant un objet et son contexte) et la recherche d'une information particulière s'effectue selon le même procédé. Un objet n'est donc connu que par son nom et ce procédé conduit à mémoriser ensemble à la fois le nom d'un objet et l'objet identifié par ce nom. Pour effectuer les traitements, il est nécessaire d'associer à chaque objet des

informations supplémentaires (marques) qui précisent d'une part le type de l'objet (marques de type : expression, fonction, liste, ...), d'autre part l'état de l'objet (marques d'état : expression réduite, partiellement évaluée ou calculable par exemple). Un objet mémorisé dans un noeud du réseau comportera donc les informations suivantes :

- marques d'état ;
- nom de l'objet ;
- marques de type ;
- valeur de l'objet.

L'ensemble des objets étant distribué sur le réseau, une tentative d'accès à une information donnée prendra la forme d'une demande adressée au(x) noeud(s) détenant l'information recherchée. Cette demande sera effectuée sous la forme d'un paquet-requête expédié vers le noeud déterminé par l'application de la fonction de hachage au nom de l'objet cherché. Ce paquet comportera des informations précisant :

- le nom de l'objet recherché ;
- le nom de l'objet auquel il faudra retourner une éventuelle réponse ;
- des marques d'état précisant
  l'état du paquet (en transit, victime d'une collision, ...),
  son avenir une fois arrivé à destination (paquet à détruire, à réexpédier sur une branche de l'arbre de collision du noeud pour une information distribuée sur cette branche ou à diffuser sur tous les noeuds de l'arbre de recouvrement par exemple) ;
- le type du paquet.

Les paquets circulant dans le réseau peuvent être de trois types :

- les **paquets-initialisation** véhiculent une information à stocker dans le noeud destinataire, ils ne nécessitent pas de réponse ;
- les **paquets-requête** recherchent une information donnée ;
- les **paquets-résultat** véhiculent les réponses aux requêtes antérieures.

Dans tous ces cas, une **communication par messages** peut être mise en oeuvre aussi bien à l'intérieur d'un noeud entre les différentes unités fonctionnelles qu'entre les différents noeuds du réseau. Dans ce dernier cas, les messages jouent le rôle des paquets définis ci-dessus.

2.1.6 Structure d'un noeud

Chaque noeud du réseau assure deux fonctions : traitement et communications. Le parallélisme envisagé dans le projet N-ARCH étant de granularité très fine, les communications joueront un rôle très important : elles ne devront pas ralentir le traitement

sinon de manière insensible. De fait, elles vont être réalisées de manière indépendante par rapport au traitement : chaque noeud comporte des outils de communication, de traitement, de contrôle et de mémoire fonctionnant en parallèle et de manière asynchrone. La figure 2.3 ci-dessous représente la structure générale d'un noeud du réseau.

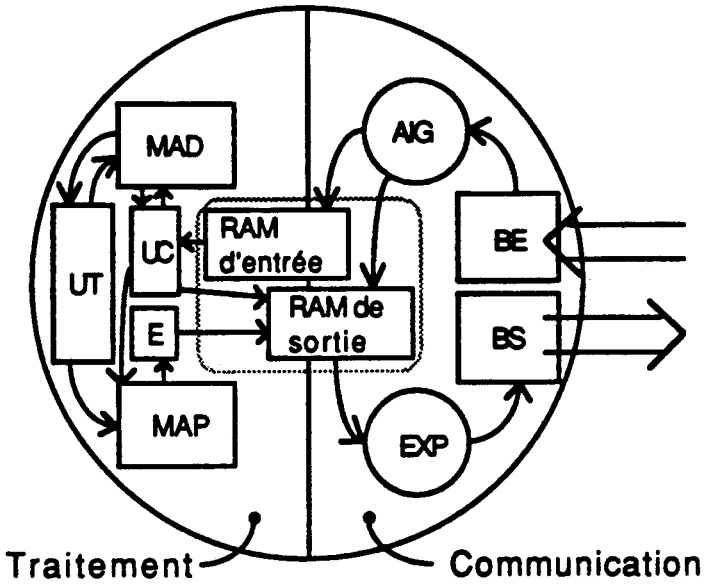


figure 2.3

On voit sur cette figure que le noeud peut en fait être considéré comme constitué de deux parties indépendantes : l'une consacrée au traitement, l'autre aux communications. Les seuls liens entre ces deux parties sont les deux mémoires RAM qui sont accessibles à la fois par le traitement et les communications. Le fonctionnement des différentes unités de chacune des parties sera détaillé dans les deux paragraphes suivants.

2.1.6.1 Partie communication

Dans le réseau, les messages ou paquets transitent de noeud en noeud jusqu'à leur destinataire. Pour chaque noeud, la partie consacrée aux communications a la structure représentée ci-dessous (fig. 2.4) :

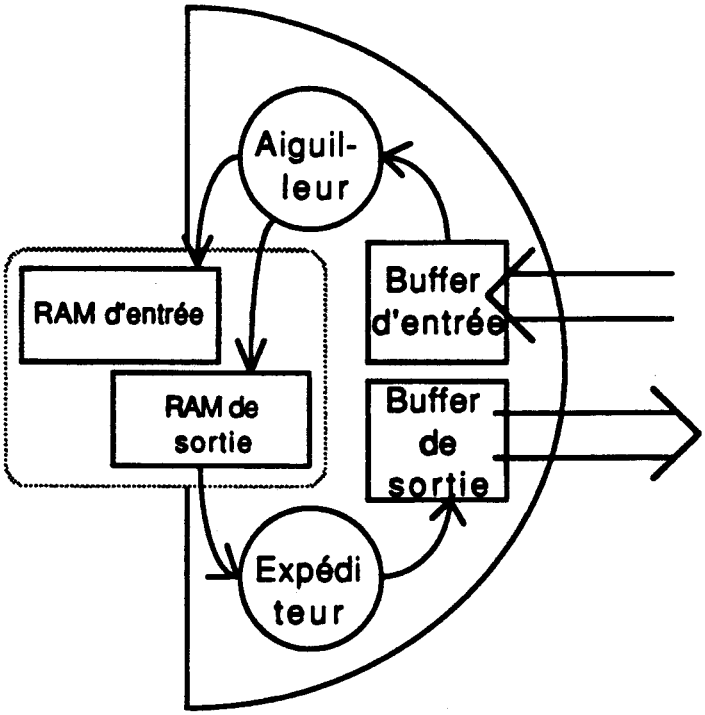


figure 2.4

Le fonctionnement de cette partie communication est le suivant : en entrée les paquets provenant de l'extérieur ou des noeuds voisins sont reçus dans des buffers d'entrée. Ils sont examinés par un aiguilleur qui les dirige vers l'une des mémoires RAM en fonction de leur destinataire :

* si ce dernier est précisément le noeud où le paquet vient de parvenir, celui-ci sera dirigé vers la RAM d'entrée en vue d'un traitement ultérieur par l'unité de contrôle ;

* dans le cas contraire, il s'agit d'un paquet en transit qui sera donc stocké en RAM de sortie en vue d'une réexpédition vers un noeud successeur.

2.1.6.2 Partie traitement

Les organes internes au noeud sont organisés en unités séparées (voir figure 2.5), fonctionnant de manière asynchrone et communiquant par l'intermédiaire de mémoires associatives spécialisées :

la mémoire associative des définitions (MAD) contient le programme : expressions, fonctions, données ;

la mémoire associative des paquets (MAP) contient les paquets générés par le noeud et en attente d'une information.

Ces mémoires sont accessibles par les unités fonctionnelles de manière associative, le critère de recherche concernant le nom de l'objet cherché et/ou des bits d'état. Ces bits d'états sont les suivants :

- pour une expression :

à réduire (AR)

en cours de réduction (CR)

évaluable (EV)

en cours d'évaluation (CE)

réduite (RE)

- pour un paquet :

à expédier (EX)

en attente (AT)

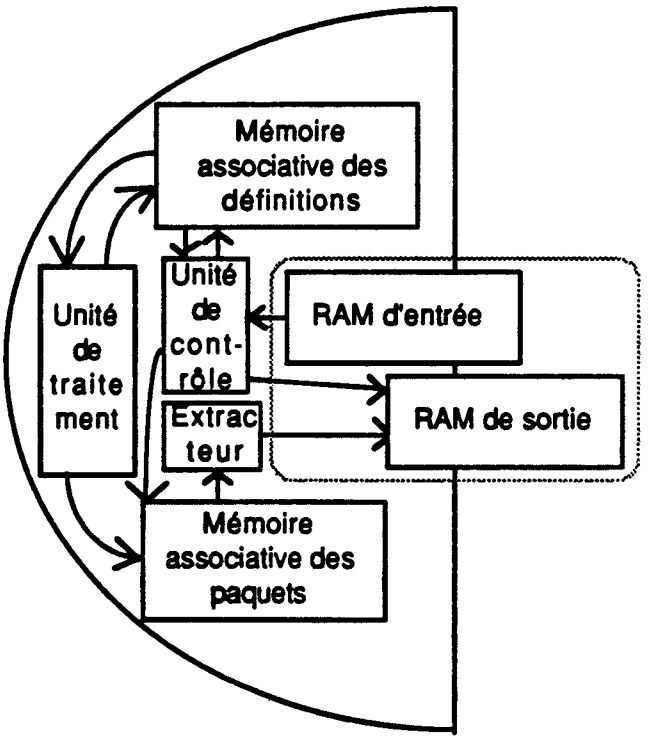


figure 2.5

L'unité de traitement (éventuellement composée de plusieurs unités parallèles) examine de manière cyclique la mémoire associative des définitions pour en extraire les opérations à effectuer. Cet examen est réalisé en testant un bit d'état particulier (EV), associé à chaque expression, qui indique si celle-ci est calculable. L'unité de traitement lit une telle expression et positionne en MAD un autre bit d'état (CE) signalant que le calcul de l'expression est en cours. Une fois l'expression réduite, l'unité de traitement la recopie dans la MAD (accès par le nom) en modifiant les bits d'état : l'expression passe à l'état réduite (bit d'état RE). Cette valeur est également recopiée dans chaque paquet qui

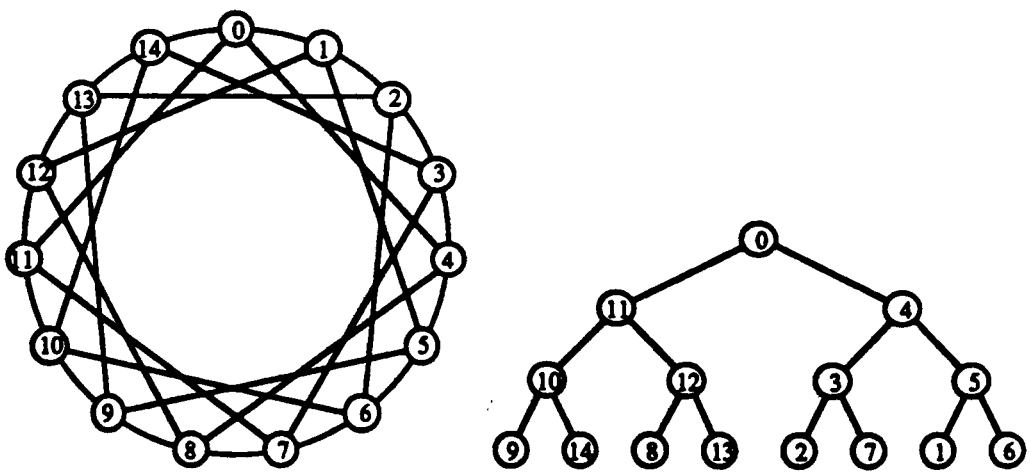
l'attend dans la MAP et tous les paquets ainsi mis à jour sont marqués "prêt à être expédiés" grâce au bit d'état EX.

Ce dernier permet à l'**extracteur** de repérer puis extraire les paquets à expédier de la MAP et de les déposer dans la mémoire de sortie.

L'unité principale d'un noeud est l'**unité de contrôle**, qui gère le mécanisme de réduction et génère les paquets. L'unité de contrôle extrait de la mémoire d'entrée un paquet qui sera de l'un des trois types vus plus haut (§ 2.1.4) : initialisation, requête ou résultat. Les opérations effectuées par l'unité de contrôle dépendent du type du paquet traité : pour un paquet-initialisation, l'unité de contrôle effectuera une écriture en MAD, sous réserve qu'il existe un emplacement disponible, tandis que pour un paquet-requête ou résultat c'est une lecture qui sera effectuée en MAD. Si l'écriture est impossible ou si la lecture est un échec, c'est-à-dire si l'information cherchée n'est pas présente, on a affaire à un phénomène de collision et le paquet traité est alors replacé en mémoire de sortie pour être expédié vers un autre noeud, déterminé par un calcul de la fonction de hachage local. Un paquet-requête ou résultat peut également être réexpédié vers un ou plusieurs autres noeuds quand il concerne une information dispersée sur plusieurs noeuds.

2.2 Exemple d'évaluation

Supposons que le réseau utilisé soit celui de la figure 2.6.a, donné avec son arbre de recouvrement (fig. 2.6.b) de racine le noeud 0.



.a .b

figure 2.6

Un arbre de recouvrement de racine le noeud n se déduit de celui de la figure 2.6.b en ajoutant n MODULO 15 à tous les numéros de noeud du recouvrement ci-dessus.

Soit l'extrait de programme suivant, qui demande la valeur de X pour le calcul d'une certaine expression W :

W = (... X ...)
X = (*, U, V)
U = (+, Y, Z)
V = (-, Y, Z)
Y = (2)
Z = (10)

Supposons enfin que le hachage ait réparti de la manière suivante les définitions :

Définition	Hachage global	Hachage(s) local(ux) éventuel(s)
U	noeud 3	
V	noeud 14	
X	noeud 3 → (collision) noeud 14 → (collision) noeud 0	
Y	noeud 5	
Z	noeud 14	

Tableau 2.1

L'arbre de recouvrement de racine 3 étant :

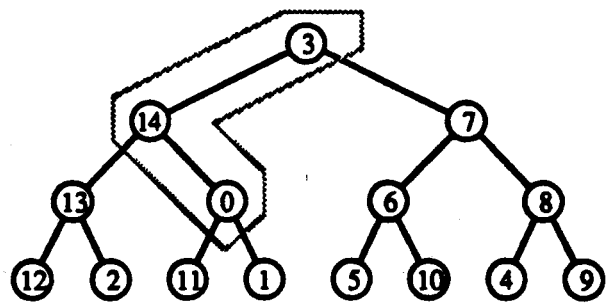


figure 2.7

Répartition des définitions :

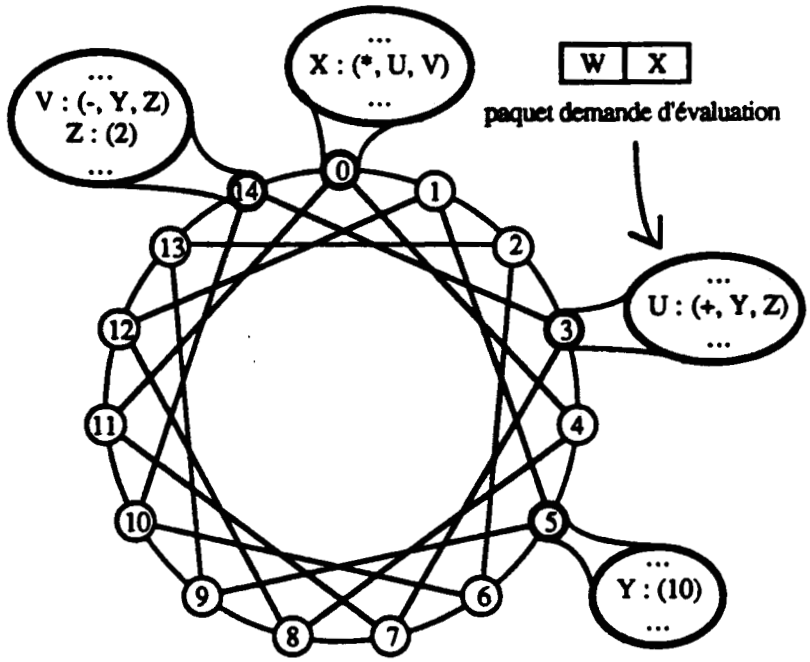


figure 2.8

Après application du hachage global, le paquet-demande-d'évaluation de X pour W est présenté au noeud 3. Puisqu'il y a eu collision au moment du rangement de X, la recherche de X dans la mémoire associative du noeud 3 échoue. Elle reste également infructueuse après une première application du hachage local, qui aura fait explorer le noeud 14. En revanche, une deuxième application du hachage local permet d'accéder à X dans le noeud 0.

--> Si X est une expression irréductible, c'est-à-dire la valeur de X est disponible, cette valeur est retournée à W au moyen d'un paquet-résultat par l'intermédiaire du réseau (routage direct puis, éventuellement, routage arborescent).

--> Dans le cas contraire, plusieurs possibilités sont envisageables.

---->Si l'on travaille en mode *réduction de graphe*, le noeud 0 va effectuer les actions suivantes :

a/ Génération et stockage en mémoire associative d'un paquet en attente de la valeur de X dont le destinataire sera W ;

b/ Envoi dans le réseau de deux paquets de demande d'évaluation concernant respectivement les valeurs de U et V pour X.

Ultérieurement, toute nouvelle demande de la valeur de X ne déclencherait que l'exécution de la seule action a/, sans envoi de demande de valeur pour U et V.

Lorsque les valeurs de U et V seront retournées au noeud 0, elles seront stockées dans la définition de X. X sera alors détecté comme étant réductible par l'extracteur qui scrute en permanence la mémoire associative, puis sera évaluée. Enfin, la valeur de X sera stockée dans toutes les cellules de la mémoire associative qui sont en attente de la valeur de X. L'extracteur du noeud détectera alors les paquets-résultat à émettre et les enverra à leurs destinataire par le réseau.

----> Si l'on travaille en mode *réduction de chaîne*, le noeud 0 retourne simplement, à destination de W, un paquet-définition de X. Une utilisation systématique de ce mécanisme conduirait à un regroupement de toutes les expressions sur le noeud détenant W et donc à une exécution locale du programme uniquement sur ce noeud.

Un mode mixte de commande peut également être réalisé : le noeud 0 pourrait par exemple envoyer des demandes de U et V à retourner directement à W, retourner un paquet-définition de X pour W et même déposer des demandes d'évaluation de U et V pour X de manière à anticiper le calcul de X pour d'éventuelles futures demandes.

On peut ainsi utiliser un mode de commande mixte en utilisant la réduction de chaîne ou de graphe selon que l'on traite une expression courte ou une fonction longue ou bien une donnée simple ou une structure de données complexe.

Une évaluation dirigée par les données peut également être implantée en associant à chaque expression les noms des consommateurs de son résultat.

2.3 Applications développées dans le cadre du projet

2.3.1 Un nouveau schéma d'évaluation de FP

L'un des objectifs initiaux du projet N-ARCH était que l'architecture envisagée soit adaptée à l'exécution de programmes déclaratifs. Pour se faire une idée de ces possibilités, il a été décidé d'essayer d'implémenter sur la machine N-ARCH un langage fonctionnel, en essayant évidemment d'en exploiter le parallélisme.

De nombreux langages fonctionnels ont été définis au cours d'une période relativement récente ; la plupart d'entre eux sont généralement implémentés au moyen de combinateurs de Turner [TURN79] ou de super-combinateurs [HUGH82].

Pour le projet N-ARCH, le langage choisi a été FP, un langage défini par J. BACKUS. Les recherches à ce sujet ont été effectuées par N. DEVESA [DEVE89], et ont permis de proposer un nouveau schéma d'évaluation de FP : au lieu d'utiliser la réduction de chaîne ou la réduction de graphe, ce nouveau schéma fait appel à la fois à la réduction de chaîne et de graphe, en transmettant des opérations d'un graphe de contrôle à un graphe de réduction parallèle.

Dans ce modèle, un programme FP, après découpage en sous-programmes pouvant présenter des relations de dépendance, est représenté par un ensemble d'arborescences fonctionnelles.

L'exécution consiste en un parcours et une analyse simultanées de ces arborescences afin d'amener les fonctions primitives vers leurs arguments, au contraire de ce qui est fait classiquement, c'est-à-dire amener les opérandes vers l'opérateur. Les raisons de ce choix, ainsi que les règles d'exploration des arborescences, sont exposées dans [DEVE89].

La figure ci-dessous schématise ce principe d'exécution :

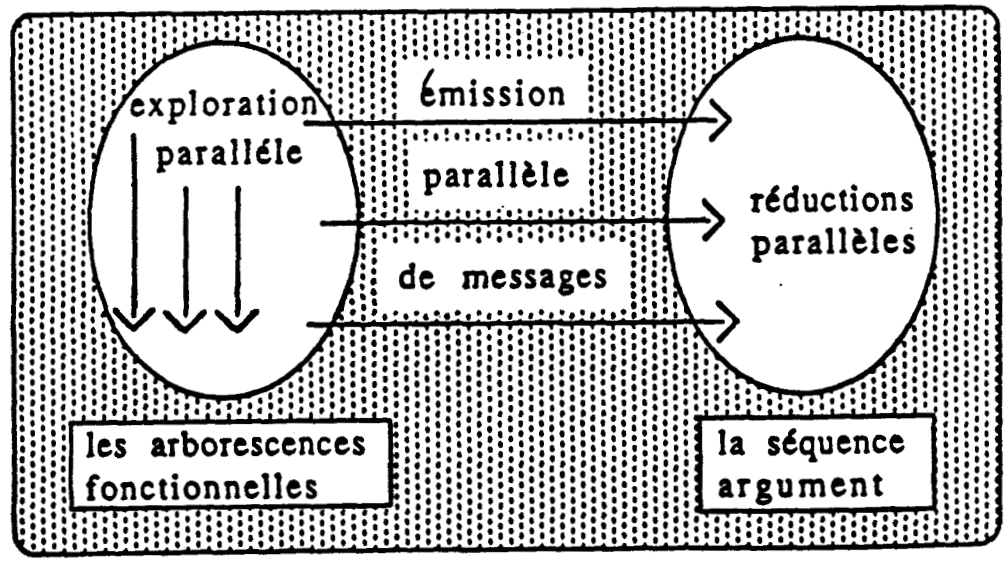


figure 2.9

Dans ce modèle, contrôle et commande se font donc au travers des arborescences fonctionnelles et de leurs règles d'exploration.

Ces arborescences forment un graphe de contrôle statique, alors que les arguments constituent un graphe de réduction parallèle dynamique.

2.3.2 Hachage orienté bases de données relationnelles

Les bases de données constituent un domaine où de nombreux traitements sont parallélisables. Ici aussi, l'efficacité de cette parallélisation va dépendre pour beaucoup de la bonne répartition des données. C'est donc la fonction de hachage qui est particulièrement importante. Les travaux de J.C. NICOLAS sur ce sujet lui ont permis de proposer des méthodes de répartition concernant les bases de données.

L'une de ces méthodes [NICO89a] permet de répartir des tuples de relations en fonction des attributs de jointure. J.C. Nicolas décrit également l'utilisation dynamique de cette méthode par un algorithme de jointure.

Une autre méthode [NICO89b] concerne la répartition de données non nécessairement sous première forme normale (N1NF : Non First Normal Form). Il s'agit donc de modèle de bases de données plus général que les bases de données relationnelles. J.C. Nicolas utilise pour cette étude le modèle de données lié au langage FAD développé au MCC Austin Texas.

J.C. Nicolas a également entreprit l'implantation des primitives d'un langage fonctionnel de manipulation.

2.3.3 Mémoire associative

Pour les besoins du projet N-ARCH, un nouveau type de mémoire associative a dû être défini, et cela pour deux raisons principales :

i) la faible capacité d'intégration des mémoires associatives. C'est l'une des raisons du recours à une technique associative à deux niveaux, par "distribution" de la mémoire sur tout le réseau et utilisation du hachage. Ceci est développé par ailleurs.

ii) le fait que, vu le fonctionnement de la partie traitement des noeuds (§ 2.1.6.2), l'accès aux mémoires associatives est partagé entre plusieurs unités : unité de contrôle, unité de traitement, extracteur. Il a donc fallu définir une mémoire associative permettant des accès parallèles, de manière à contre balancer le surcroît de communications introduit par la distribution envisagée.

2.3.3.1 Fonctionnement

Cette mémoire doit assurer deux fonctions :

- mémoriser des objets auxquels pourront accéder plusieurs unités ;
- gérer les bits d'états permettant la mise en oeuvre du mode de contrôle envisagé.

Trois types de requêtes peuvent lui être adressées :

- *lecture* ;
- *écriture* ;
- *mise à jour.*

Pour satisfaire ces demandes, on dispose de trois actions primitives :

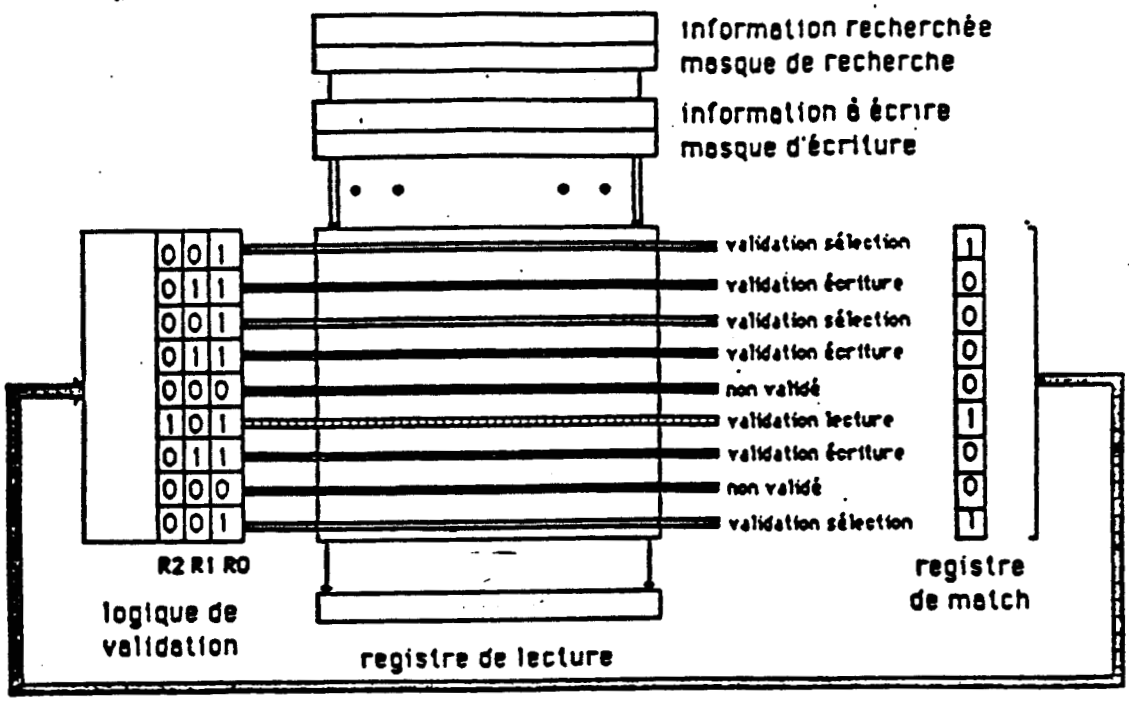
- la **sélection** : comparaison, en parallèle, d'une information avec tous les mots non vides de la mémoire. Le résultat de cette action est retourné dans un "registre de match". Un registre-masque peut être utilisé au cours de cette action afin de ne faire porter la comparaison que sur une partie de mot ;
- l'**écriture** : mise à jour d'une information dans tous les mots mémoire préalablement sélectionnés en vue de cette action. Ici aussi, un registre-masque peut être utilisé ;
- la **lecture** : un mot mémoire, préalablement sélectionné, est copié dans un registre de lecture.

Le respect de règles simples permet de conserver la validité des informations tout en réalisant certaines actions en parallèle :

- ne pas sélectionner un même mot dans un même cycle mémoire à la fois pour une écriture et une lecture ;
- une sélection et une écriture ne peuvent être faite en parallèle sur un même mot mémoire que si le masque de sélection et le masque d'écriture sont disjoints.

Par exemple, sur la figure 2.10, si les masques de sélection et d'écriture sont disjoints, on peut réaliser simultanément :

- une opération de lecture,
- une écriture sur 3 mots simultanément,
- une sélection sur toutes les lignes validées.



R0 : bit = 1 (mot occupé)
bit = 0 (mot libre)
R1 : bit = 1 (mot à écrire)
R2 : bit = 1 (mot à lire)

figure 2.10

Un registre de match reçoit les résultats des comparaisons et est ensuite recopié dans les registres de validation pour les lectures ou écritures ultérieures.

De plus, des registres de copie de match, à raison d'un par unité pouvant accéder à la mémoire en lecture, permettent un entrelacement de plusieurs requêtes de lecture simultanées.

Les requêtes peuvent alors être traduites de la manière suivante :

lecture = sélection + lecture(s) ;
mise à jour = sélection + écriture(s parallèles) ;
écriture = sélection + écriture unique.

2.3.3.2 Implémentation

L'étude préliminaire d'une implémentation VLSI de cette mémoire a été réalisée grâce au laboratoire de micro-électronique de l'ISEN (Institut Supérieur d'Electronique du Nord). Le résultat est une cellule mémoire élémentaire à 5Y+3X connections dont une représentation figure ci-dessous :

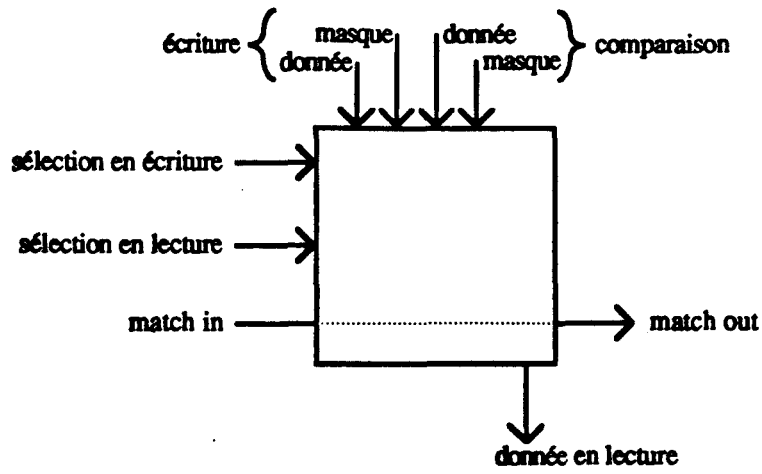


figure 2.11

Une description fonctionnelle totale de cette mémoire associative a été effectuée à l'aide du langage de description LIDO [BAKO87] et doit maintenant être suivie de la réalisation du circuit VLSI.

2.3.4 LogArch

Plutôt qu'une application dans le cadre du projet N-ARCH, LogArch en constitue une nouvelle orientation, plus axée, comme son nom l'indique, sur la programmation logique et plus particulièrement le langage PROLOG. Un nouveau schéma d'évaluation parallèle de ce langage a été élaboré par I. Hannequin [HANN89].

2.3.4.1 Le schéma d'évaluation

Les deux objectifs principaux de ce schéma sont :

* conserver la sémantique opérationnelle du langage PROLOG,
* en exploiter le parallélisme implicite sans aucune intervention du programmeur (pas de modification ni d'extension à la syntaxe).

Ce schéma d'évaluation utilise un arbre ET/OU de résolution élaboré dynamiquement. Le parallélisme OU y est partiellement exploité, tandis que le parallélisme ET l'est sous forme de pipeline.

L'arbre est construit de manière descendante et, lorsqu'un processus ET engendre plusieurs processus OU, seul le processus le plus à gauche est activé. Lorsque ce dernier retourne une première solution, le processus père active le processus OU suivant. De plus, pour conserver l'ordre d'obtention des solutions comme en PROLOG séquentiel, un mécanisme de blocage est utilisé, qui empêche la remontée immédiate des solutions du deuxième processus OU vers le processus père. Les solutions élaborées par le deuxième

processus ne seront renvoyées que sur déblocage du processus par le processus père. La remontée des solutions se fait donc selon un mécanisme demand-driven. Le même mécanisme se reproduit pour les autres processus OU. On voit donc en quel sens le parallélisme OU est partiellement limité : (1) sur un même niveau de l'arbre de résolution, tous les processus OU ne sont pas activés simultanément, (2) le blocage de leurs solutions n'est levé que progressivement de gauche à droite. Le nombre de branches "activées-bloquées" peut être modifié suivant la charge du système. Grâce à ces limitations dans l'exploitation du parallélisme OU et au mécanisme demand-driven, la gestion des environnements est simplifiée et on prévient l'explosion combinatoire du nombre de processus actifs, deux inconvénients principaux d'une parallélisation systématique des processus OU.

Le parallélisme ET pipeliné permet de résoudre le problème des variables partagées. Il utilise le même mécanisme d'activation bloquée que ci-dessus. Lorsqu'un processus OU génère plusieurs processus ET, le processus fils le plus à gauche est tout d'abord activé ; puis, dès le retour d'une première solution de ce dernier vers le processus père, celui-ci bloque le premier fils (qui ne retourne plus de solutions mais reste tout de même actif) et active le deuxième. Le premier ne sera débloqué que sur demande du processus père. La même opération se reproduit pour tous les autres processus ET.

2.3.4.2 Implémentation

Le programme PROLOG est complètement distribué sur le réseau, la clause jouant le rôle de particule élémentaire dans ce fractionnement. On trouvera donc sur un même noeud du réseau une clause entière, c'est-à-dire un noeud OU de l'arbre de résolution et ses noeuds-fils ET.

La structure d'un noeud du réseau est légèrement modifiée par rapport à ce qui a été défini au paragraphe 2.1.6.2, afin de prendre en compte certaines spécificités de PROLOG. On remarque en particulier, sur la figure 2.12, une partie spécialisée prenant en charge l'unification. De plus, des messages particuliers sont introduits : messages d'activation, de blocage, de déblocage.

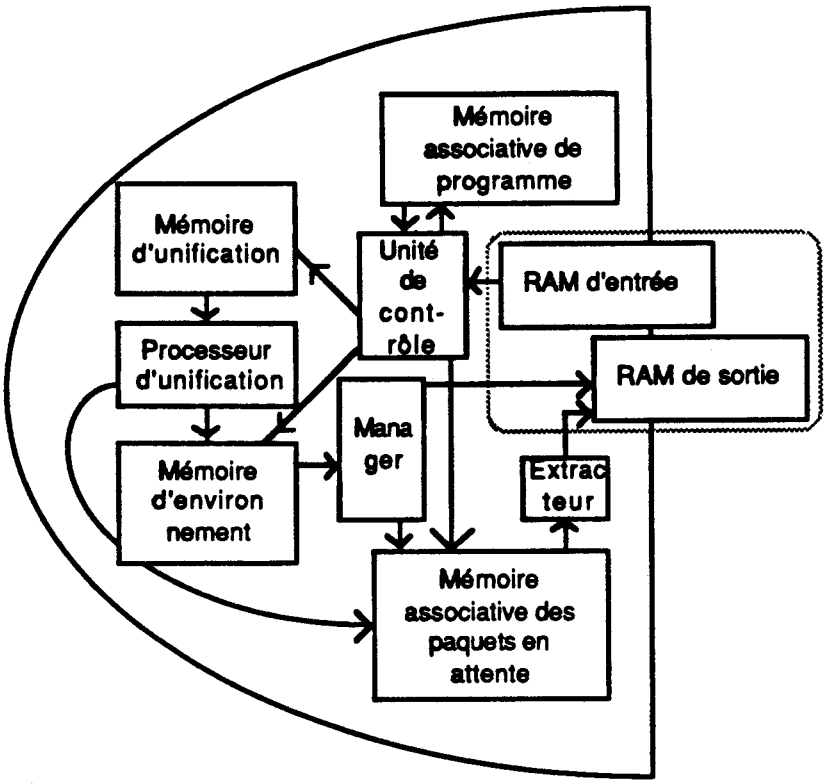

Mémoire associative de programme
Mémoire d'unification
Unité de cont-rôle
RAM d'entrée
Processeur d'unification
RAM de sortie
Mana ger
Mémoire d'environ nement
Extrac teur
Mémoire associative des paquets en attente

figure 2.12

CHAPITRE 3
DISTRIBUTION PAR HACHAGE.

Comme on l'a vu au chapitre 1, l'un des principaux problèmes qui se posent dans les machines distribuées réside dans la répartition des données et des traitements. Pour le projet N-ARCH, il a été décidé de répartir sur l'ensemble des processeurs tous les objets, qu'il soient statiques ou dynamiques, cette répartition faisant appel à des techniques de hachage.

3.1 Hachage

Le hachage, ou adressage dispersé, intervient lorsque l'on doit ranger, puis ultérieurement rechercher, un ensemble d'items dans un ensemble d'emplacements. Etant donné un item, le hachage calcule sa position au lieu de faire une recherche exhaustive. En règle générale, on peut placer un item par emplacement et on dispose de plus d'emplacements qu'il n'y a d'items. Le hachage consiste à calculer, à partir d'un item I auquel on associe une clé C, une fonction de hachage h(C) qui va donner un emplacement où ranger (ou chercher) l'item I. L'idéal serait évidemment que pour deux items différents I et I', h(C) et h(C') soient toujours différents. Ce n'est pas le cas en général et lorsque 2 ou plusieurs items sont "envoyés" par la fonction de hachage sur un même emplacement on dit qu'il y a collision. Il existe de nombreuses techniques de hachages [MORR68, BELL70, LUM&71, KNUT73], dont quelques unes parmi les plus classiques sont rappelées ci-dessous.

3.1.1 Fonctions de hachage

Généralement le problème est le suivant : étant donné une clé C, associée au contenu d'un item I, on désire identifier cette clé à une adresse dans une table. Si l'item est déjà dans la table, alors l'adresse, appelée adresse de hachage, sera celle de son emplacement, sinon ce sera celle d'un emplacement vide. Pour obtenir l'adresse de hachage à partir de la clé on peut procéder par

- **division** de la clé par un entier, en général premier, proche du nombre d'emplacements. Le reste de la division donne l'adresse.

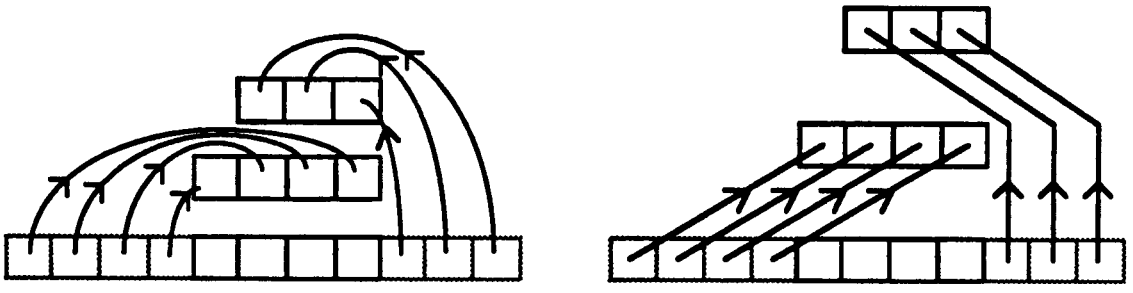
- **analyse des clés** : on élimine dans les clés les bits ou séquences de bits dont la distribution est déséquilibrée, on utilise les bits restants pour obtenir l'adresse de hachage. L'élimination porte sur les mêmes bits dans toutes les clés.

- **extraction de bits** :

* si la clé est codée sur un mot machine, élévation au carré de la clé puis choix de quelques bits au milieu de ce carré pour donner l'adresse (middle square). Le nombre de bits à retenir dépend du nombre de bits codant l'adresse. L'élévation au carré qui précède l'extraction des bits est destinée à éliminer les collisions dues à l'apparition d'une même séquence de bits dans de nombreuses clés .

* si la clé est codée sur plusieurs mots, l'élévation au carré est remplacée par le produit des différents mots puis on effectue un choix de bits dans le produit.

- **pliage** : on coupe la clé en plusieurs séquences de bits qu'on ajoute, puis on garde des bits de la somme. La figure ci-dessous présente deux façons de "plier" la clé.



.a .b

figure 3.1

- **changement de base** : on choisit trois nombres p, q et m , p et q premiers entre eux. La clé étant représentée en binaire, ses bits sont groupés de façon à ce que chaque groupe puisse être considéré comme la représentation d'un chiffre en base p. Le nombre obtenu est traduit en un nombre en base dix dont on prend le reste modulo q^m comme adresse de hachage.

Exemple : Supposons que la clé soit C = 975 et le nombre d'adresses 48. Si on prend pour p, q, m les valeurs suivantes : p = 8, q = 7, m = 2, l'adresse sera calculée de la manière suivante :

975 ---> 100101110101 (codage DCB) ;
en groupant les bits par 3, on obtient 4565 en base 8 ;
la traduction en base 10 donne 2421;
et enfin le reste modulo 49 donne l'adresse : 20.

- **codage polynomial** : chaque chiffre de la clé est considéré comme un coefficient d'un polynôme. Le polynôme ainsi obtenu est divisé par un polynôme P, le même pour toutes les clés. Les coefficients du polynôme reste de la division donnent la clé. Dans cette méthode, le choix du polynôme P est important, voir par exemple à ce propos [LUM&71].

3.1.2 Méthodes de résolution des collisions

Assez souvent, même après un choix bien adapté de la fonction de hachage, des collisions se produisent. On doit donc créer une suite d'adresses $h_i(C)$ (i = 1, 2, 3, ...) à essayer jusqu'à satisfaction. Pour minimiser le coût de génération de cette suite, il est nécessaire de raccourcir le temps de calcul de chaque h_i ou de raccourcir la longueur moyenne de la suite.

La méthode la plus simple consiste à calculer : $h_i(C) = h_1(C) + D(i)$, où D représente un déplacement. Ce déplacement peut être :

linéaire : D(i) = i ; D(i) = i.c. Cette méthode est peu rapide car elle amène la création de "grappes". La constante c doit être choisie de façon à ce que l'on puisse parcourir toute la table si nécessaire.

aléatoire : $D(i) = r_i$, où (r_i) désigne une suite de nombres aléatoires. Cette méthode est meilleure en nombre d'essais que la précédente.

quadratique : $D(i) = a.i+b.i^2$. Cette méthode est équivalente à la précédente en nombre d'essais, mais plus rapide du point de vue des calculs.

On peut citer également d'autres méthodes :

- **chaînage** : une partie de l'emplacement est réservée à un pointeur vers le prochain emplacement à essayer.

- **espace de débordement** séparé ou non : une zone particulière, faisant partie de la table ou disjointe de celle-ci, est réservée au stockage des items en collision.

3.2 Cas du projet N-ARCH

Dans le cas du projet N-ARCH, le réseau peut être vu comme une immense mémoire à deux niveaux : un niveau global, pour lequel les processeurs constituent des

"macro-emplacements", et un niveau local, pour lequel les emplacements sont les cellules des mémoires associatives présentes en chaque processeur.

· Le hachage "global" peut donc sans problème attribuer un même processeur à plusieurs items. La seule limitation réside dans la saturation des capacités mémoire du noeud. C'est dans ce cas seulement que l'on parlera de collision.

En résumé, étant donné un objet à ranger ou rechercher,

a) un hachage global détermine le noeud où il devrait se trouver ;

b) à l'intérieur de ce noeud, une recherche associative détermine son emplacement.

Si cette recherche s'avère infructueuse, alors il y a collision.

C'est au niveau du traitement de ces collisions que la méthode de hachage utilisée dans le projet N-ARCH est particulière.

3.2.1 Inadéquation des techniques habituelles

S'agissant d'une architecture distribuée, les techniques de hachage habituelles peuvent-elles être utilisées dans l'organisation et la répartition des objets?

Pour mettre en place un adressage dispersé le concepteur se trouve devant un double choix :

1) choisir une fonction de hachage (soit F cette fonction) ;

2) choisir une méthode de résolution des collisions (soit M cette méthode).

Classiquement - par exemple pour un rangement ou une recherche dans une table en mémoire, c'est-à-dire un traitement essentiellement local - un objet étant donné, F fournit son emplacement présumé dans la table. Puis, à partir de cet emplacement, lors de chaque collision, M fournit un nouvel emplacement, i.e. les applications successives de M fournissent une suite d'emplacements à examiner séquentiellement, dans l'un desquels pourra être satisfaite la demande.

Pour une architecture distribuée comme celle envisagée dans le projet N-ARCH, où les communications se font par messages entre les différents noeuds d'un réseau, lorsque l'un des noeuds doit ranger une valeur, retourner un résultat ou demander une valeur, il émet un message à destination d'un noeud où sa demande devrait être satisfaite.

On pourrait donc imaginer, par analogie avec le fonctionnement de la technique classique rappelé ci-dessus, que F fournisse un numéro de noeud, vers lequel le message va se diriger tout d'abord ; puis, en cas de collision, que M fournisse un nouveau numéro

de noeud. Afin d'éviter des délais de routage trop importants, cette solution n'est intéressante que si le noeud fourni par M est physiquement rattaché au noeud, fourni par F ou par la précédente application de M, où se produit une collision. Evidemment, aucune application de M ne doit fournir le même noeud que l'une des applications précédentes ayant déjà amené une collision. En conséquence, les noeuds successifs fournis par M à partir du noeud défini par F doivent correspondre, dans le graphe modélisant le réseau, à des sommets formant une chaîne élémentaire ou même une chaîne hamiltonienne. En cas de saturation d'un noeud, cette saturation va donc se propager le long de la chaîne, créant un phénomène de bouchon, allongeant ainsi le temps d'acheminement du message jusqu'au noeud où il pourra être traité.

Pour pallier ce problème, une méthode originale a été définie pour le réseau N-ARCH, cette méthode permettant de répartir les demandes redirigées, à partir d'un noeud n_0, sur l'ensemble ou un sous-ensemble des noeuds voisins de n_0.

3.2.2 Technique originale

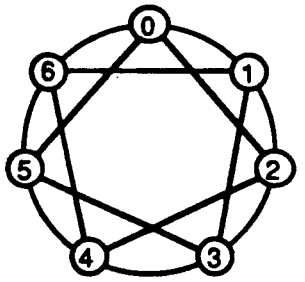
Cette méthode fait appel soit à des arbres de recouvrement, soit à des arbres d'exploration de graphe.

3.2.2.1 Méthode avec arbre d'exploration de graphe

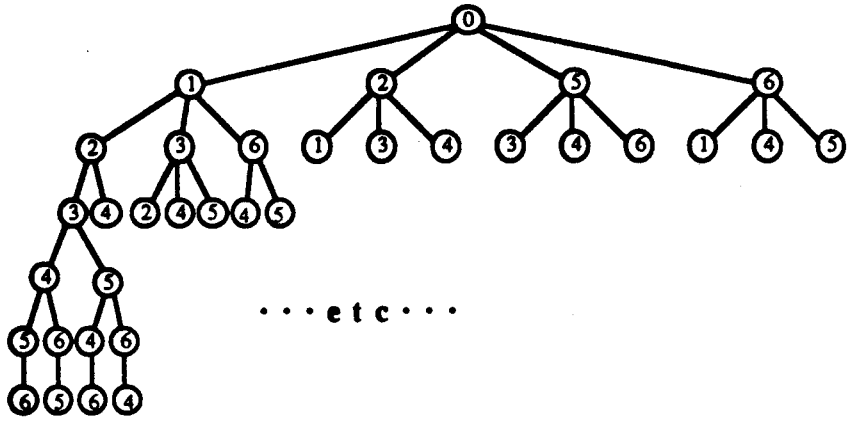
Dans un arbre d'exploration de graphe, chaque branche est un chemin halmiltonien du graphe, de point de départ le noeud racine de l'arbre.

Cette méthode, appliquée aux réseaux en hypercube, a fait l'objet d'un mémoire de DEA [BOUR88] dans le cadre du projet N-ARCH. Lorsqu'une collision se produit sur un noeud n_0, M consiste à choisir, parmi toutes les chaînes hamiltoniennes passant par le noeud n_0, une chaîne qui sera parcourue progressivement au fur et à mesure des redirections. Ici les messages redirigés à partir de n_0 sont donc répartis sur toutes les chaînes hamiltoniennes contenant n_0, qui sont en nombre assez élevé.

Exemple : La figure ci-dessous représente un graphe à 7 noeuds (3.2.a) ainsi qu'une partie d'un arbre d'exploration de ce graphe, de racine le noeud 0 (3.2.b).



.a



.b

figure 3.2

3.2.2.2 Méthode avec arbre de recouvrement

C'est celle qui a été retenue pour le présent travail. Un arbre de recouvrement est tel que :

* les noeuds de l'arbre sont en nombre égal aux noeuds du graphe ;
* chaque noeud ne figure qu'une seule fois dans l'arbre ;
* pour un noeud donné n_0, ses fils dans l'arbre sont voisins de n_0 dans le graphe.

Cette méthode consiste à considérer

1) que dans le réseau d'une machine multiprocesseurs chaque noeud du graphe est la racine d'un arbre de recouvrement ;

2) M consiste en une *fonction de hachage locale* qui détermine, dans l'arbre de recouvrement, parmi les fils du noeud où se produit une collision, celui vers lequel va être redirigée la demande.

Ainsi, à partir d'un noeud donné n_0, les redirections ne se font plus systématiquement sur un même noeud mais sont réparties sur d noeuds, d étant le demi-degré

extérieur (c'est-à-dire le nombre de fils) du noeud n_0 dans l'arbre de recouvrement. Ceci permet d'éviter le phénomène de bouchon décrit plus haut.

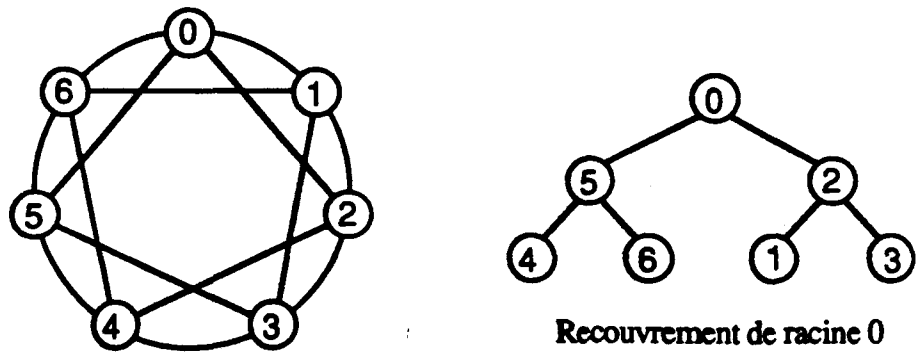
Un avantage de la méthode par arbre d'exploration de graphe par rapport à la résolution des collisions par l'intermédiaire d'un arbre de recouvrement réside dans le nombre de collisions supportable par un message donné : dans le cas de la première, un message redirigé pourrait éventuellement passer par tous les noeuds du graphe. Dans le cadre d'une redirection par arbre de recouvrement en revanche, le nombre de noeuds parcourus par le message redirigé est limité par la profondeur de l'arbre de recouvrement. Il faudra donc que le degré de l'arbre de recouvrement ne soit pas trop grand de façon à ce que sa profondeur soit suffisante pour garantir que le réseau supporte un nombre minimal raisonnable de collisions successives . En revanche, les arbres à manipuler sont nettement plus importants dans le cas de l'exploration de graphe que dans celui des arbres de recouvrement, comme on peut le voir en comparant les figures 3.2.b et 3.3.b.

3.3 Conséquences sur la topologie

Cette méthode particulière introduit des contraintes au niveau de l'arbre de recouvrement : celui-ci doit, si possible, être régulier et complet.

Par suite, le graphe modélisant le réseau doit être tel que chaque sommet soit racine d'un arbre de recouvrement obéissant à la contrainte ci-dessus.

Exemple : La figure ci-dessous représente un réseau à 7 noeuds de degré d = 4, de diamètre D = 2, et dont chaque noeud est racine d'un arbre binaire de recouvrement.



.a .b

figure 3.3

On obtient l'arbre de recouvrement de racine n (n = 1, 2, ..., 6) en ajoutant n MODULO 7 à chaque numéro de noeud dans l'arbre de recouvrement de racine 0.

Bien évidemment, en sus des contraintes introduites pour les besoins de la technique de distribution utilisée, les contraintes ordinairement rencontrées lors de la définition de tout réseau d'interconnexion restent valables : le graphe modélisant le réseau doit être régulier, en particulier pour des raisons d'utilisation de technologies VLSI ; il faut arriver à un bon compromis entre les divers paramètres - plus ou moins contradictoires - que sont le nombre de noeuds, le degré de ces noeuds (et donc le nombre d'arètes dans le graphe), le diamètre du graphe, le coût de réalisation ...

Obtenir un gain en jouant sur l'un des paramètres implique généralement une perte sur un autre. L'un des exemples les plus évidents est le suivant : en vue d'augmenter la rapidité des communications et leur fiabilité, on peut augmenter le nombre de liaisons entre les processeurs dans le réseau d'où davantage de chemins dans le graphe modélisant le réseau, davantage de "raccourcis", possibilité de "court-circuiter" un noeud ou une liaison momentanément ou définitivement hors d'usage, et donc la communication d'un point à un autre se fait plus rapidement. En contre-partie, le nombre de pattes d'entrée/sortie sur chaque processeur élémentaire augmente et avec lui le coût de construction.

Jusqu'à présent, de nombreux réseaux d'interconnexion ont été proposés ou étudiés, en prenant en compte ces dernières contraintes, de façon à maximiser les performances, surtout en ce qui concerne les communications. L'un des problèmes très étudiés est, par exemple, celui de l'augmentation du nombre de noeuds tout en restreignant le plus possible le diamètre du graphe et le degré de ses noeuds. On le rencontre généralement sous le nom de "(d, k) graph problem", consistant à trouver le plus grand graphe possible ayant un degré et un diamètre donné [MEMM82, DELO85].

Pour le réseau N-ARCH, tout en tenant compte de ces problèmes, l'élément nouveau dans l'approche réside dans la possibilité que l'on doit avoir de recouvrir le graphe, à partir de n'importe quel noeud, par un arbre équilibré. Les deux cas extrêmes répondant à cette contrainte sont, d'une part, un anneau de N+1 processeurs recouvert par un arbre "unaire" de profondeur N (fig 3.4.a), et, d'autre part, un réseau complètement connecté de N+1 processeurs recouvert par un arbre "N-aire" de profondeur 1 (fig 3.4.b).

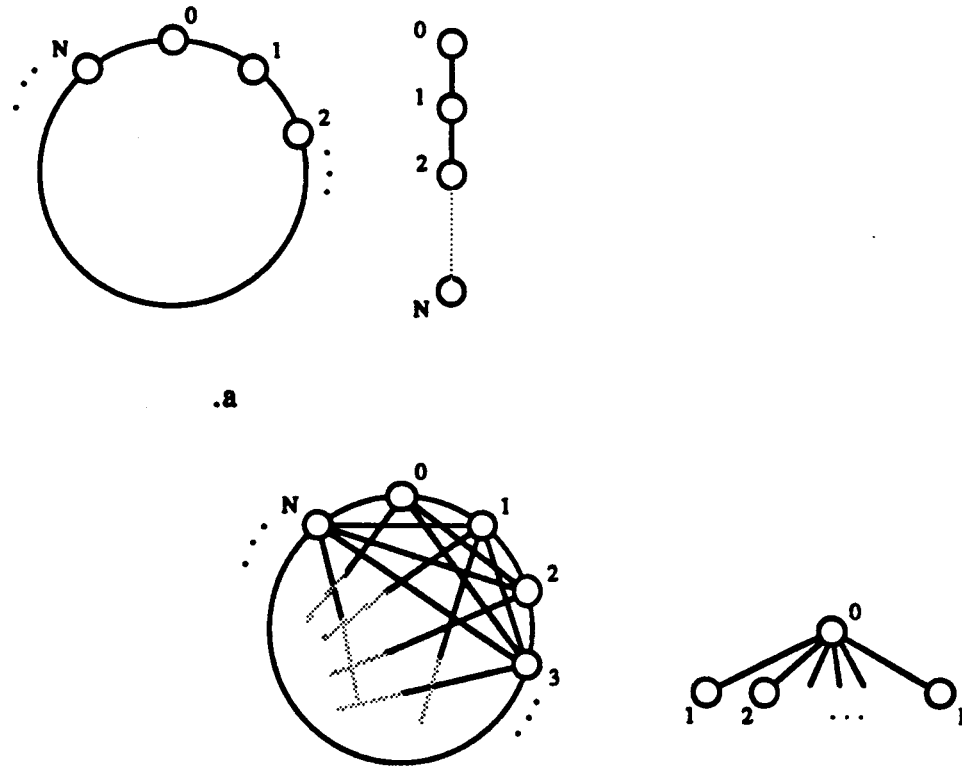


figure 3.4

C'est quelque part entre ces deux topologies extrêmes que devra se situer celle servant de base au réseau N-ARCH. La détermination d'une topologie convenant aux critères ci-dessus est assez délicate. Le chapitre suivant y est consacré.

CHAPITRE 4
ANALYSE DES TOPOLOGIES.

Dans le but d'utiliser la technique de hachage définie plus haut, le graphe modélisant le réseau doit, comme on l'a vu, être tel que tout sommet soit racine d'un arbre de recouvrement régulier et équilibré.

Le problème du recouvrement d'un graphe donné par une arborescence quelconque est un problème classique mais dont les solutions habituelles n'obéissent pas aux contraintes posées ici : arbres de recouvrement non réguliers, non équilibrés ...

Comme on va le voir, les graphes représentant les réseaux de communication proposés à la base des machines multiprocesseurs ne conviennent généralement pas.

Pour cette étude des topologies de divers réseaux, ces derniers seront classés en quatre catégories : réseaux en arbre ; réseaux en anneau ; réseaux en hypercube ; puis enfin les réseaux n'entrant dans aucune des catégories précédentes. Ces catégories sont introduites uniquement dans un souci de clarté et ne représentent pas une classification rigoureuse basée sur la topologie des réseaux considérés. Par exemple, un hypercube d'ordre 3, classé naturellement dans la catégorie "hypercube" et représenté comme ci-dessous :

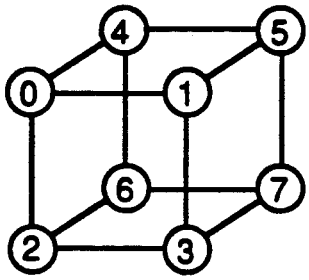


figure 4.1

pourrait très bien être vu de la façon suivante :

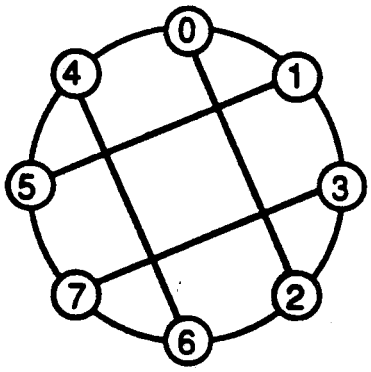


figure 4.2

et être en conséquence classé dans la catégorie "anneau".

4.1 Réseaux déjà existants

4.1.1 Réseaux en arbre

4.1.1.1 Machine en arbre binaire

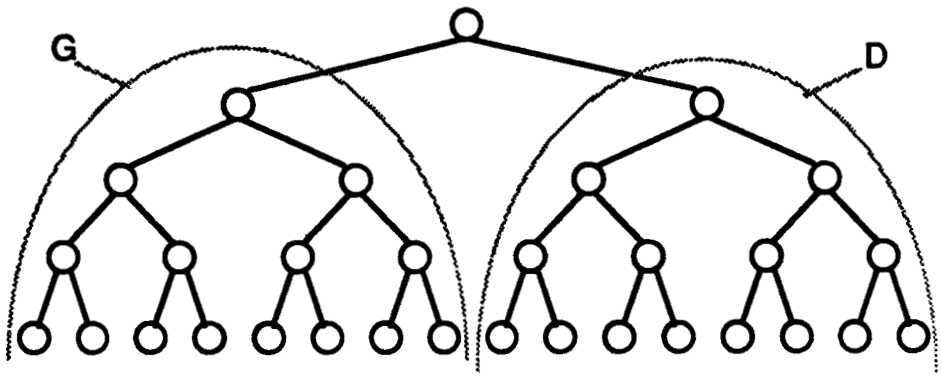


figure 4.3

Ce type de réseau souffre d'un défaut important : toute communication d'un noeud du sous-arbre G (resp. D) à un noeud du sous-arbre D (resp. G) doit passer par la racine. On aboutit donc à un effet de goulot d'étranglement.

D'autre part, sur la base des critères N-ARCH, ce réseau

- n'est pas régulier : la racine est de degré 2, les feuilles sont de degré 1, les autres noeuds de degré 3 ;
- n'admet pas de recouvrement en arbre à partir d'un autre noeud que la racine.

Ce type de réseau sert de base à la définition d'autres réseaux, par adjonction d'arètes de façon à supprimer l'effet de goulot à la racine.

4.1.1.2 X-Arbre

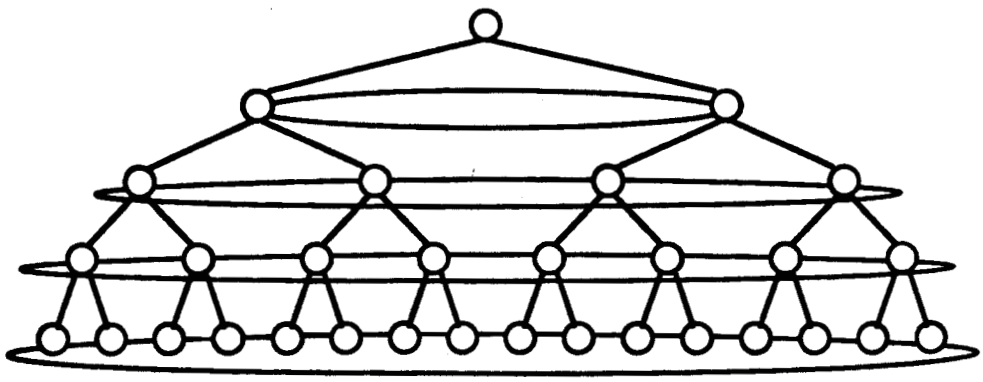
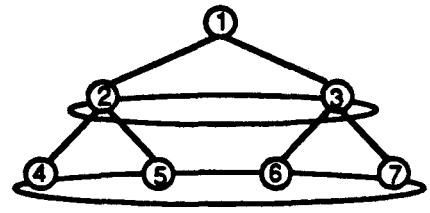


figure 4.4

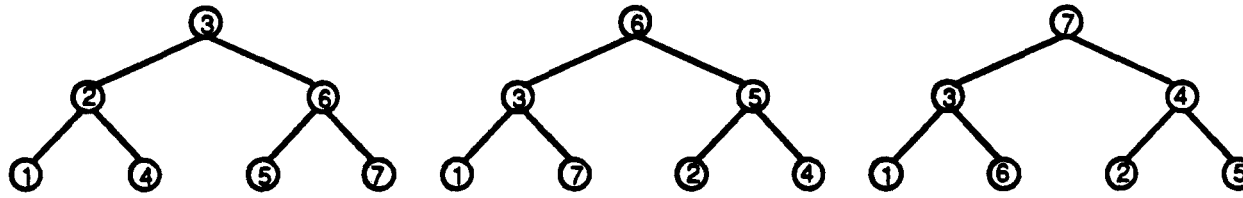
La base est la même que pour le réseau précédent, mais ici tous les noeuds situés sur un même niveau dans l'arbre sont reliés entre eux par un anneau. Le problème de régularité du graphe reste : la racine est de degré 2, les feuilles sont de degré 3, les autres noeuds de degré 5.

Quant au recouvrement à partir d'un noeud quelconque, il n'existe que pour les graphes de petite taille (3 ou 7 noeuds).

Pour un nombre de noeuds N = 7 on a par exemple:



Graphe



Recouvrements
(Les autres recouvrements se déduisent aisément de ceux-ci par symétrie)

figure 4.6

Pour des graphes plus importants, les recouvrements par des arbres binaires équilibrés deviennent impossibles.

Montrons par exemple que pour N = 15 il existe au moins un noeud qui ne peut être racine d'un arbre binaire de recouvrement complet et équilibré. Le graphe est le suivant :

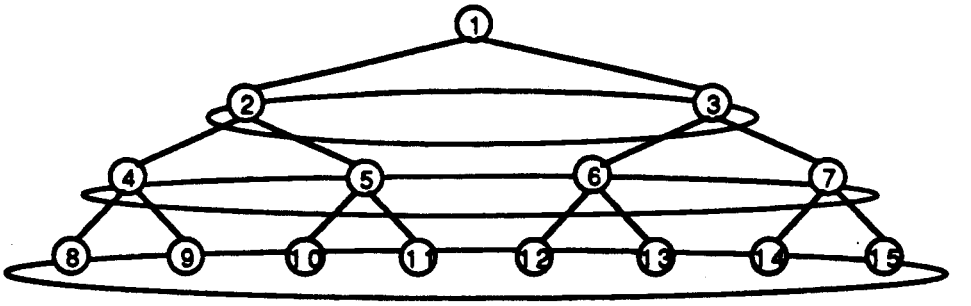
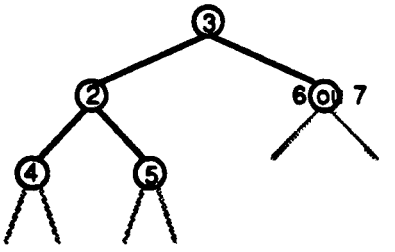


figure 4.7

En conservant la même numérotation que ci-dessus - c'est-à-dire on donne à la racine le numéro 1 ; puis, pour chaque noeud i, ses fils gauche et droit sont respectivement numérotés 2*i et 2*i+1 - on ne peut pas trouver de recouvrement de racine 3.

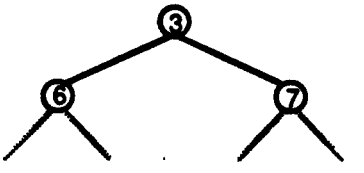
En effet, le degré du noeud 1 dans le graphe étant 2, dans le recouvrement, si 1 n'est pas racine, il ne pourra être qu'une des feuilles de l'arbre. Les fils de 3 ne peuvent donc être que 2, 6 ou 7.

2 est à écarter : il ne peut avoir pour fils 1 (qui doit être une feuille), l'arbre de recouvrement ne peut donc commencer que par :



et 1 ne peut plus être atteint (il faudrait retraverser 2 ou 3).

Donc, si un arbre de recouvrement de racine 3 existe, il commence par :



Pour atteindre maintenant le noeud 1, à partir de 6 ou 7, et bien entendu sans retraverser 3, il faudra traverser au moins trois arcs. Le noeud 1 serait donc à la profondeur 4 dans l'arbre de recouvrement alors que l'arbre cherché est de profondeur 3 [CQFD].

Des constatations du même type permettent de se convaincre de l'impossibilité du recouvrement pour des X-arbres de taille supérieure (N = 31, 63 ...).

4.1.1.3 HyperArbre

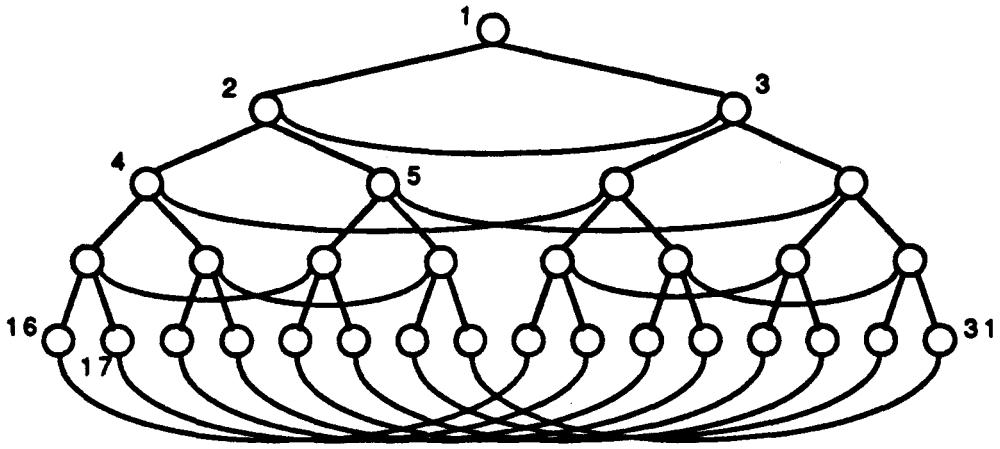


figure 4.8

La numérotation restant toujours la même que ci-dessus, on ajoute ici des arcs joignant, sur un même niveau, des noeuds qui seraient voisins dans un hypercube i.e. ceux dont la distance de Hamming est de 1 (c'est-à-dire dont les écritures binaires respectives diffèrent d'un seul bit). (Pour les critères de sélection des arcs d'hypercube à retenir, voir [GOOD81]).

Ici encore, la racine et les feuilles sont de degré 2 alors que les autres noeuds sont de degré 4.

De plus, il est là aussi impossible de recouvrir le graphe à partir d'un autre noeud que la racine, le contre-exemple le plus évident étant obtenu en essayant de déterminer un recouvrement à partir d'une des feuilles du graphe (numéros 16 à 31 sur la figure 4.8)

4.1.1.4 Sneptree

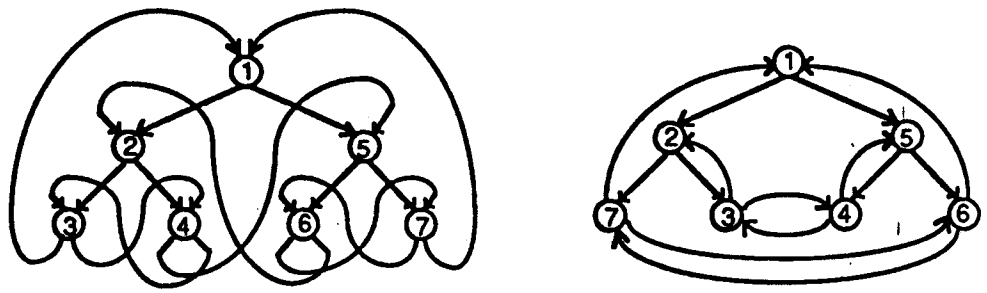
Définition : Un **sneptree** à n niveaux est un arbre binaire complet de 2^n-1 noeuds, dans lequel on considère que les arcs sont dirigés dans le sens racine --> feuilles, 2^n arcs supplémentaires étant ajoutés des feuilles vers les autres noeuds de telle sorte que chaque noeud ait 2 arcs incidents intérieurs et 2 arcs incidents extérieurs.

Pour chacun des noeuds intérieurs de l'arbre un des arcs incidents intérieurs est un arc supplémentaire.

Pour la racine les deux arcs incidents intérieurs sont des arcs supplémentaires.

Note : Les arcs sont considérés comme orientés pour faciliter la définition du réseau mais dans la pratique les liaisons sont bidirectionnelles.

Exemple :



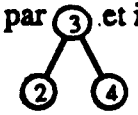
.a .b

figure 4.9

Le nombre de possibilités de connexion des 2^n arcs supplémentaires est élevé mais seuls les deux exemples ci-dessus sont extensibles récursivement, ce qui est intéressant en vue d'une éventuelle réalisation VLSI.

Le réseau 4.9.b est moins intéressant que le réseau 4.9.a du fait du grand nombre de liaisons redondantes qu'il comporte.

Pourrait-on envisager l'utilisation de tels réseaux pour le projet N-ARCH ? Il s'agit en fait de tenter de les recouvrir, à partir de n'importe quel noeud, par un arbre binaire complet.

Pour la version b. le contre-exemple est évident : il suffit de prendre pour racine un noeud dont les 4 arcs sont en fait 2 arcs redondants : les noeuds numérotés 3 et 4 sur la figure. Ces noeuds n'ont que 2 voisins et sont voisins entre eux : dans l'arbre de recouvrement si on prend 3 pour racine, l'arbre commence par  et il ne reste plus qu'un seul voisin pour 4.

Pour la version a. l'impossibilité survient pour la taille de réseau immédiatement supérieure (Sneptree à 4 niveaux).

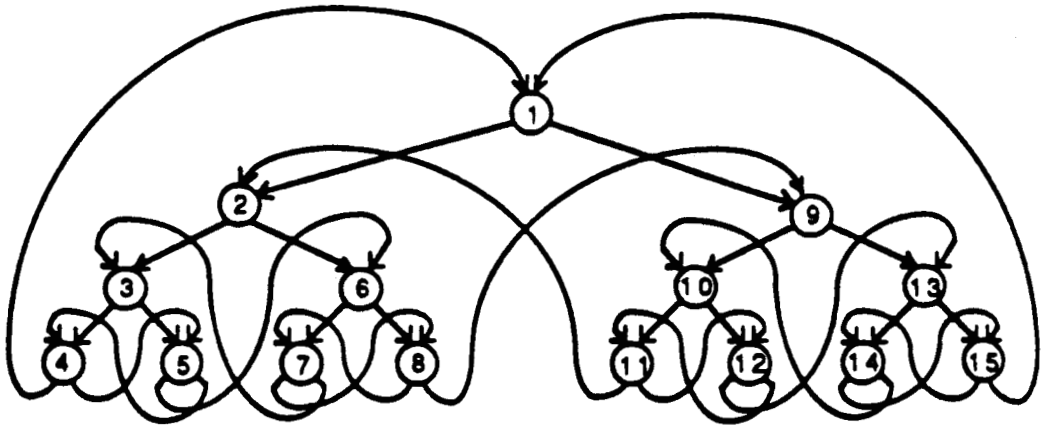
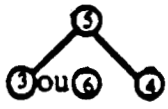


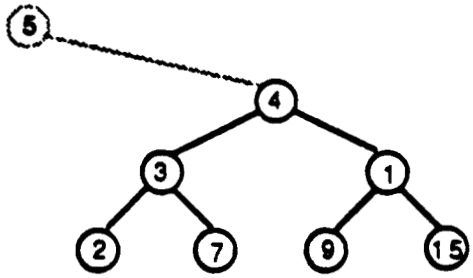
figure 4.10

Cherchons à établir un recouvrement par un arbre binaire complet de racine 5 .

5 possède 3 voisins : 3 , 4 , 6 .

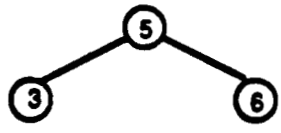
L'arbre ne peut commencer par 

. En effet, le sous-arbre de racine 4 serait nécessairement le suivant :



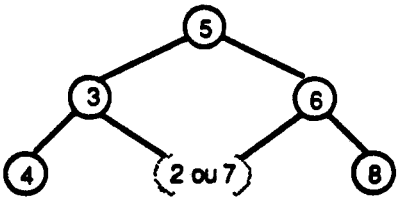
Le deuxième sous-arbre de 5 a donc pour racine 6 , 3 apparaissant dans le sous-arbre de racine 4 . Les voisins de 6 sont : 5 , racine de l'arbre ; 2 et 7 qui apparaissent déjà dans le sous-arbre de racine 4 ; et il ne reste donc qu'un fils possible pour 6 : le noeud 8 ce qui rend impossible la construction de l'arbre binaire.

Donc l'arbre de racine 5 commence obligatoirement par :



Les fils possibles de 3 et 6 sont respectivement 2, 4 et 7 d'une part et 2, 7 et 8 d'autre part.

L'arbre cherché a donc pour premiers niveaux :



De 4 on ne peut atteindre que 5 , 3 et 1, or 5 et 3 figurent déjà dans l'arbre : il n'existe donc pas d'arbre binaire complet de recouvrement ayant 5 pour racine.

4.1.2 Réseaux en anneau

Un simple réseau en anneau ne convenant pas de toute évidence, nous nous intéresserons ici aux réseaux de type "chordal ring", éventuellement en nous écartant légèrement de leur définition habituelle. Ces réseaux sont constitués d'un anneau auquel sont ajoutées des cordes.

Le "chordal ring" classique est de degré 3, l'anneau comporte un nombre pair de noeuds, numérotés de 0 à N-1, des cordes lui sont ajoutées de la façon suivante :

Tout noeud de numéro i pair (resp. impair) est connecté au noeud numéroté (i-ω) MOD N (resp. (i+ω) MOD N) où ω désigne un nombre impair inférieur ou égal à N/2.

La figure ci-dessous est un exemple de chordal ring avec N = 18 et ω = 5 :

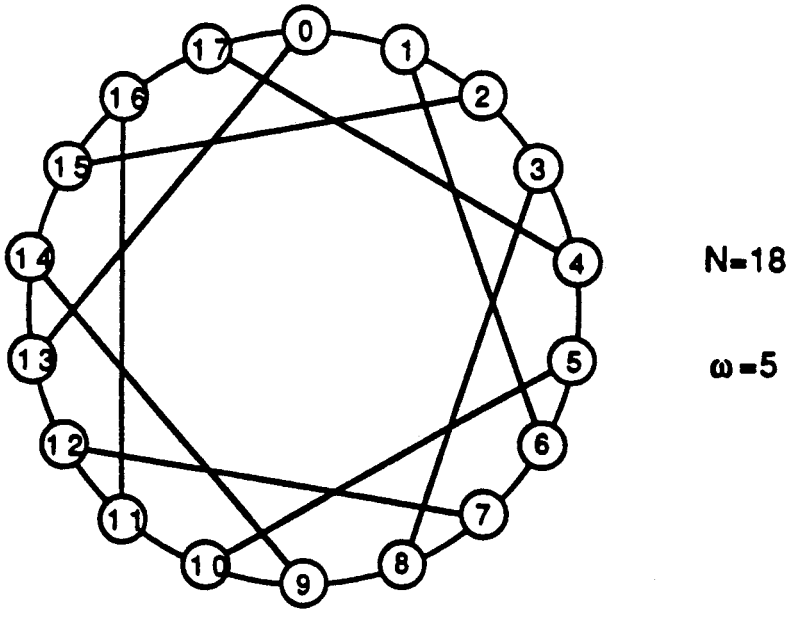


figure 4.11

Pour N=8 et ω=3 on obtient le 3-cube évoqué en début de chapitre (figures 4.1 & 4.2), à une renumérotation près :

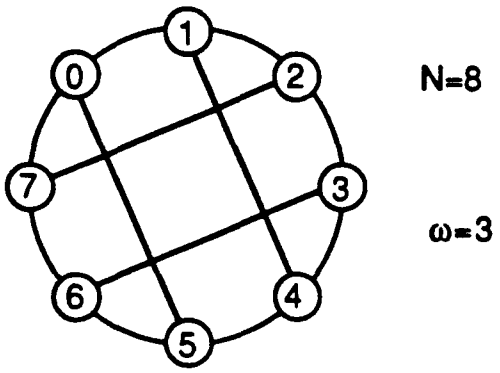


figure 4.12

Pour un entier N donné, la longueur de corde peut être choisie de façon à optimiser le diamètre du graphe obtenu.

Ce réseau est intéressant par sa régularité -tous les noeuds sont de degré 3- mais la parité du nombre de noeuds rend impossible le recouvrement par un arbre binaire complet puisque ce dernier comporte un nombre impair de noeuds. On pourrait éventuellement se contenter de recouvrir tout le graphe hormis un noeud mais ceci n'est pas possible non plus comme on va le voir ci-dessous. Il est à noter également que tout recouvrement par un arbre n-aire, avec n supérieur ou égal à 3, est impossible du fait du degré trop faible du graphe.

4.1.2.2 Impossibilité de recouvrement à un noeud près d'un anneau cordé

Remarque préliminaire : Le nombre de noeuds d'un arbre binaire complet à n niveaux étant de la forme $N' = 2^n-1$, on ne s'intéressera ici qu'aux "chordal rings" ayant un nombre de noeuds de la forme $N = 2^n$.

Proposition :

> Soit G le graphe d'un réseau "chordal ring" dont le nombre de noeuds est $N = 2^n$, ces noeuds étant numérotés de 0 à N-1.
>
> Il est impossible de recouvrir N-1 noeuds de G par un arbre binaire complet à n niveaux de racine un noeud i donné de G.

Pour démontrer cette proposition on peut se ramener à rechercher un recouvrement de racine 0 grâce au lemme suivant :

Lemme :

Soit R_0 un arbre de recouvrement de G à un noeud près,dont la racine est le noeud 0 de G.

Soit i un sommet de G.

Soit R la renumérotation des sommets de R_0 pour laquelle tout sommet j de R_0 est renuméroté :

$$R(j) = (j+i) \text{ MOD } N \text{ si i est pair;}$$

$$R(j) = (i-j) \text{ MOD } N \text{ si i est impair.}$$

Le graphe R_i ainsi obtenu est un arbre de recouvrement de G à un noeud près, dont la racine est le noeud i.

Démonstration :

1/ Deux sommets différents sont renumérotés différemment. En effet :

si $(i+j_1) \text{ MOD } N = (i+j_2) \text{ MOD } N$ (resp. $(i-j_1) \text{ MOD } N = (i-j_2) \text{ MOD } N$)

alors $j_1=j_2$ MOD N, donc $j_1=j_2$ car les sommets sont numérotés par des entiers compris entre 0 et N-1.

R_i est donc un arbre.

2/ Reste à démontrer que si j_1 et j_2 sont voisins dans R_0, alors l'arc $(R(j_1),R(j_2))$ correspondant dans est bien un arc de G, c'est-à-dire que les sommets $R(j_1)$ et $R(j_2)$ sont voisins dans G.

Or j_1 et j_2 sont voisins dans R_0, et donc dans G, si et seulement si

$j_1 = j_2 \pm 1$

ou

$j_1 = j_2 - \omega$ si j_2 pair (et donc j_1 impair)

$j_1 = j_2 + \omega$ si j_2 impair (et donc j_1 pair)

1er cas : $j_1 = j_2 \pm 1$

si $j_1 = j_2 \pm 1$

alors, pour tout i,on a

$i - j_1 = i - j_2 \pm 1$ <=> $R(j_1) = R(j_2) \pm 1$ si i impair

$j_1 + i = j_2 \pm 1 + i$ <=> $R(j_1) = R(j_2) \pm 1$ si i pair

donc $R(j_1)$ et $R(j_2)$ sont voisins dans G.

2ème cas : $j_1 = j_2 \pm \omega$

a) Si i est pair, pour tout sommet j, j et (i+j) MOD N = R(j) sont de même parité.

Si j_2 est pair, $j_1 = (j_2 - \omega)$ MOD N, $R(j_2) = (i+j_2)$ MOD N est pair et donc voisin dans G de $(i + j_2 - \omega)$ MOD N = $R(j_1)$

Si j_2 est impair, $j_1 = (j_2 + \omega)$ MOD N, $R(j_2) = (i+j_2)$ MOD N est impair et donc voisin dans G de $(i + j_2 + \omega)$ MOD N = $R(j_1)$

b) Si i est impair, pour tout sommet j, j et (i+j) MOD N = R(j) sont de parités différentes.

Si j_2 est pair, $j_1 = (j_2 - \omega)$ MOD N, $R(j_2) = (i-j_2)$ MOD N est impair et donc voisin dans G de $(i - j_2 + \omega)$ MOD N = $R(j_1)$

Si j_2 est impair, $j_1 = (j_2 + \omega)$ MOD N, $R(j_2) = (i-j_2)$ MOD N est pair et donc voisin dans G de $(i - j_2 - \omega)$ MOD N = $R(j_1)$. CQFD

Démonstration de la proposition :

Chaque noeud étant de degré trois, pour recouvrir le graphe à partir du noeud 0 par un arbre binaire complet on n'a en fait de choix qu' à la racine :

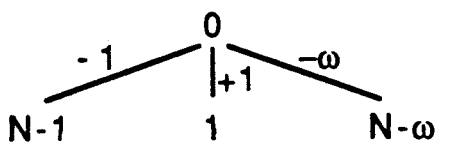


figure 4.13

Pour tous les autres niveaux de l'arbre, un des arcs du graphe qui a pour extrémité le noeud considéré étant l'arc supérieur dans l'arbre il ne reste plus que deux arcs à utiliser pour les deux arcs inférieurs.

Ce qui donne, pour les quatre premiers niveaux :

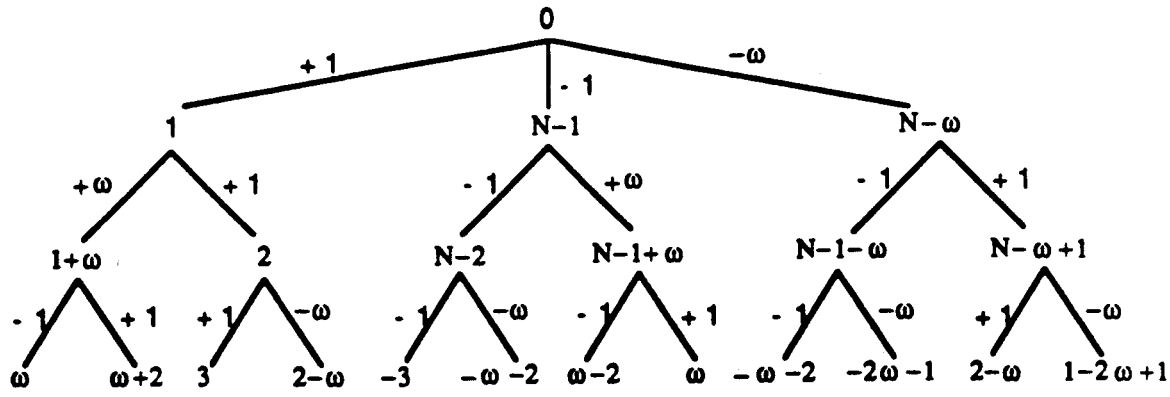


figure 4.14

Un arbre binaire de recouvrement de racine 0 serait constitué de la racine 0 et de deux des trois sous-arbres de racines respectives 1, N-1, N-ω.

Or, au niveau 4, ces sous-arbres possèdent des noeuds en commun quelque soit la façon dont on les considère deux à deux :

ω	apparait dans les sous-arbres de racines	1 et N-1 ;
2 - ω	apparait dans les sous-arbres de racines	1 et N-ω ;
- ω - 2	apparait dans les sous-arbres de racines	N-1 et N-ω.

Donc un recouvrement tel que celui qui est recherché est impossible dès que l'arbre possède plus de trois niveaux, c'est-à-dire dès que N>8.

Pour N = 8, ω, qui doit être impair et inférieur à N/2 (cf. définition), ne peut qu'être égal à 3 et on obtient :

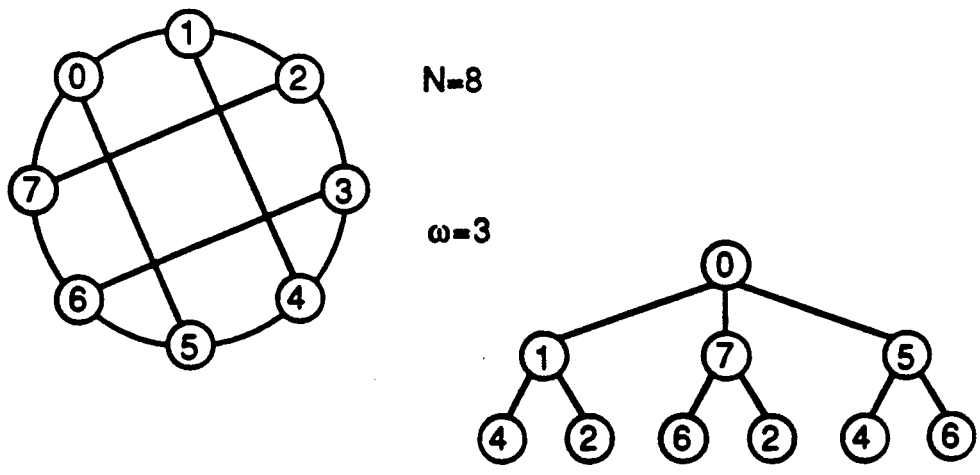


figure 4.15

Donc là aussi on aboutit à une impossibilité.

Donc, quelque soit la valeur de N (le cas où N = 4, inintéressant, n'est pas considéré), le recouvrement recherché n'existe pas et la proposition est démontrée.

4.1.2.3 Anneau avec circuit hamiltonien

On classe également dans cette famille une forme de réseau vers laquelle on s'est orienté dans le projet N-ARCH.

Il s'agit là encore d'un anneau avec adjonction de cordes mais ici celles-ci forment des circuits hamiltoniens (i.e. passant une fois et une seule par chacun des noeuds).

Le nombre N de noeuds du graphe est impair, son degré d est pair, et , les noeuds étant numérotés de 0 à N-1, chaque noeud i est connecté aux noeuds $(i \pm \omega_k)$ MOD N où $k \in [1, d/2]$ les ω_k étant à déterminer. Les connections sur l'anneau correspondent à $\omega_1=1$.

Exemple : Avec N = 7; d = 6; $\omega_1 = 1$; $\omega_2 = 2$ on obtient le graphe ci-dessous :

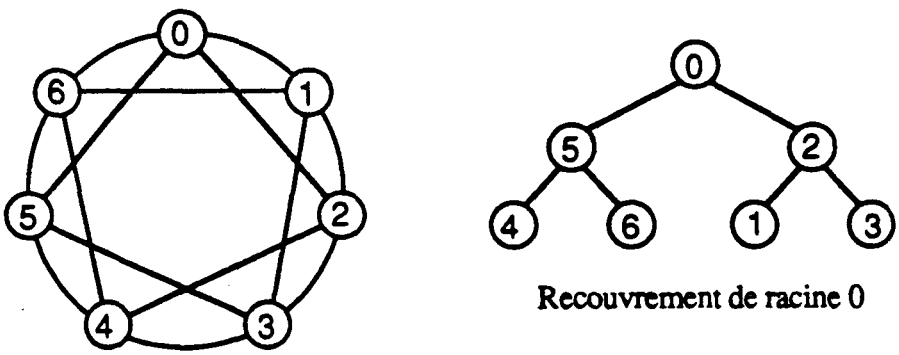


figure 4.16

Un recouvrement de racine i ($i \in [1,N-1]$) se déduit du recouvrement de racine 0 en ajoutant i (mod N) à chaque numéro de noeud dans le recouvrement. Pour de grandes valeurs de N, si on garde un degré faible, il n'est plus possible de trouver un recouvrement par un arbre.

4.1.3 Réseaux en hypercube

Je classe dans cette famille l'hypercube au sens habituel (appelé aussi n-cube, cube binaire, cosmic-cube [SEIT85] ...) ainsi que les réseaux à base d'hypercubes généralisés [BHUY84], de tores , CCC (Cube Connected Cycle) [PREP81] ...etc...

4.1.3.1 Hypercube simple

Il s'agit d'un réseau comportant $N=2^n$ noeuds assimilable à un cube à n dimensions dont chaque arète comporte 2 sommets.

De nombreuses études ont été effectuées à son sujet. Des machines parallèles dont l'architecture est basée sur un hypercube sont déjà commercialisées.

Ce réseau dispose d'un degré et d'un diamètre raisonnables puisque tous deux valent $n = \log_2 N$.

Pour le projet N-ARCH on s'est donc posé la question de l'utilisation des hypercubes [TOUR86]. Le problème vient de l'impossibilité de recouvrir un hypercube par un arbre complet et équilibré. On a donc envisagé des recouvrements par des structures voisines des arbres, par exemple des quasi-arbres, qui sont des arbres avec redondance de certains noeuds aux feuilles.

Quasi-arbres:

Définition : Les noeuds redondants apparaissant en feuilles ne seront représentés qu'une seule fois avec plusieurs arcs d'entrée.

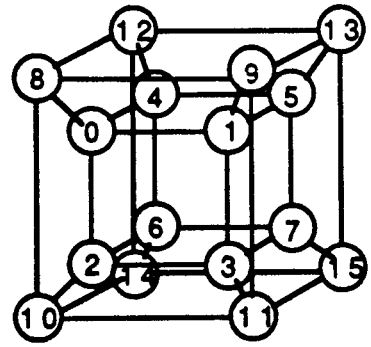
La définition d'un **quasi-arbre** d'ordre **n** et de degré **d** est alors donnée par le tableau suivant :

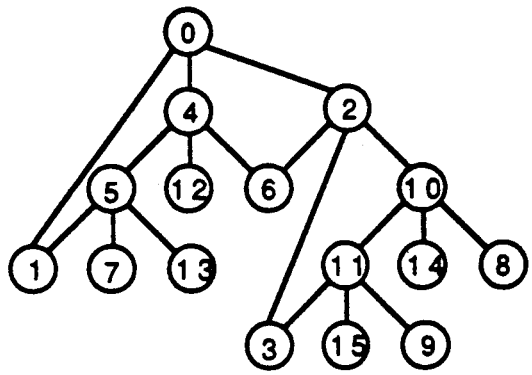
	Nombre d'arcs incidents	
	intérieurs	extérieurs
racine	0	d
feuille	≤n	0
autre noeud	1	d

Tableau 4.1

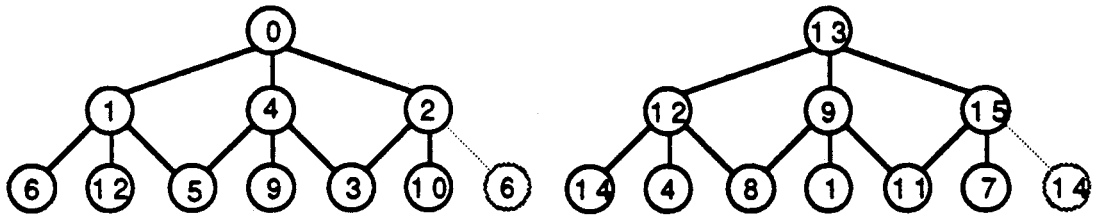
Exemples :

Recouvrements d'un 4-cube





par un quasi-arbre d'ordre 2, de degré 3 (très déséquilibré)



par deux quasi-arbres (2,3) équilibrés

figure 4.17

Ce type de recouvrement n'a finalement pas été retenu.

Le nombre de noeuds d'un hypercube à n dimensions étant de 2^n et celui d'un arbre binaire complet à n niveaux étant de 2^n-1, on peut se demander s'il est possible de recouvrir à un noeud près un hypercube par un tel arbre. Pour n>2 la réponse est non [BRAN86].

4.1.3.2 Hypercube généralisé

De même que l'hypercube peut être envisagé comme un cube à n dimensions dont chaque arête comporte 2 sommets, l'hypercube généralisé est un parallélépipède à n dimensions dont chaque arête comporte un nombre quelconque de sommets. C'est ce point qui en fait une généralisation de l'hypercube "classique".

On peut donc, contrairement à ce qui se passe dans le cas de l'hypercube, dont le nombre de sommets doit être une puissance de 2, définir un hypercube généralisé pour tout nombre de sommets, le résultat n'étant toutefois pas toujours intéressant en ce qui concerne l'aspect réseau d'interconnexion et plus particulièrement dans le cadre du projet N-ARCH comme on va le voir.

Définition de l'hypercube généralisé :

Soit N un nombre entier. On peut construire un HCG (HyperCube Généralisé) à N sommets de la façon suivante :

On décompose N en produit de facteurs (non nécessairement premiers) :

$$N = \prod_{i=1}^{n} f_i$$

n sera le nombre de dimensions du HCG, f_i le nombre de sommets dans la ième dimension.

Un sommet S (S∈ [0,N-1]) est représenté par un n-uplet $(x_1, x_2, ..., x_n)$ chaque xi étant compris entre 0 et f_i-1 et ayant un poids p_i avec $p_i = \prod_{k=1}^{i-1} f_k$ et $\sum_{i=1}^{n} x_i p_i = S$.

S sera relié aux sommets de la forme $(x_1, ..., x_i', ..., x_n)$ pour i variant de 1 à n et x_i' variant de 0 à f_i-1.

Exemple : Pour n = 12 = 3.2.2 on obtient :

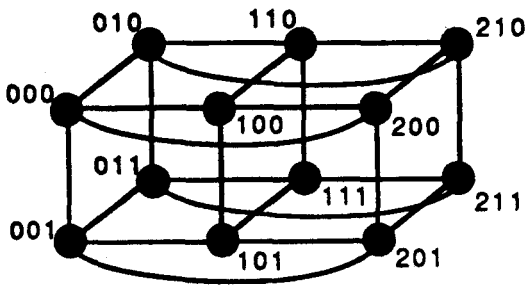


figure 4.18

Le degré d du graphe est donné par la formule : $d = \sum_{i=1}^{n} (f_i - 1)$, son diamètre D est égal à n. On retrouve ici le problème habituel : il vaut mieux décomposer N en facteurs faibles pour obtenir un degré faible, mais ces facteurs seront alors plus nombreux et donc on obtiendra un diamètre plus élevé.

Pour être utilisé dans le réseau N-ARCH, un tel graphe ne sera intéressant que si on peut le recouvrir par un arbre complet et équilibré.

Si on choisit un arbre binaire comme recouvrement on est amené à décomposer en facteurs des nombres de la forme 2^n-1.

Les nombres de la forme 2^n-1 sont généralement désignés sous le nom de nombres de Mersenne (cette appellation est toutefois parfois réservée aux nombres premiers de cette forme, ou pour lesquels n est premier).

Dans l'optique qui nous intéresse, i.e. décomposition de 2^n-1 en facteurs en vue de la construction d'un HCG recouvrable par un arbre binaire, une "bonne" décomposition doit comporter un nombre raisonnable de facteurs puisque de lui dépend le diamètre du réseau, ces facteurs étant faibles puisque jouant sur le degré du graphe. Le tableau ci-dessous permet de se rendre compte de la rareté des bonnes décompositions des nombres de Mersenne.

n	2^n-1	décomposition	degré	diamètre
1	1	premier		
2	3	premier		
3	7	premier		
4	15	3.5	6	2
5	31	premier		
6	63	7.9	14	2
		7.3.3	10	3
7	127	premier		
8	255	3.5.7	22	3
9	511	7.73	78	2
10	1023	3.11.31	42	3
		33.31	62	2
11	2047	23.89	108	2
12	4095	3^2.5.7.13	26	5

Tableau 4.2

Remarque: de plus les graphes correspondants n'admettent pas nécessairement de recouvrement par un arbre binaire complet et équilibré.

Les décompositions obtenues sont donc généralement peu intéressantes, mais pour n = 6 c'est-à-dire N = 63 = 7.3.3 on obtient un graphe que l'on peut recouvrir par un arbre binaire. Ce graphe est donné sur la figure 4.19 dans laquelle, pour des raisons de lisibilité, tous les arcs ne sont pas représentés : seul le noeud 020 comporte tous ses arcs incidents. Sur un même "axe", tous les noeuds devraient être reliés deux à deux. Le recouvrement de ce réseau par un arbre binaire est représenté quant à lui par la figure 4.20.

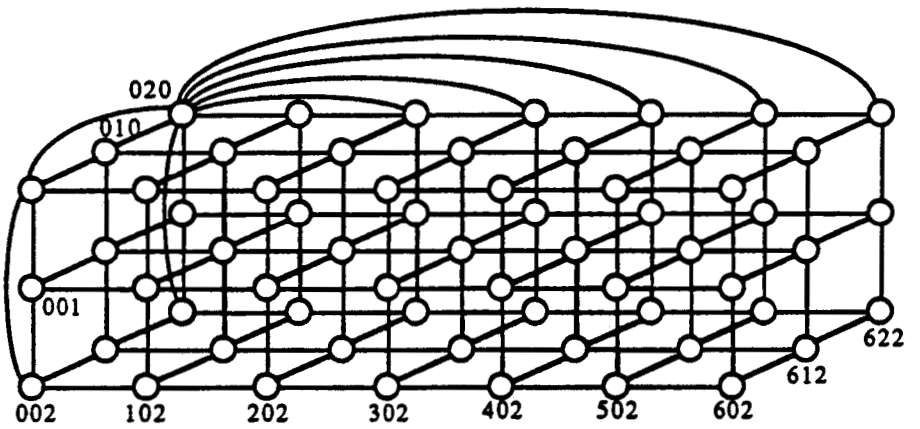


figure 4.19

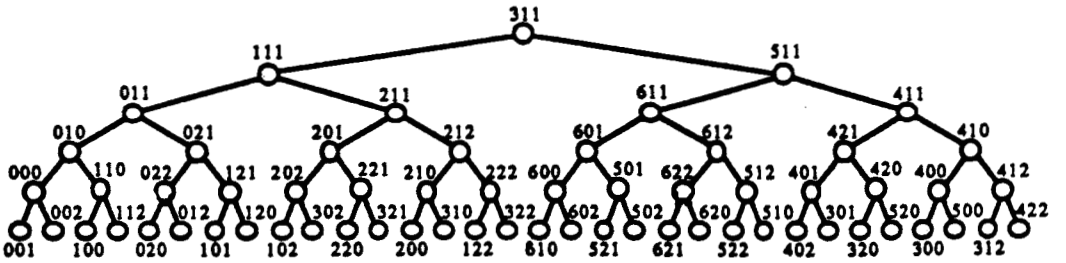


figure 4.20

Si on choisit un arbre ternaire comme recouvrement on est alors amené à décomposer en facteurs des nombres de la forme $N = 1/2(3^n-1)$ ce qui donne par exemple pour n = 4 : N = 40 = 5.4.2 et donc un degré d = 8 et un diamètre D = 3 (la décomposition N = 40 = 5.2.2.2 est à écarter car le diamètre du graphe obtenu serait D = 4 alors qu'un arbre ternaire à 40 noeuds est de profondeur 3 seulement). Ce graphe ainsi

que son recouvrement par un arbre ternaire sont représentés ci-dessous (figure 4.21). Comme pour la figure 4.19, sur la figure 4.21.a tous les arcs ne sont pas représentés.

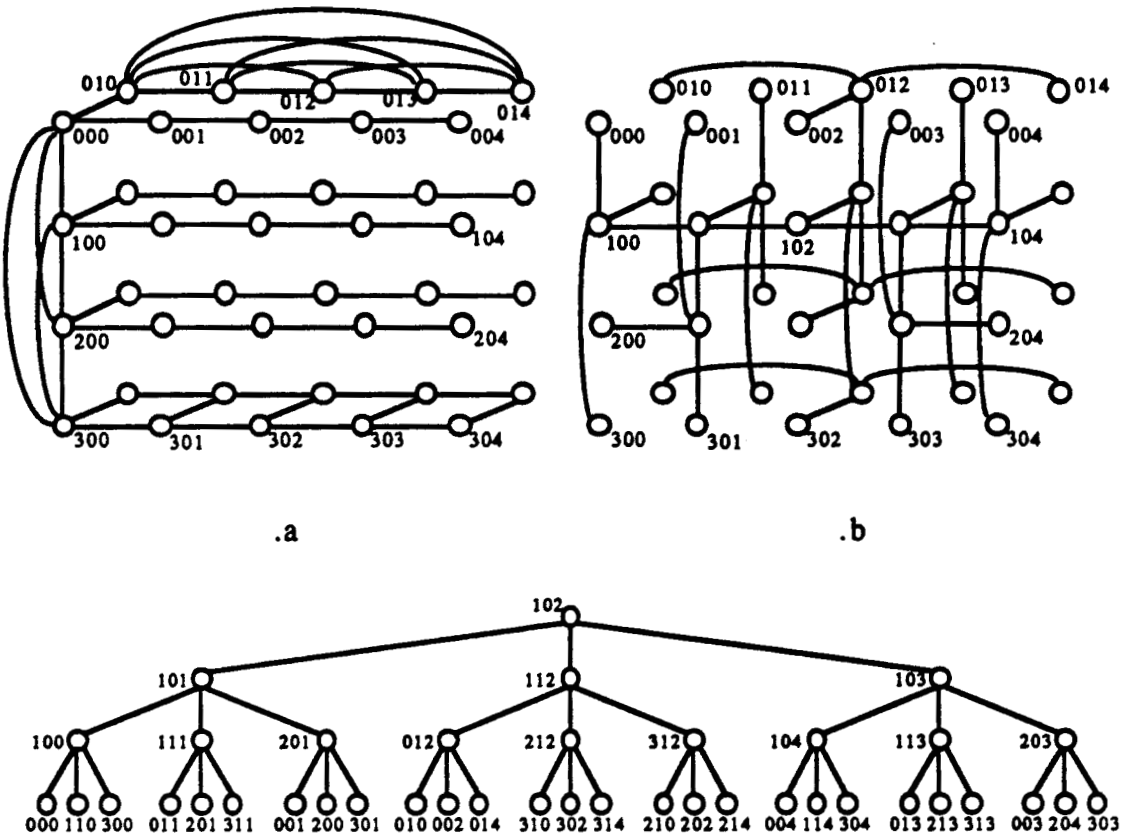


figure 4.21

4.1.4 Autres

4.1.4.1 Graphes de de Bruijn

Ce sont des graphes sur lesquels il est très facile de répartir de manière optimale des arbres binaires.

Ils sont définis pour des graphes dont le nombre de noeuds est une puissance de 2. Leur degré est d = 4 quelque soit la taille du graphe.

Construction : Soit $N = 2^n$ le nombre de noeuds du graphe, ces noeuds étant numérotés de 0 à N-1.

Selon une méthode déjà employée plus haut, pour la construction du graphe on considèrera que les arcs du graphe sont orientés.

De chaque noeud i on fait partir deux arcs le reliant respectivement aux noeuds 2i MOD N et (2i + 1) MOD N.

On remarque tout de suite une particularité de ces graphes : les noeuds 0 et N-1 comportent une boucle (arc reliant un noeud à lui-même). En effet

$$2.0 \text{ MOD } N = 0 \qquad \text{et} \qquad (2(N-1)+1) \text{ MOD } N = 2N-1 \text{ MOD } N = N-1$$

C'est ce qui en fait un type de réseau non classable dans une des catégories précédentes. On peut toutefois "voir" ces réseaux pratiquement comme des arbres.

Exemple : Pour N = 8 on obtient le graphe ci-dessous :

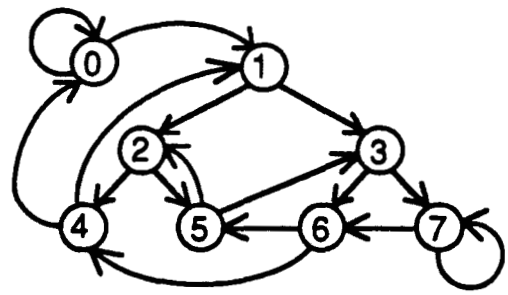


figure 4.22

Sur un tel graphe un arbre binaire de profondeur 4 sera réparti de la manière suivante en parcourant simplement les arcs à partir du noeud numéro 1 :

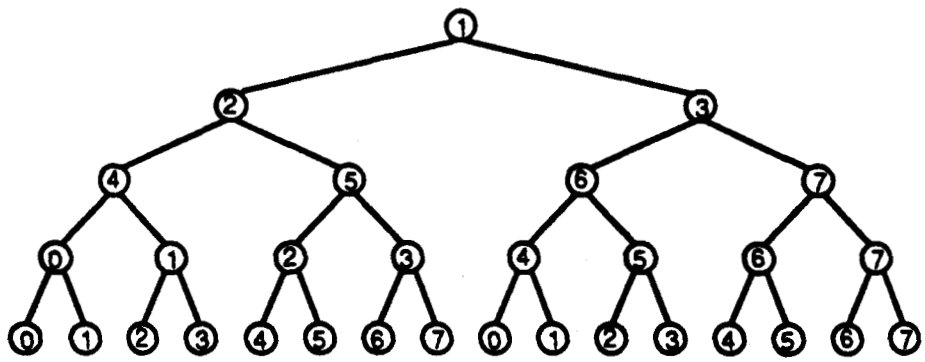


figure 4.23

Sur cette figure, chaque noeud de l'arbre porte le numéro du noeud du graphe de De Bruijn de la figure 4.21 sur lequel il sera placé. On arrive donc à répartir de façon très équilibrée un arbre binaire sur un tel réseau.

Mais, pour le projet N-ARCH, le problème n'est pas de répartir mais de recouvrir à partir de n'importe quel noeud par un arbre . Le nombre de noeuds étant une puissance de 2, on ne peut espérer recouvrir le graphe par un arbre binaire qu'à un noeud près. Pour l'exemple ci-dessus, le recouvrement à un noeud près ayant le noeud 1 pour racine est trivial :

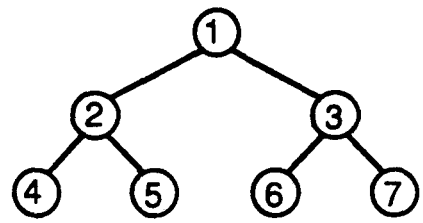


figure 4.24

Par contre, en représentant le graphe comme ci-dessous :

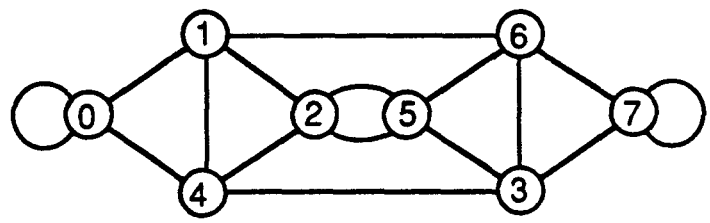
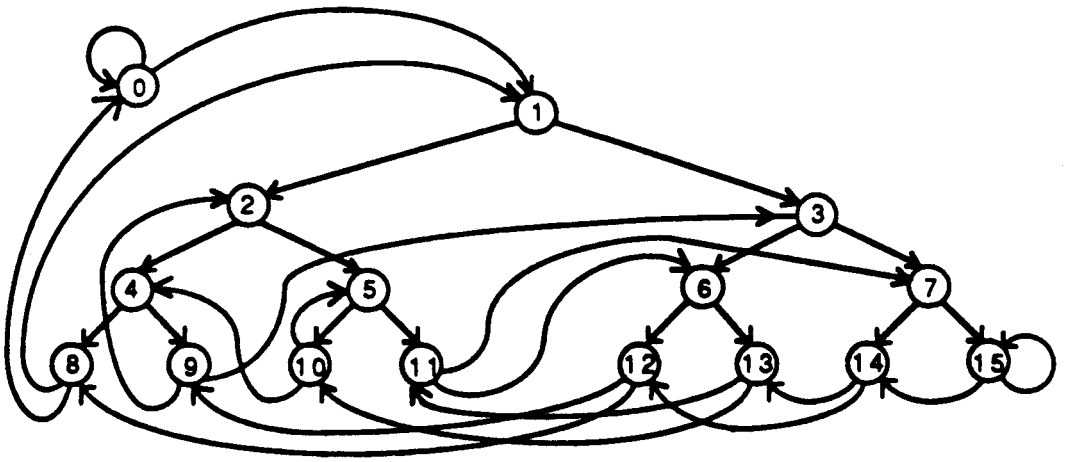


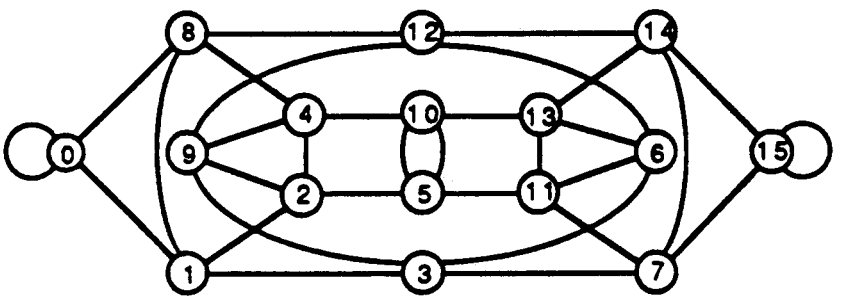
figure 4.25

on voit immédiatement que le recouvrement cherché existera pour tous les noeuds sauf les noeuds particuliers 0 et 7. Pour 0 par exemple, ses fils gauche et droit ne peuvent qu'être respectivement 4 et 1, qui eux-mêmes ne peuvent avoir comme fils que 3 et 2 (pour 4) et 2 et 6 (pour 1). Il y a donc impossibilité du fait de la présence du noeud 2 dans les deux cas.

Pour des graphes de taille plus élevée, non seulement le recouvrement à partir de 0 ou N-1 reste impossible mais de plus d'autres impossibilités apparaissent pour d'autres noeuds. Pour N = 16 on obtient le graphe suivant :



a. représentation "arborescente"



b. représentation "symétrique"

figure 4.26

L'arbre de recouvrement de racine 0, construit de gauche à droite et de haut en bas, commencerait obligatoirement par :

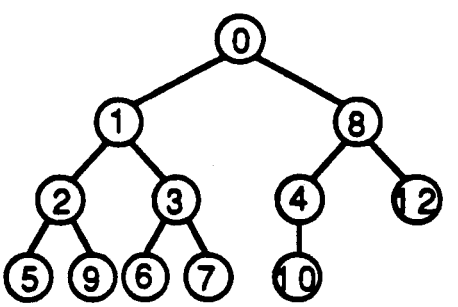


figure 4.27

et le noeud 4 n'a plus qu'un fils possible, d'où l'inexistence du recouvrement cherché.

Une autre impossibilité apparait si on cherche un recouvrement, toujours à un noeud près, de racine le noeud 2. Pour ses fils quatre possibilités existent : 1, 4, 5, 9.

Si on choisit 5 comme racine de l'un des sous-arbres de 2, alors ce sous-arbre commence obligatoirement par :

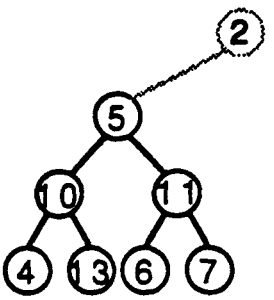
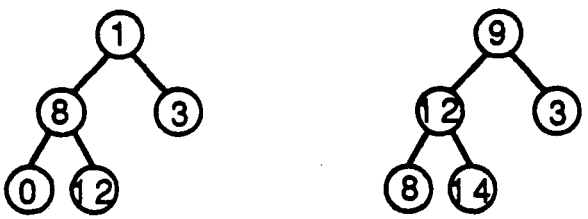


figure 4.28

Comme racine du deuxième sous-arbre le noeud 4 est, par conséquent, à écarter d'office. Le deuxième sous-arbre, de racine 1 ou 9, a alors la forme suivante :



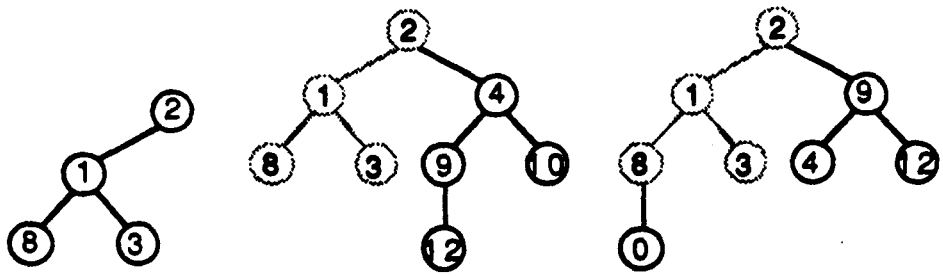
.a .b

figure 4.29

Dans les deux cas, représentés sur la figure 4.28, le noeud 3 n'a plus qu'un seul fils possible : 9 pour 4.28.a, 1 pour 4.28.b, les autres voisins (6 et 7 dans les deux cas) figurant déjà dans le sous-arbre de racine 5.

Donc le noeud 5 est à éliminer de la liste des fils possibles pour le noeud 2.

Si maintenant on choisit 1 comme racine de l'un des sous-arbres de 2, alors le début de l'arbre de recouvrement est obligatoirement comme ci-dessous (fig. 4.30.a), ce qui laisse comme deuxième sous-arbre les deux possibilités représentées sur les figures 4.30.b et 4.30.c.



.a .b .c

figure 4.30

Là encore, dans les deux cas, on aboutit à une impossibilité puisque d'une part, en 4.30.b, le noeud 9 n'a plus qu'un fils possible : 12 ; d'autre part, en 4.29.c, le noeud 8 n'a plus qu'un fils possible : 0.

Donc le noeud 1 est également à éliminer de la liste des fils possibles du noeud 2.

Il reste donc comme seules possibilités les noeuds 4 et 9, ce qui donne le recouvrement ci-dessous (fig. 4.31), sans autre choix possible que ceux représentés :

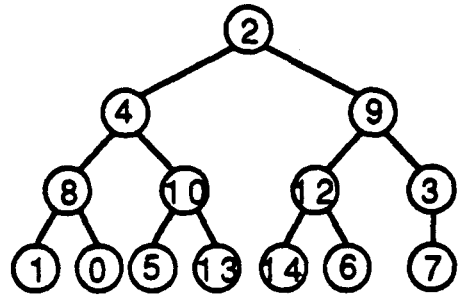


figure 4.31

Ce recouvrement ne convient pas non plus puisque le noeud 3 n'y possède qu'un seul fils.

4.2 Réseaux N-ARCH

Devant les difficultés rencontrées dans la recherche d'une topologie répondant aux caractéristiques imposées pour l'élaboration du réseau N-ARCH, il a été envisagé de s'écarter de certaines des contraintes posées au départ. On a déjà vu au passage pour certains des réseaux ci-dessus qu'on s'était intéressé à des recouvrements à un noeud près. Dans le cas particulier de l'hypercube, nous avons vu qu'un tel recouvrement était

impossible. Par contre, partant de recouvrements partiels de l'hypercube, il est possible d'établir un recouvrement qui, bien qu'assez particulier, pourrait convenir.

4.2.1 Hypercube et arbre binaire à double racine

Comme on l'a vu plus haut, recouvrir un n-cube à 1 noeud près par un arbre binaire complet équilibré est impossible dès que n>2. Par contre on trouve dans [BRAN86] le théorème suivant :

En notant T_n un arbre binaire complet à n niveaux et C_n un hypercube à n dimensions,

Théorème :

| Quelque soit n, on peut inscrire T_n dans C_{n+1}.

Le nombre de sommets de C_{n+1} étant 2^{n+1}, autrement dit $2(2^n-1)+2$, nous avons envisagé la possibilité de recouvrir , à 2 noeuds près, par deux arbres binaires complets à n niveaux, T_n et T_n'. Cette fois-ci la réponse est oui et on peut même choisir T_n, T_n' et les deux noeuds restant x et x' de façon à recouvrir entièrement C_{n+1} par une structure, que nous appellerons **arbre binaire à double racine**, ayant la forme suivante :

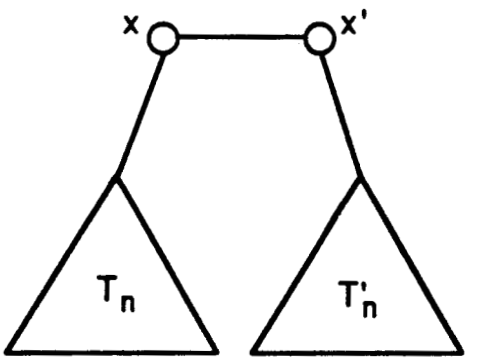


figure 4.32

Une démonstration de cette propriété pourra être trouvée dans [DESH86].

Exemple : Pour C_4 on obtient le recouvrement représenté sur la figure suivante :

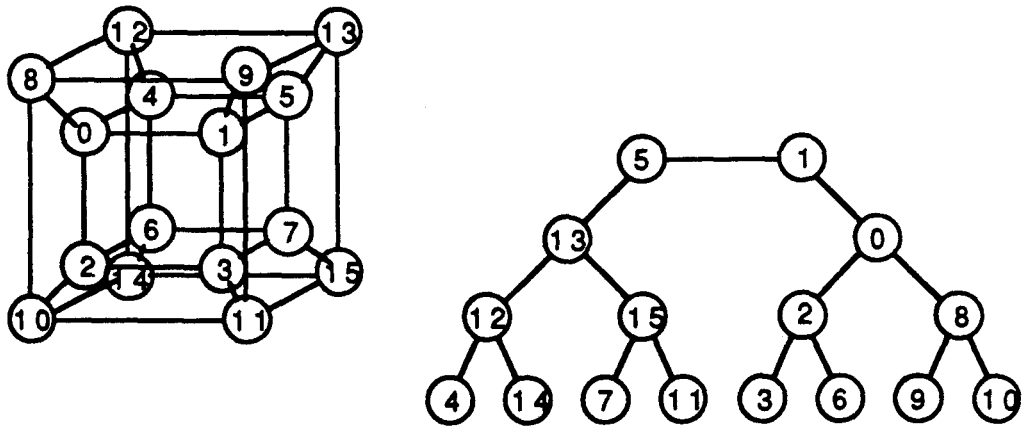


figure 4.33

Connaissant le recouvrement de C_n de racine droite ou gauche x, on déduit le recouvrement de racine y en remplaçant tout noeud s du recouvrement par le *ou exclusif* entre s d'une part et le *ou exclusif* entre x et y d'autre part.

L'hypercube, muni de ce recouvrement en arbre binaire à double racine, représente une solution envisageable pour le réseau N-ARCH à ceci près que le recouvrement en question n'est pas aussi équilibré qu'on ne l'avait souhaité dans les contraintes de départ.

4.2.2 Réseaux AFM

Il s'agit ici de réseaux de type "anneau cordé" pour lesquels on acceptera un degré relativement élevé.

4.2.2.1 Réseau AFM

Le réseau AFM [MOUY86] consiste en une généralisation des réseaux avec circuit hamiltonien, introduits plus haut, à tout anneau comportant un nombre de noeuds de la forme $N = 2^n - 1$, qui est le nombre de sommets d'un arbre binaire complet à n niveaux.

Il s'agit donc d'un anneau auquel on ajoute des cordes. Son degré est d=2(n-1) et, les noeuds étant numérotés de 0 à N-1, chaque noeud i est connecté aux noeuds $(i \pm \omega_k)$ MOD N où $k \in [1, d/2]$ les ω_k étant déterminés par : $\forall\, k \in [\,1, n-1\,]\,,\ \omega_k = 2^{k-1}$.

Un arbre de recouvrement est déterminé par :

- Racine : noeud 0 considérée comme le niveau 0 de l'arbre;

- Pour tout noeud i ($i \in [\,0, N-1\,]$) au niveau j
  son fils gauche est $(i - 2^{n-2-j})$ MOD N
  son fils droit $(i + 2^{n-2-j})$ MOD N

Exemple: Avec N = 15; n = 4; d = 6; $\omega_1 = 1$; $\omega_2 = 2$; $\omega_3 = 4$ on obtient :

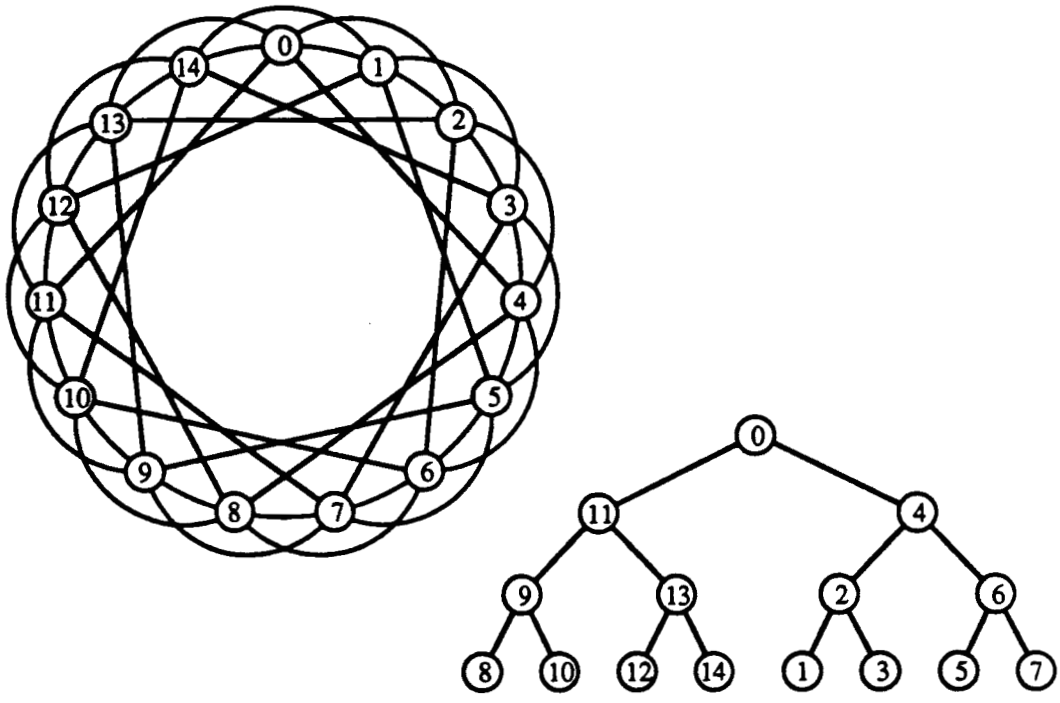


figure 4.35

4.2.2.2 Réseau AFM+

Le graphe de ce réseau est en fait un graphe partiel du graphe AFM, avec un degré diminué de 2.

Son degré est donc d = 2(n-1) - 2 = 2(n - 2), le nombre de sommets reste $N = 2^n - 1$ et, les noeuds étant numérotés de 0 à N-1, les connections entre les noeuds sont modifiées comme suit :

Chaque noeud i ($i \in [0, N-1]$) est connecté aux noeuds $(i \pm \omega_k)$ MOD N, les ω_k étant déterminés par :

$$\forall\, k \in \{1\} \cup [3, n-1]\ ,\ \omega_k = 2^{k-1}\ .$$

(Il s'agit du graphe AFM "amputé" des arcs i--(i±2))

Un arbre de recouvrement est déterminé par :

- Racine : noeud 0 considérée comme le niveau 0 de l'arbre;

- Pour tout noeud i ($i \in [0, N-1]$) au niveau j

pour j∈ [0, n - 4]

son fils gauche est $(i - 2^{n-2-j}) \text{ MOD } N$

son fils droit $(i + 2^{n-2-j}) \text{ MOD } N$

pour j = n - 3

son fils gauche est $(i - 1) \text{ MOD } N$

son fils droit $(i + 1) \text{ MOD } N$

pour j = n - 2

son fils gauche est $(i - 1) \text{ MOD } N$ ou $(i - 4) \text{ MOD } N$

son fils droit $(i + 4) \text{ MOD } N$ ou $(i + 1) \text{ MOD } N$

Définit-on bien ainsi un recouvrement?

En fait, il s'agit du recouvrement AFM sauf aux deux derniers niveaux.

Pour un noeud quelconque i au niveau n-3 le sous-arbre de recouvrement de racine i est le suivant :

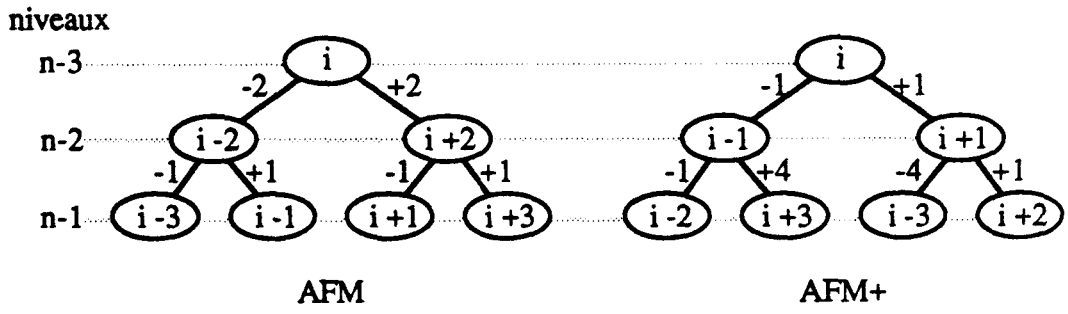


figure 4.36

On retrouve bien les mêmes noeuds quoique permutés.

En résumé :

-- jusqu'au niveau (n-3) le recouvrement AFM+ est identique au recouvrement AFM qui , jusqu'à ce niveau, ne fait pas intervenir les liaisons (i±2) qui sont absentes dans le réseau AFM+ ;

-- aux niveaux suivants, (n-2) et (n-1), le recouvrement AFM+ ne fait intervenir que des liaisons effectivement existantes dans le graphe AFM+ et recouvre les mêmes noeuds que le recouvrement AFM.

Conclusion : le recouvrement AFM+ est valide.

Remarque :

Au niveau (n-2), les fils gauche et droit d'un noeud i sont définis

soit par (i-1) MOD N
(i+4) MOD N (DG)

soit par (i-4) MOD N
(i+1) MOD N (DD)

Le problème qui se pose est le suivant : comment choisir simplement entre ces deux définitions?

Si i est le fils gauche de son père (de niveau n-3) alors on utilise la définition (DG)
Si i est le fils droit de son père alors on utilise la définition (DD).

Exemple :

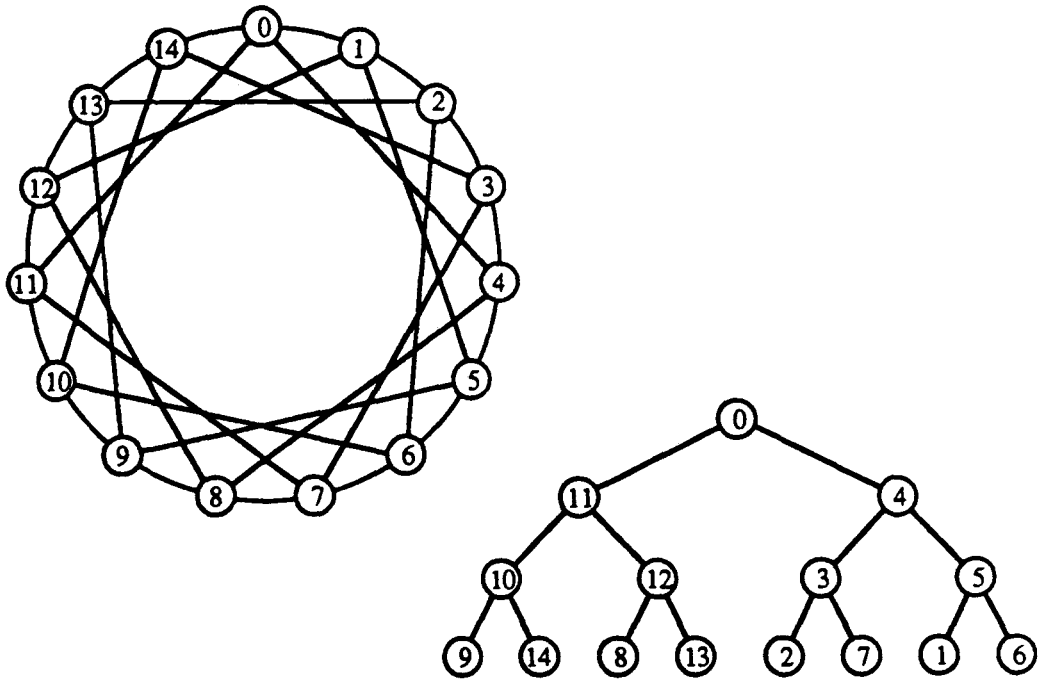


figure 4.37

4.3 Récapitulation

Les principales caractéristiques des réseaux vus ci-dessus et pouvant convenir pour le projet N-ARCH sont réunies dans le tableau ci-dessous (entre parenthèses, les valeurs correspondant, pour chaque réseau, à une taille comparable à celle de l'hypercube

généralisé de la figure 4.19) {entre accolades, les valeurs correspondant, pour chaque réseau, à une taille comparable à celle de l'hypercube généralisé de la figure 4.21.a}

Réseau	Hypercube	AFM	AFM+	HyperCubeGénéralisé	
Nombre de noeuds	$N = 2^n$ (64) {32}	$N = 2^n-1$ (63) {31}		N quelconque (63)	{40}
Degré	$d = n$ (6) {5}	$d = 2(n-1)$ (10) {8}	$d = 2(n-2)$ (8) {6}	(10)	{8}
Diamètre	$D = n$ (6) {5}	(3) {3}		suivant décomposition (3)	{3}
Recouvrement	Arbre binaire à double racine	Arbre binaire			Arbre ternaire
Extensibilité	Par puissances de 2			Non	

Tableau 4.2

CHAPITRE 5
SIMULATION.

La simulation a pour but d'étudier le comportement du réseau d'interconnexion, c'est-à-dire qu'on s'intéressera essentiellement à la circulation des messages et à la partie communication des noeuds.

5.1 Le simulateur

5.1.1 Présentation générale

La simulation est faite pour deux réseaux : l'hypercube muni du recouvrement en arbre binaire à double racine d'une part, le réseau AFM recouvert par un arbre binaire complet d'autre part.

La technique de hachage simulée est à deux niveaux, le traitement des collisions étant fait par un hachage local utilisant l'arbre de recouvrement du réseau considéré.

Pour étudier le comportement du réseau, on simule une exécution de programme avec communication par messages. Cette simulation est faite à un niveau macroscopique: en effet, c'est bien seulement le *comportement* du réseau qui nous intéresse ici, et non pas le détail du déroulement de l'exécution. En conséquence le *contenu* proprement dit des messages est ici sans signification. Seules les caractéristiques du message intervenant dans le routage à travers le réseau interviennent dans le déroulement de l'exécution simulée. Ces caractéristiques sont :

* le type du message : initialisation, requête ou résultat;

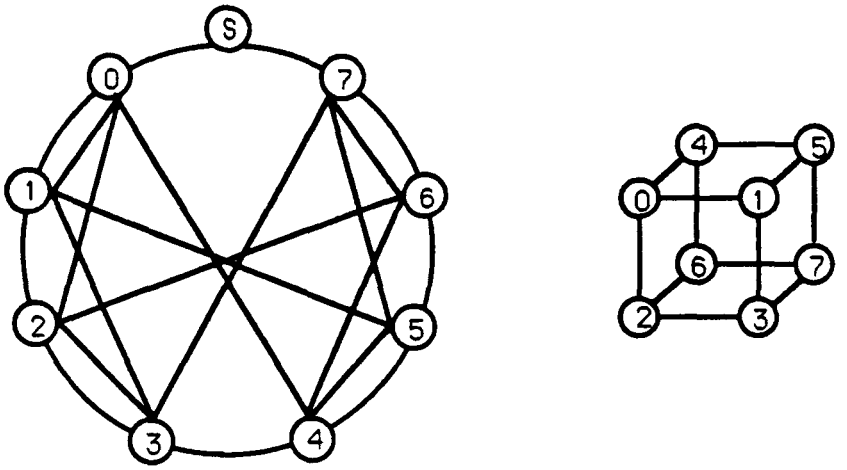
* le noeud destinataire;

* le mode de routage : direct ou arborescent;

* le niveau atteint dans l'arbre de recouvrement.

Le simulateur est écrit en MODULA2. Il consiste en un anneau de processus (au sens MODULA2 du terme), chacun d'entre eux représentant un noeud du réseau. Un noeud supplémentaire S est ajouté sur l'anneau : c'est celui-ci qui dirige la simulation et rassemble les résultats. La topologie effective du réseau dont on désire étudier le comportement (réseau AFM, hypercube, ... éventuellement d'autres réseaux ...) est plaquée indépendamment sur l'anneau, par adjonction de cordes qui représentent les liens dans le réseau.

Exemple : La figure 5.1 représente la configuration de l'anneau-simulateur (5.1.a) dans le cas de la simulation d'un 3-cube (hypercube à 2^3 noeuds) (5.1.b).



.a .b

figure 5.1

5.1.2 Les messages : génération, vie et mort

5.1.2.1 "Big Bang"

A l'instant initial (temps 0) de la simulation chaque noeud génère un certain nombre de messages dont la quantité est choisie au hasard entre 0 et MaxMess0, le destinataire et le type respectifs de ces messages étant également déterminés aléatoirement. MaxMess0 est un paramètre de la simulation.

5.1.2.2 Essaimage

A partir de cet instant-là les messages vont parcourir le réseau, se dirigeant tout d'abord par le chemin le plus court jusqu'à leur noeud destinataire (c'est le routage direct).

Arrivé à destination un tirage aléatoire permet de déterminer, en fonction d'une probabilité de présence voulue par les conditions de simulation (ceci représente en fait la prise en compte du taux de collision associé à la fonction de hachage utilisée), si le message va être traité sur place ou redirigé.

En cas de redirection, la même opération se reproduit à chaque nouveau noeud traversé par le message parcourant l'arbre de recouvrement ayant pour racine le destinataire initial du message (c'est le routage arborescent).

5.1.2.3 Génération et mort

Dans le cas où le message doit être traité, il "meurt", ayant au préalable assuré une éventuelle "descendance": le traitement amène en effet la génération d'un nombre de message(s) compris entre 0 et une valeur donnée MaxMess, cette valeur dépendant à la fois :

- du type de message traité (requête, résultat, initialisation)

- de l'instant de ce traitement au cours de la simulation.

A leur tour ces messages vont parcourir le réseau ...etc...(cf. 5.1.2.2).

5.1.2.4 Modalités de génération

On considère que l'exécution du programme produit un nombre total de messages déterminé, soit NbMaxMess ce nombre.

En cours de simulation, à chaque instant le nombre MaxMess de messages "générables" est régi par la courbe de la figure 5.2, où sont représentés en abscisse le nombre total de messages générés depuis le début de la simulation et en ordonnée le nombre maximal de messages pouvant être générés.

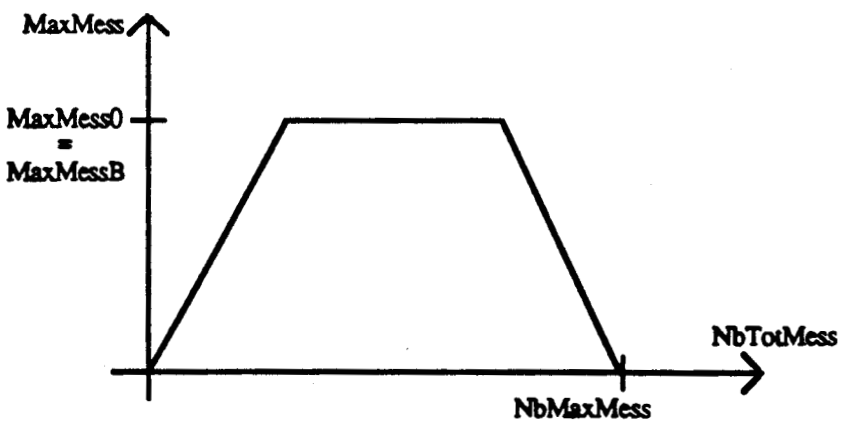


figure 5.2

Ce qui, en fonction du temps, nous donne la courbe suivante :

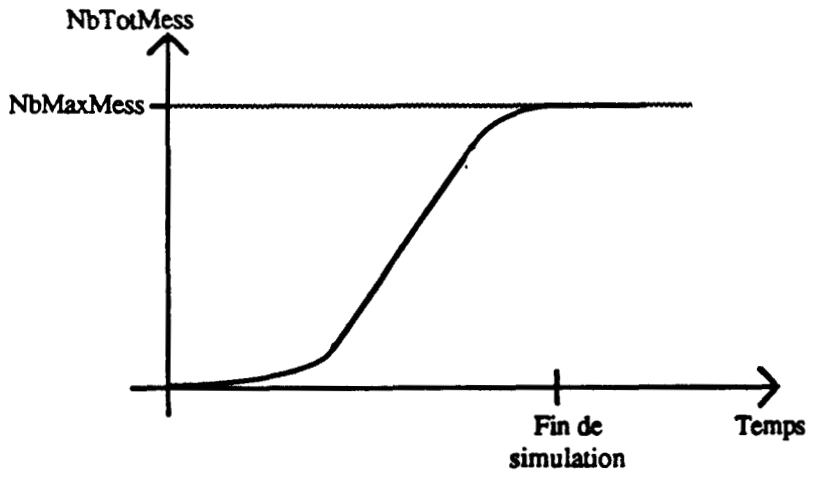


figure 5.2 bis

Le programme et ses données sont répartis au départ sur tous les processeurs du réseau grâce à la fonction de hachage global. C'est ce qui est simulé en créant un certain nombre de messages dans chaque noeud lors du "Big Bang", qui servent en fait à faire démarrer de manière artificielle l'exécution. MaxMess0 donne une sorte de charge initiale. La courbe 5.2 est ensuite bornée par une valeur MaxMessB qui, pour la simulation, a été prise égale à MaxMess0 mais aurait pu être différente.

A chaque génération de messages, le nombre de messages effectivement généré est déterminé aléatoirement entre 0 et MaxMess.

Parmi ces messages, la répartition des types entre initialisation, requête et résultat est de 10% initialisation et 90% (requête+résultat).

La répartition de la part de messages requête/résultat est régie par la courbe représentée par la figure 5.3 ci-dessous :

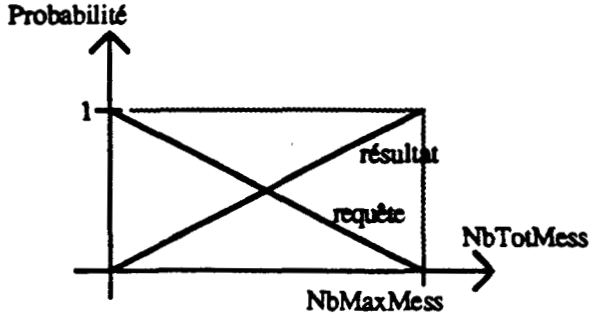


figure 5.3

Ceci correspond à l'exécution d'un programme avec évaluation par nécessité : au début, le nombre de messages croît lentement, la plupart sont des requêtes ; en milieu d'exécution le nombre de messages présents à chaque instant est plus important, la part requête/résultat tend à s'équilibrer ; en fin d'exécution, le nombre de messages diminue progressivement, les derniers étant essentiellement des messages de type résultat.

5.1.3 Routage

5.1.3.1 Routage direct

Hypercube

Les noeuds sont numérotés de telle sorte que les représentations binaires de deux noeuds voisins ne diffèrent que d'un seul bit.

La distance entre deux noeuds est alors égale au nombre de bits qui diffèrent dans la représentation binaire de leurs numéros respectifs.

Le routage direct est effectué grâce à l'algorithme suivant :

```
Algo
    {soit n0 le numéro du noeud recevant un message M dont le destinataire est le
    noeud de numéro n}
    Si n=n0
        Alors {M est arrivé à destination}
        Sinon {soit nb le nombre de bits communs entre n0 et n}
(*)         Transmettre M à l'un des voisins ni ayant nb+1 bits communs avec n
    FinSi
FinAlgo
```

En procédant ainsi, la distance au noeud destinataire est systématiquement diminuée de 1 à chaque nouveau noeud traversé. On est donc assuré d'emprunter l'un des plus courts chemins entre le noeud où a été généré le message et le noeud destinataire. Par contre, le routage direct entre deux noeuds donnés n'est pas unique (sauf dans le cas où les deux noeuds sont voisins). Loin d'être un inconvénient, ce fait peut être mis à profit pour répartir sur plusieurs chemins les communications entre deux noeuds non voisins : dans le simulateur le voisin n_i sélectionné en (*) l'est aléatoirement. On pourrait envisager

de le sélectionner en fonction de la charge de la liaison n_0 -- n_i. Ceci peut également être intéressant dans une optique "tolérance aux pannes".

Réseau AFM

Pour ce réseau, en l'absence d'algorithme de routage simple et efficace, le routage direct est effectué au moyen d'une table de routage déterminée empiriquement et dont une copie est présente en chaque noeud du réseau.

5.1.3.2 Routage arborescent

Hypercube

Ici encore, le routage se fait par l'intermédiaire d'une table de routage contenant l'arbre de recouvrement de racine 0, dont une copie est détenue par chaque noeud.

Soit n_e un noeud détenant un message M dont le destinataire initial est n_d, M devant être réexpédié en routage arborescent (on peut éventuellement être dans le cas $n_e=n_d$). L'arbre de recouvrement que M doit parcourir est donc l'arbre de racine n_d.

Le noeud n_e effectue les opérations suivantes:

1/ appliquer la fonction de hachage locale qui détermine si le message doit être redirigé vers le fils droit ou le fils gauche ;

2/ calculer, à partir de l'arbre de recouvrement A_0 de racine 0, l'arbre de recouvrement A_d de racine n_d. Cette opération est simple : il suffit de remplacer chaque noeud $n \in A_0$ par le *OU exclusif* entre n et n_d (cette opération pourrait être effectuée en parallèle sur tous les noeuds) ;

3/ recherche dans A_d du fils (droit ou gauche) déterminé en 1/ (éventuellement recherche associative) ;

4/ envoi de M au noeud voulu.

Réseau AFM

De par la définition du réseau et de l'arbre de recouvrement la simplicité du calcul des fils d'un noeud donné rend inutile l'utilisation d'une table de routage. Pour un noeud donné n_0 au niveau k dans l'arbre de recouvrement, ses fils gauche et droit sont respectivement (n_0-2^{n-2-k}) MOD N et (n_0+2^{n-2-k}) MOD N où $N=2^n-1$ désigne le nombre de noeuds du graphe. La connaissance de la racine de l'arbre de recouvrement est inutile, seul le niveau atteint dans l'arbre intervient dans le calcul. En fait, de toute façon, le noeud

racine de l'arbre de recouvrement figure dans le message puisqu'il est le destinataire initial du message, et le niveau y figure également puisqu'il intervient dans le calcul de la fonction de hachage locale.

5.1.4 Simulation du fonctionnement d'un noeud

Pour la simulation, un noeud est vu de la manière suivante:

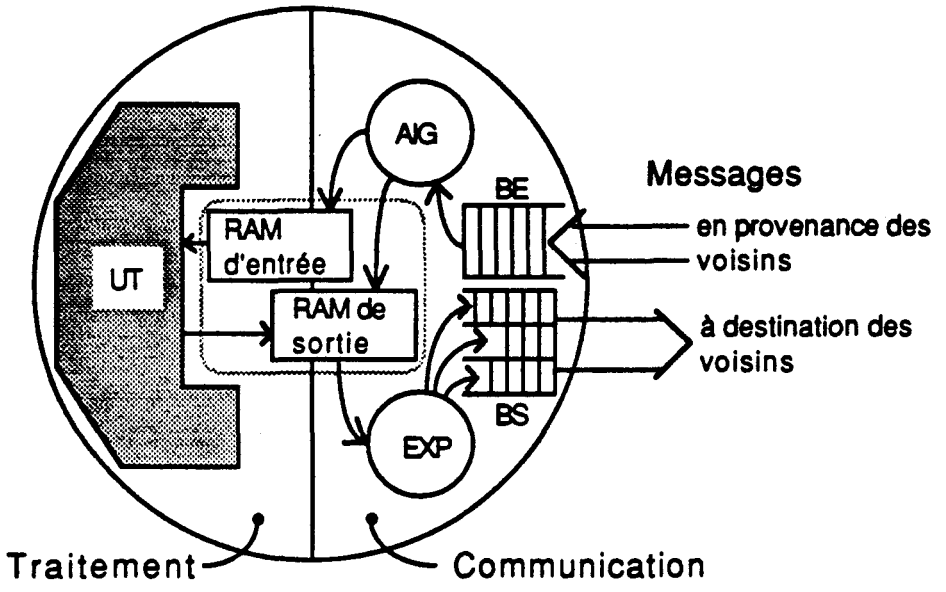


figure 5.4

Les messages arrivent en BE, depuis les voisins du noeud.

A chaque unité de temps :

* AIG extrait au plus MC messages de BE. Chacun de ces messages est dirigé vers une des RAM :

- la RAM d'entrée si le message a pour destinataire le noeud considéré ou si le message est en cours de routage arborescent;

- la RAM de sortie si le message a pour destinataire un autre noeud.

* UT extrait MT messages de la RAM d'entrée et les traite. Ce traitement amène la génération de 0 à MaxMess messages, dirigés vers l'une des RAM.

* EXP extrait au plus MC messages de la RAM de sortie et les dirige vers les buffers de sortie BS correspondant aux voisins destinataires ou par lesquels ils doivent transiter.

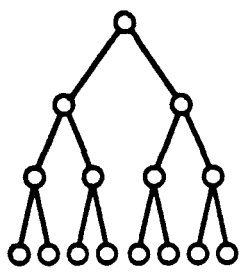
* Un message est expédié sur chacun des liens reliant le noeud à ses voisins.

5.1.5 Paramètres

5.1.5.1 Taux de collision

Ils sont représentés par des probabilités de présence associées à chaque niveau de l'arbre de recouvrement. Celles-ci permettent de tester le comportement du réseau pour diverses fonctions de hachage caractérisées par des taux de collision différents.

Sept possibilités sont prédéfinies (toute autre série de probabilités pouvant être entrée au clavier) ; le tableau ci-dessous définit les probabilités (en %) de présence par niveau dans l'arbre de recouvrement, pour chacune de ces sept possibilités.



	1	2	3	4	5	6	7
0	100	75	66	50	33	25	0
1	(100)	94	88	75	55	50	0
2	(100)	98	96	87	70	75	0
3	(100)	100	100	100	100	100	100

Tableau 5.1

5.1.5.2 Temps de traitement et communication

Ils sont modifiables par l'intermédiaire des paramètres MC et MT, correspondant respectivement aux nombres de messages expédiés vers les voisins et traités dans le noeud. Les courbes présentées plus loin correspondent à MT=1 ou 2 et MC=6.

5.1.5.3 Comportement du programme

On peut influer sur le comportement du programme dont on simule l'exécution par l'intermédiaire des paramètres NbMaxMess, MaxMess0, MaxMess qui permettent de simuler une charge plus ou moins importante.

5.1.5.4 Topologie

Les deux types de réseau définis en 4.2 sont disponibles, les résultats ayant été obtenus pour un hypercube de 16 noeuds et un réseau AFM de 15 noeuds.

5.2 Les résultats

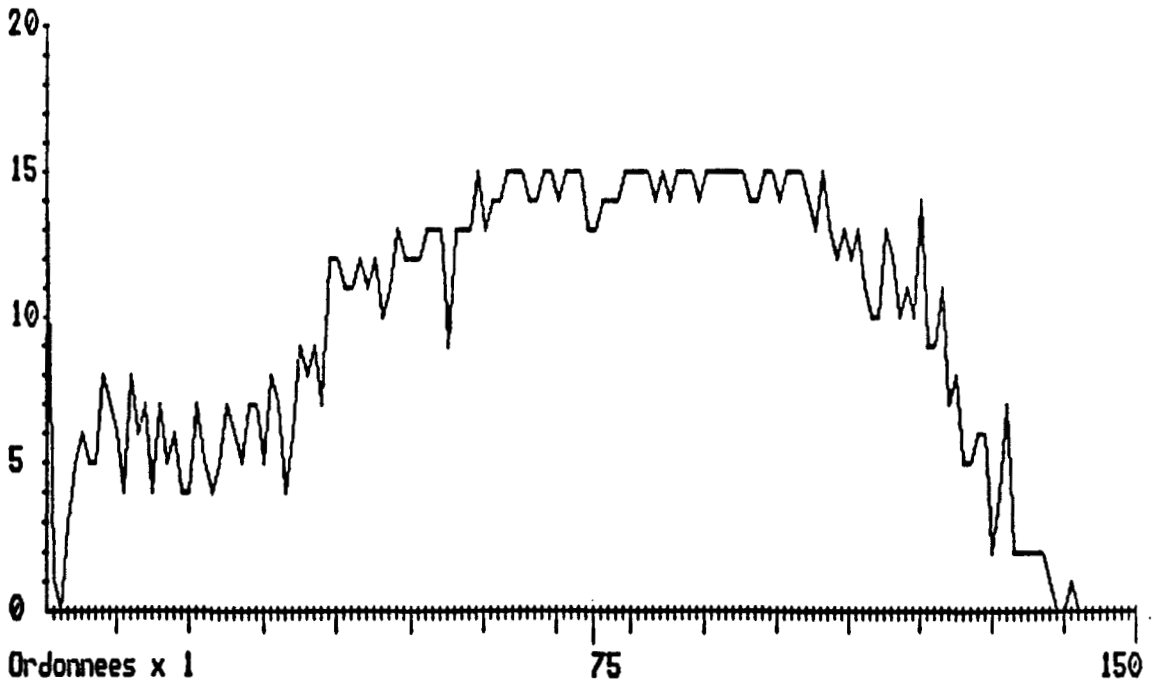
La simulation a été effectuée pour plusieurs séries de valeurs des différents paramètres.

5.2.1 Activité des processeurs

Une première classe de courbes représente l'activité des processeurs. La notion d'activité est entendue au sens du traitement : un processeur qui, pendant une certaine période, ne fait que transmettre des messages est, à ce sens, considéré comme inactif.

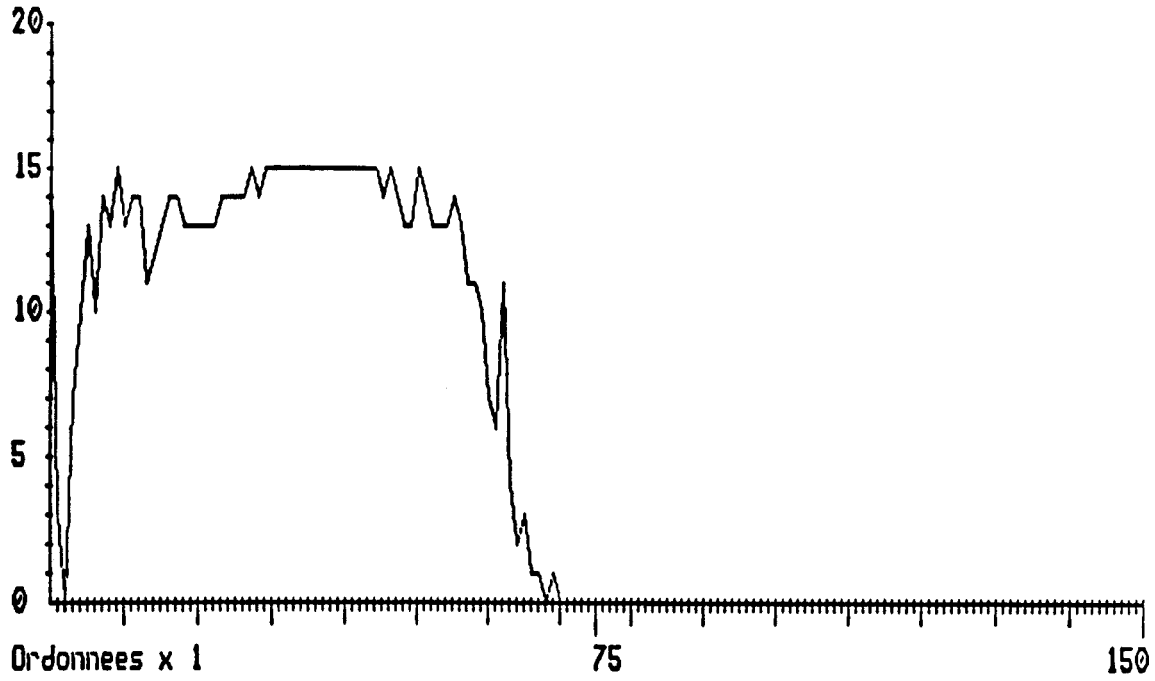
Les temps de traitement/communication considérés sont les suivants : en prenant comme référence le temps de transmission d'un message sur un lien entre deux noeuds voisins, soit t_u ce temps, le temps de communication interne (c'est-à-dire le temps de transmission sans traitement d'un message) est $t_c = t_u / 6$, le temps de traitement quant à lui est $t_t = t_u / 2$ ou $t_t = t_u$.

On peut voir ci-dessous quelques exemples de ces courbes, pour lesquelles on trouve en abscisses : le temps ; en ordonnées : le nombre de processeurs actifs.



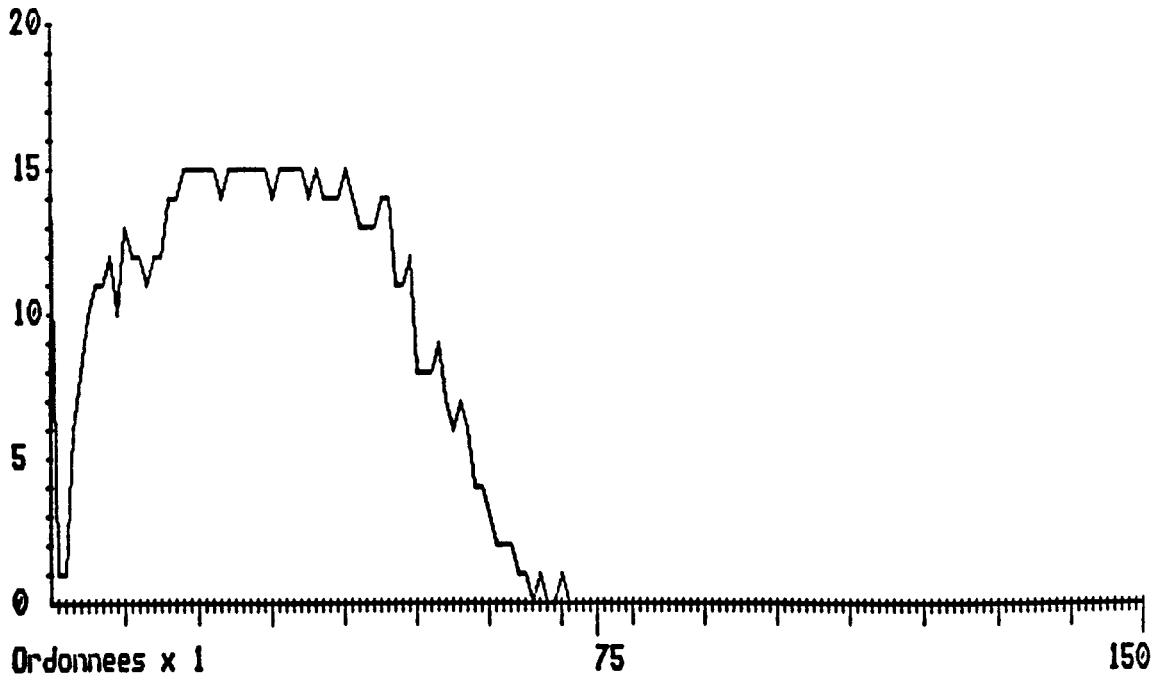
Taux de collision : 7

$t_t = t_u$



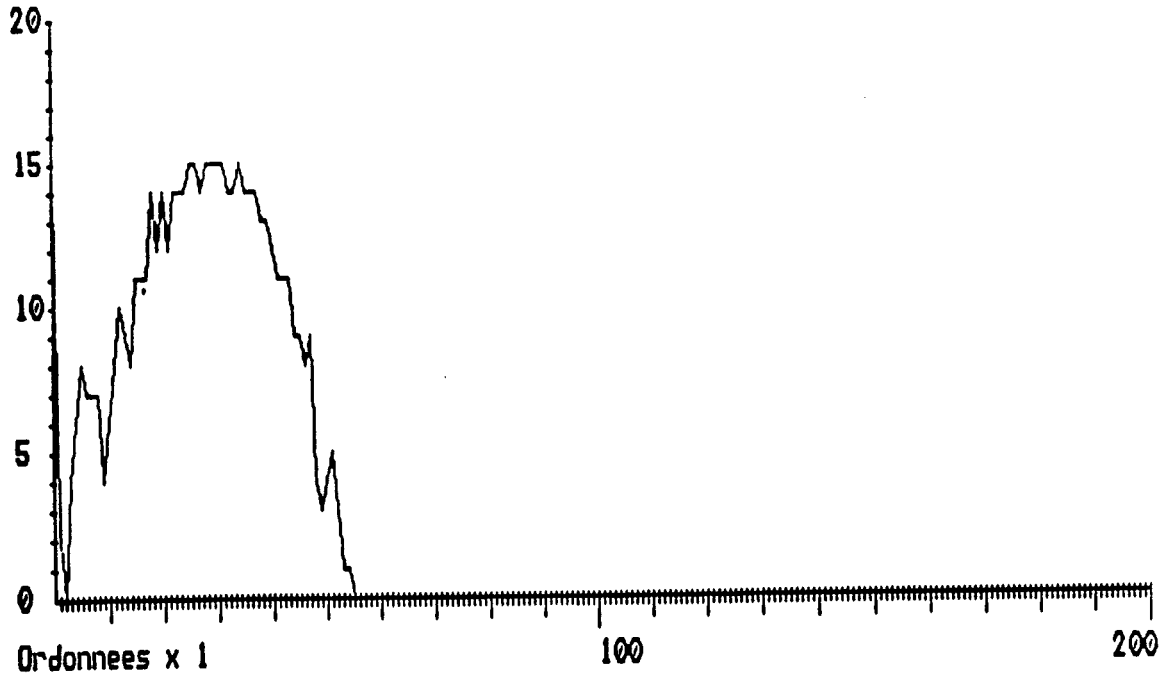
Taux de collision : 7

$t_l = t_u / 2$



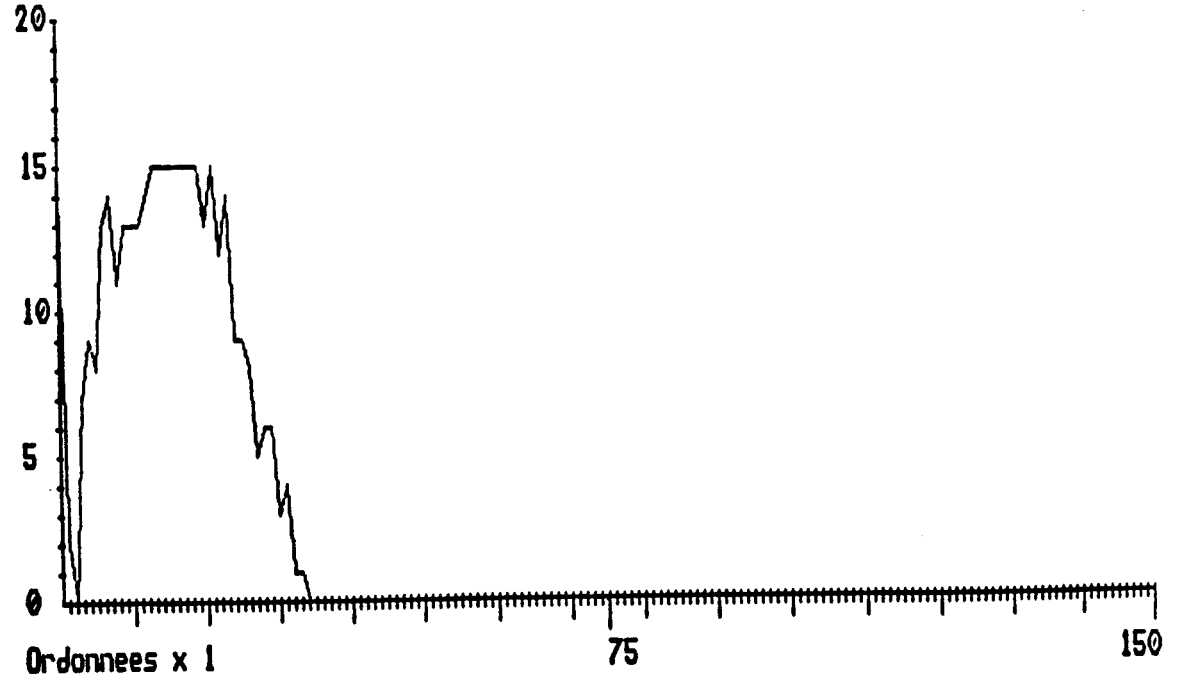
Taux de collision : 6

$t_l = t_u$



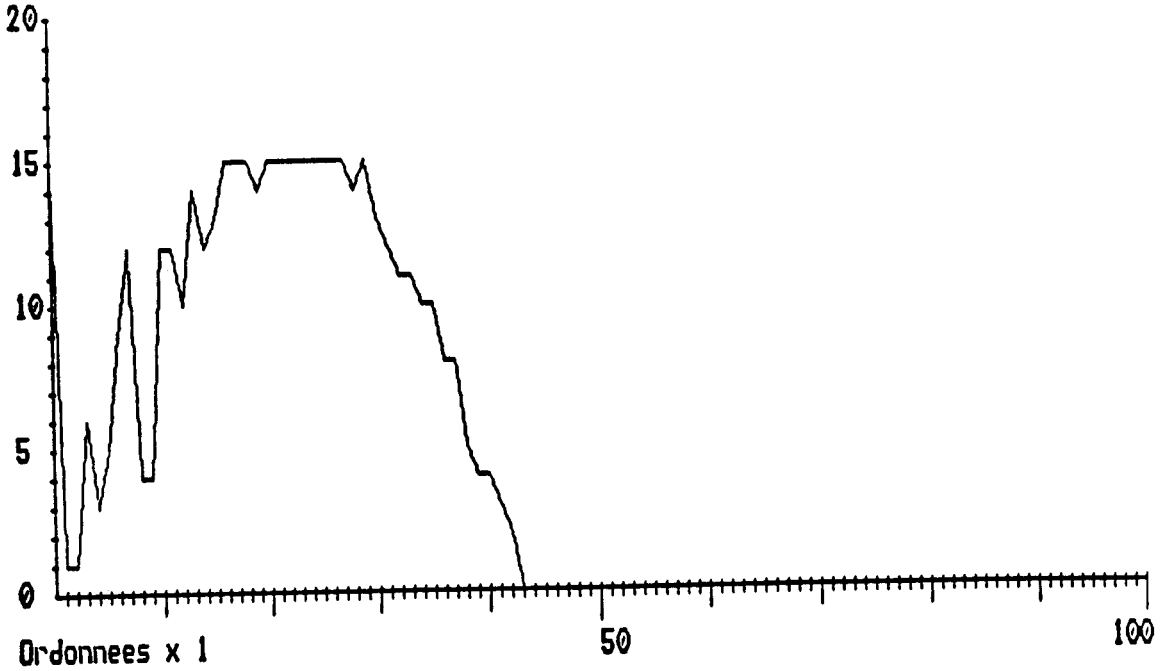
Taux de collision : 4

$t_t = t_u$



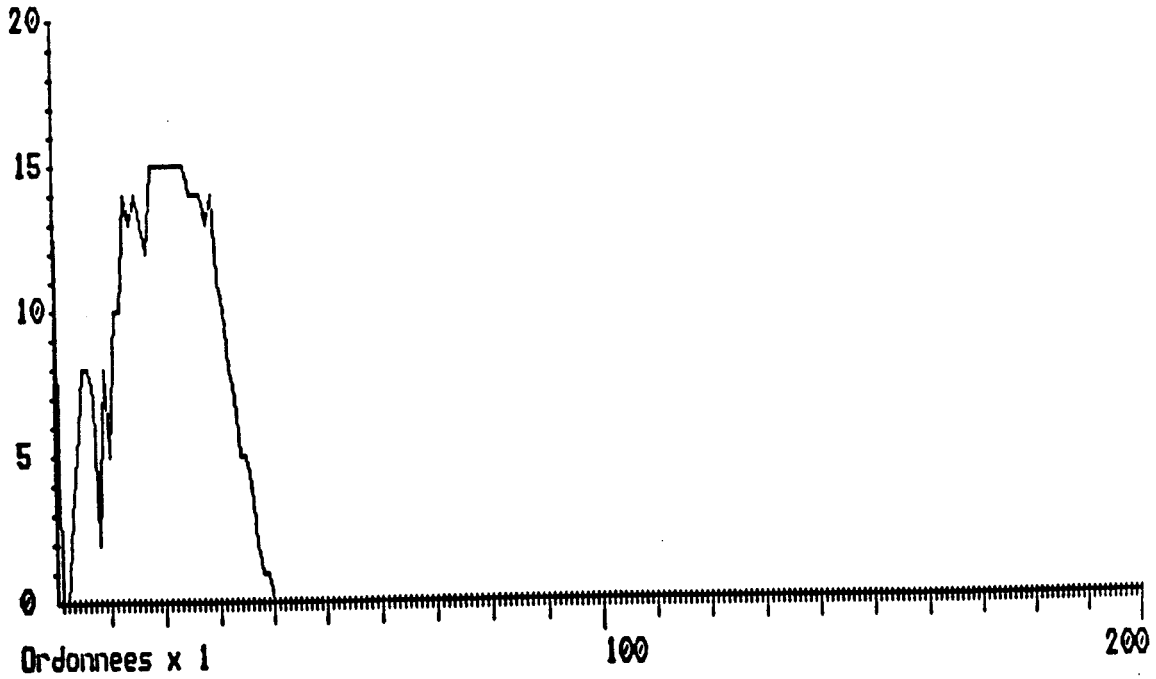
Taux de collision : 4

$t_t = t_u / 2$



Taux de collision : 2

$t_t = t_u$



Taux de collision : 1

$t_t = t_u$

Il n'y a pas de différence significative, pour ces courbes, entre les résultats obtenus d'une part pour l'hypercube, muni du recouvrement en arbre binaire à double racine, et d'autre part le réseau AFM, avec un recouvrement en arbre binaire.

On constate dans tous les cas une phase d'amorçage d'autant plus longue que les messages issus du "Big Bang" arrivent plus tard sur les lieux de leur traitement (taux de collision élevé). Vient ensuite une phase de rendement maximal où tous les noeuds sont actifs ; puis une phase de dégradation des performances en fin de simulation : elle correspond au retour des ultimes résultats et il y a donc de moins en moins de traitements à effectuer.

La phase à rendement maximal est évidemment plus longue lorsque le taux de collisions est élevé. On obtient donc dans ce cas des taux de parallélisme plus élevés dans l'absolu, comme on peut le voir sur la courbe de la figure 5.5, qui représente le taux de parallélisme (en %) en fonction du taux de collision (en %). En abscisses, la deuxième ligne de chiffres renvoie aux colonnes du tableau 5.1. On entend ici par taux de parallélisme le rapport : (nombre moyen de processeurs actifs) / (nombre total de processeurs).

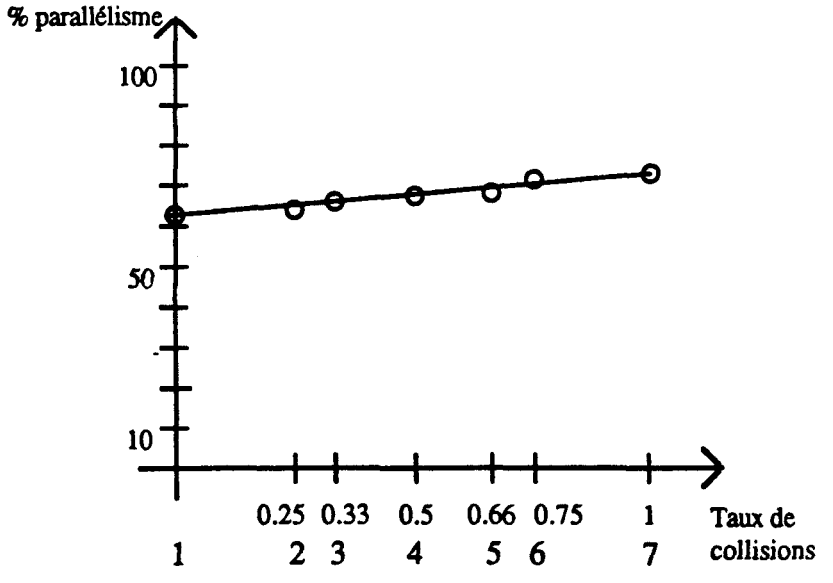


figure 5.5

Mais, rapporté au temps d'exécution, plus long lui aussi lorsque le taux de collision est plus élevé, les résultats deviennent plus "normaux" :

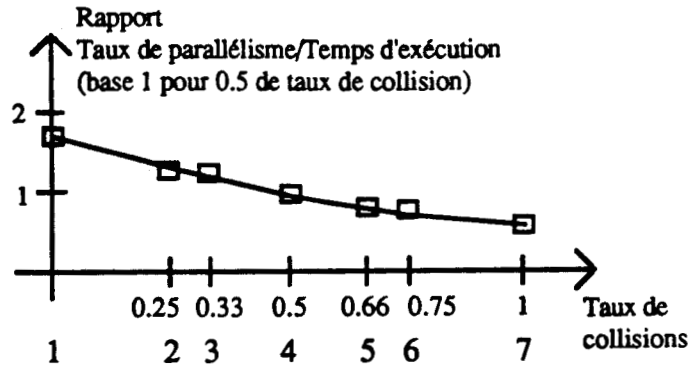


figure 5.6

L'accélération du temps de traitement est répercutée sur le temps d'exécution de manière amortie : pour un même taux de collisions, le passage de $t_t = t_u$ à $t_t = t_u / 2$ réduit au maximum de moitié le temps d'exécution. Ceci signifie que l'accroissement de l'encombrement des liens, provoqué par l'augmentation de la production de messages par la partie traitement plus rapide, est absorbée par la partie communication et le réseau sans provoquer de ralentissement notable. Cette réduction du temps d'exécution est mieux répercutée lorsque le taux de collisions est élevé : le faible ralentissement provoqué par le surcroît d'encombrement des liens est masqué par l'accroissement du nombre de messages présents à chaque instant dans le réseau du fait des redirections après collision.

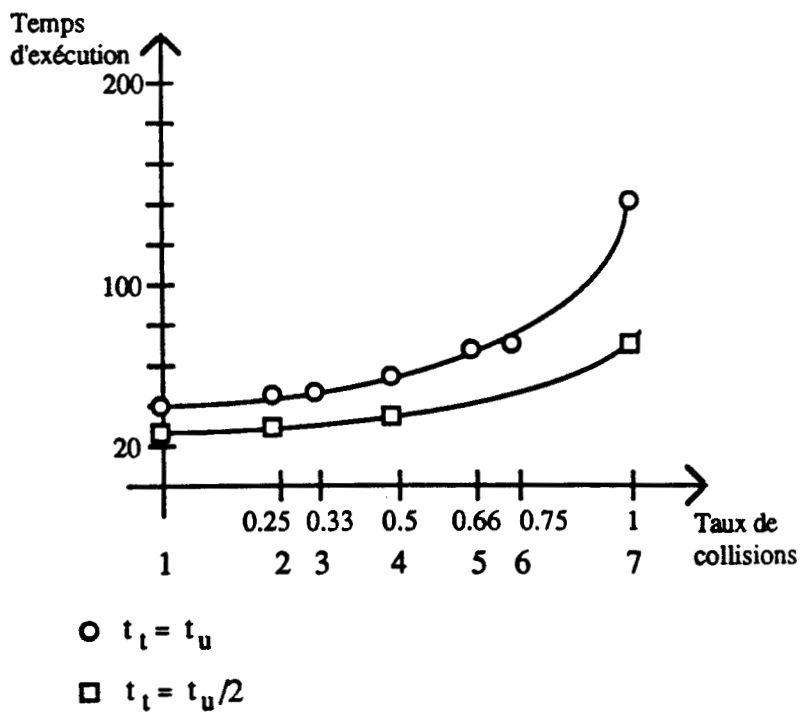
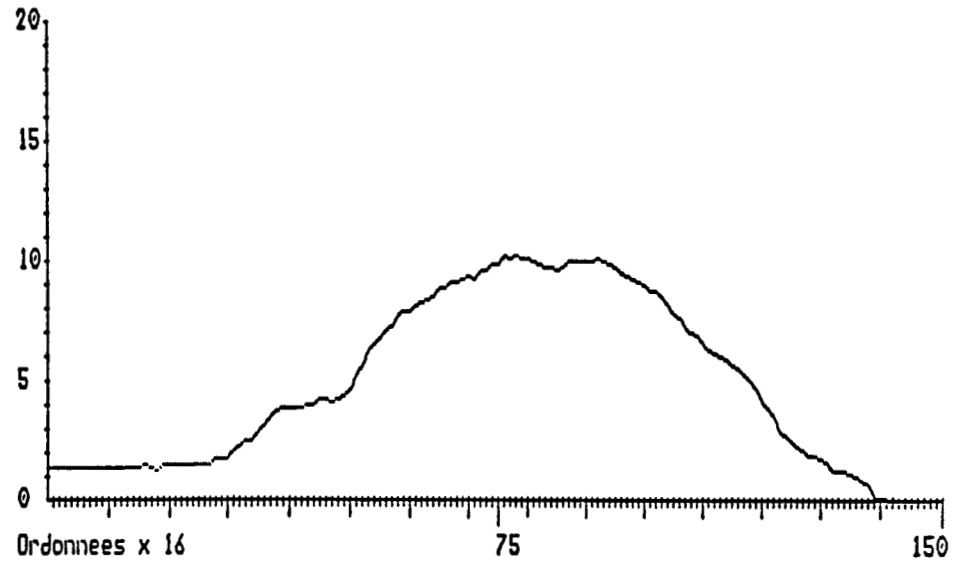


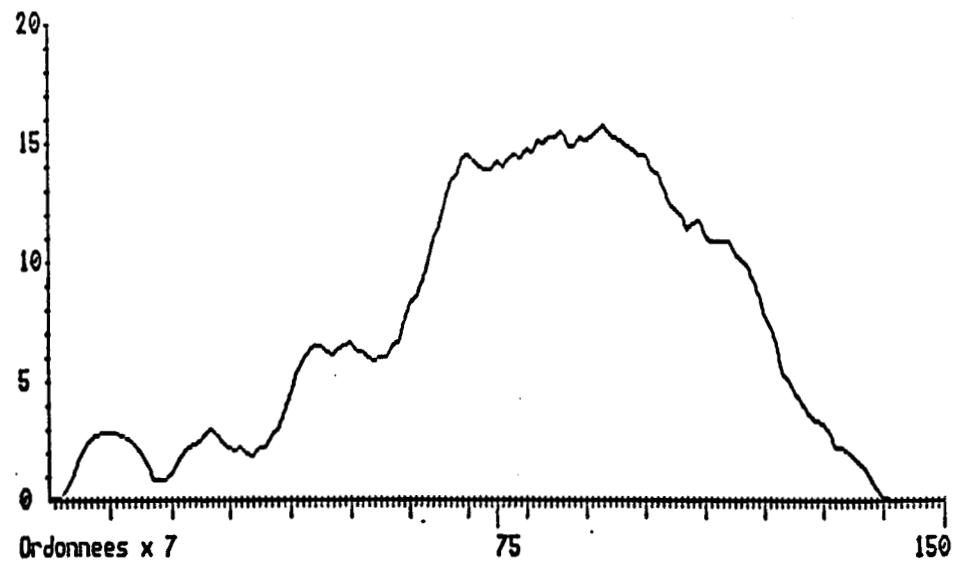
figure 5.7

5.2.2 Messages

Une deuxième classe de courbes, présentées ci-dessous, donne la quantité de messages présents à chaque instant dans le réseau et, parmi ceux-ci, ceux qui sont en cours de routage arborescent. On retrouve là aussi l'influence du taux de collisions (courbes "Nombre total de messages" et "Nombre de messages en routage arborescent" ci-dessous, pour lesquelles on trouve en abscisses : le temps ; en ordonnées : le nombre de messages).



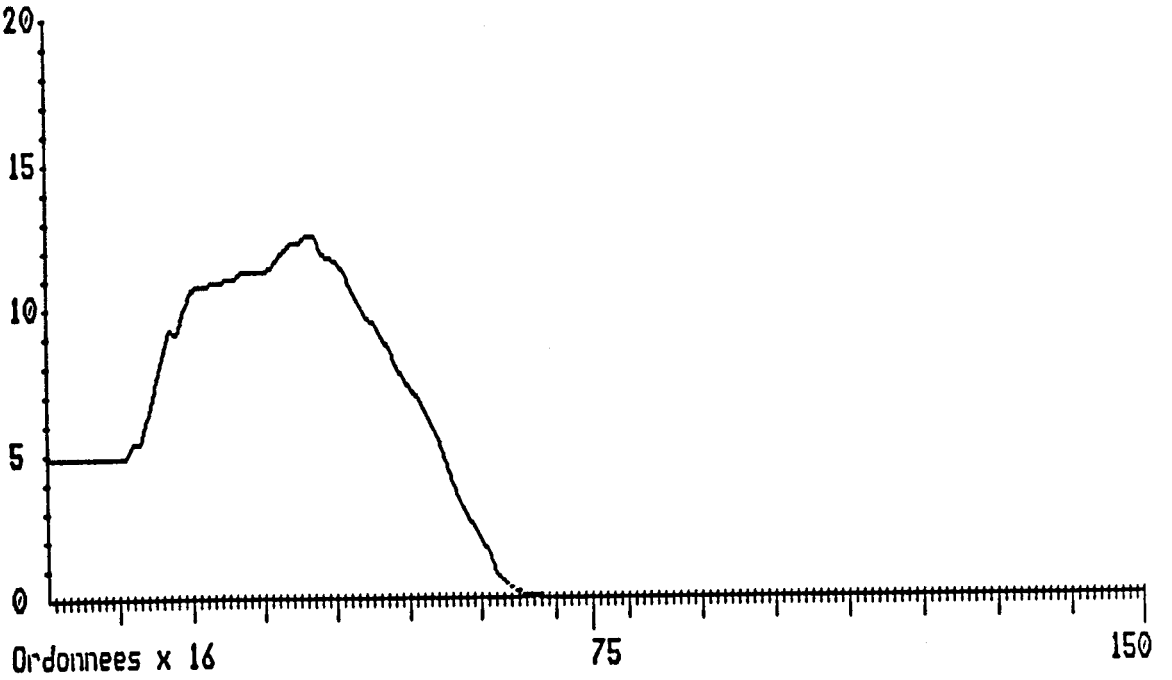
Nombre total de messages



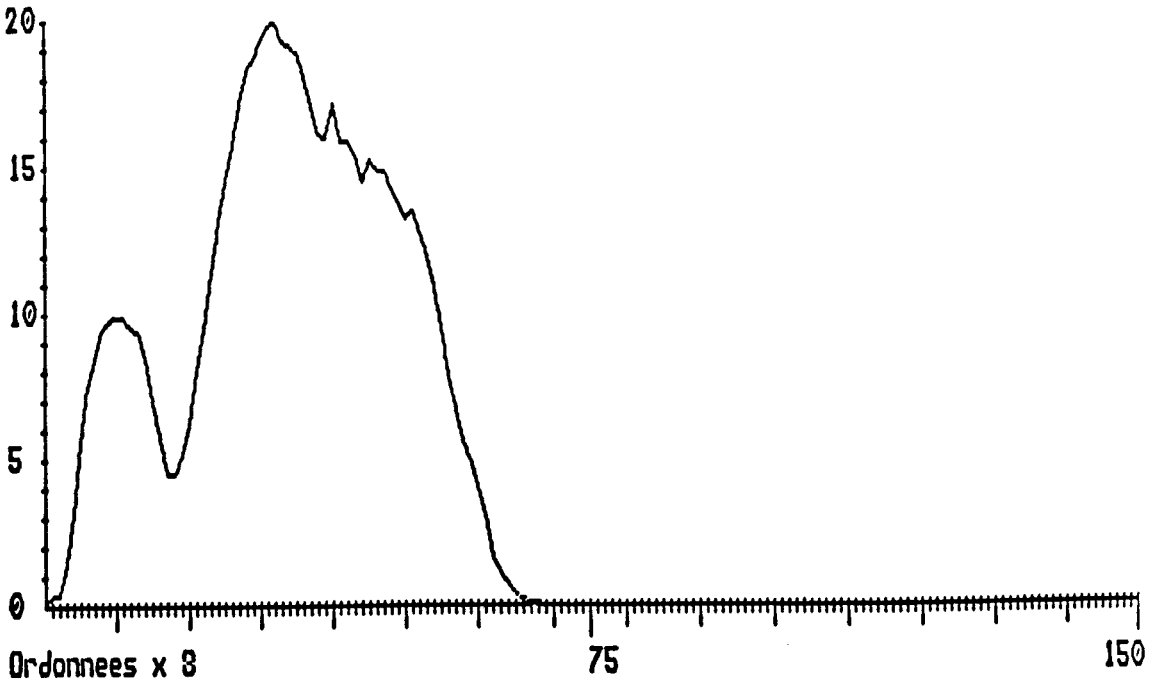
Nombre de messages en routage arborescent

Taux de collision : 7

$t_t = t_u$



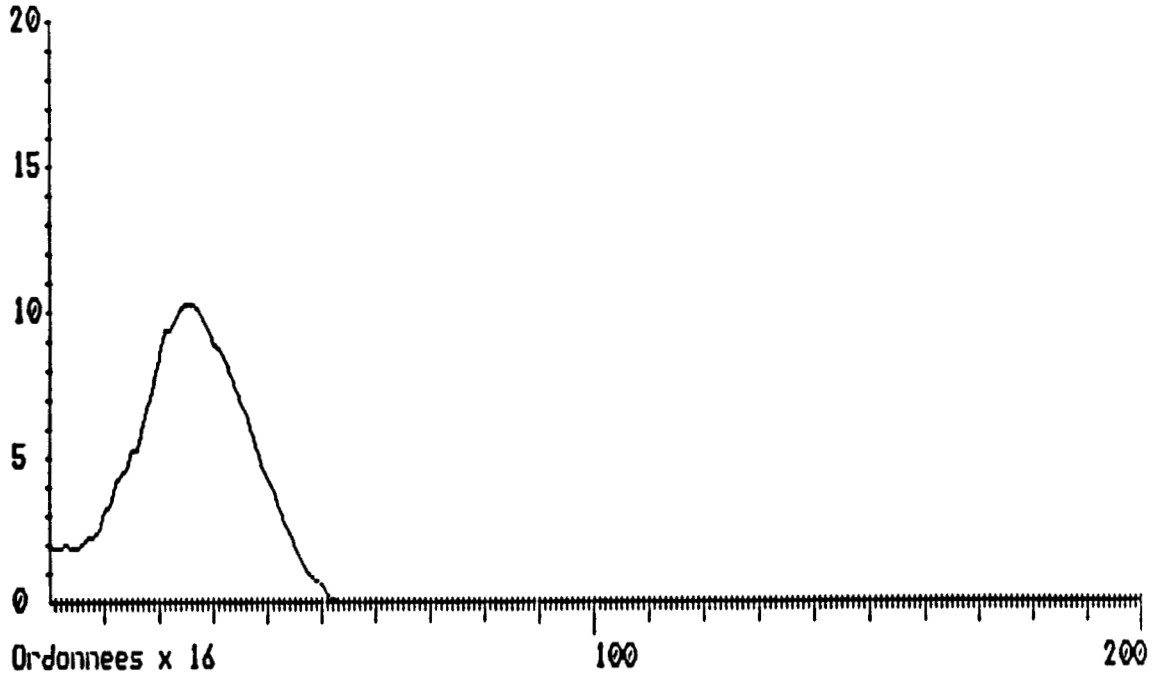
Nombre total de messages



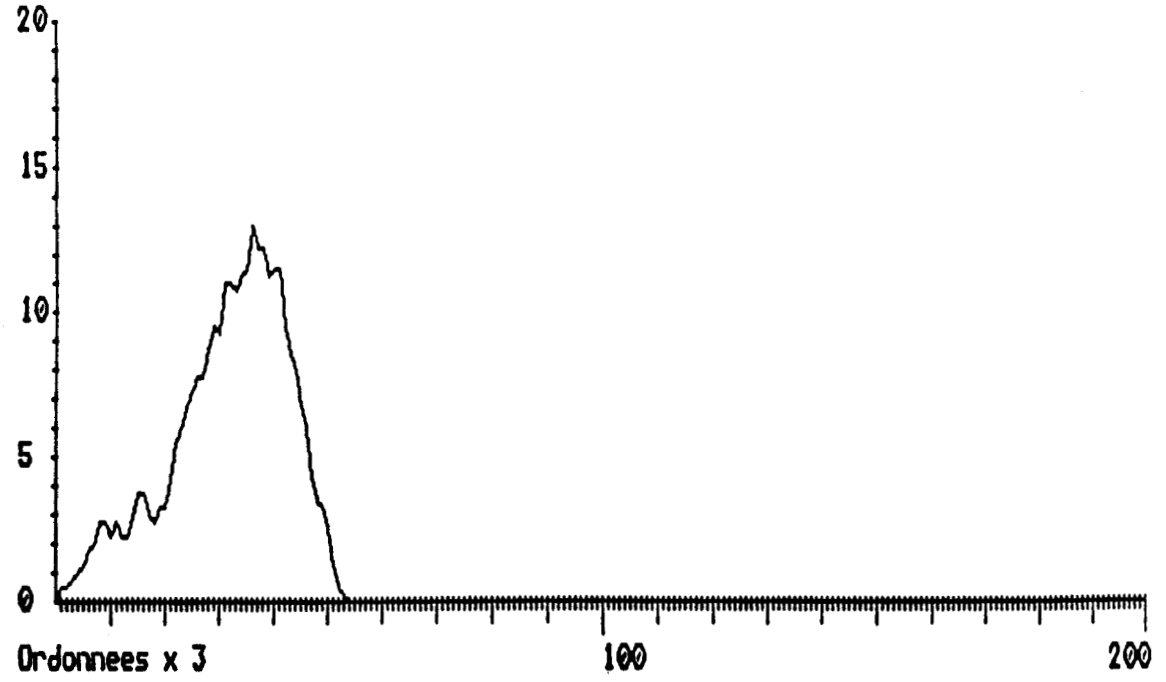
Nombre de messages en routage arborescent

Taux de collision : 7

$t_i = t_u / 2$



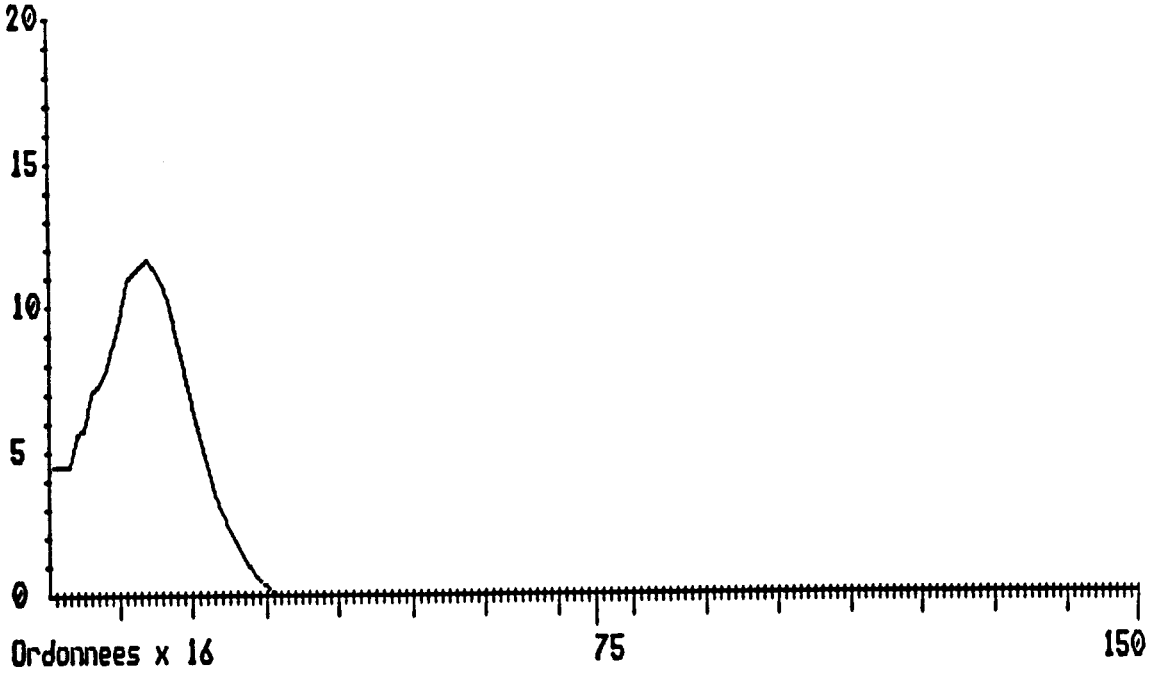
Nombre total de messages



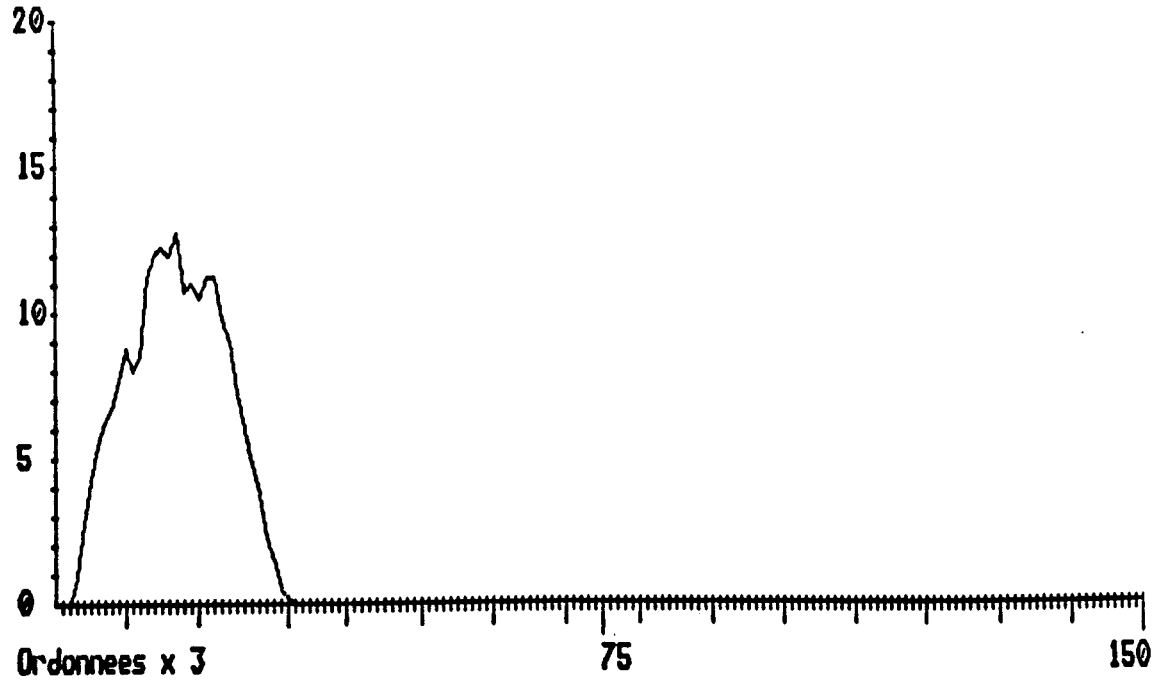
Nombre de messages en routage arborescent

Taux de collision : 4

$t_i = t_u$



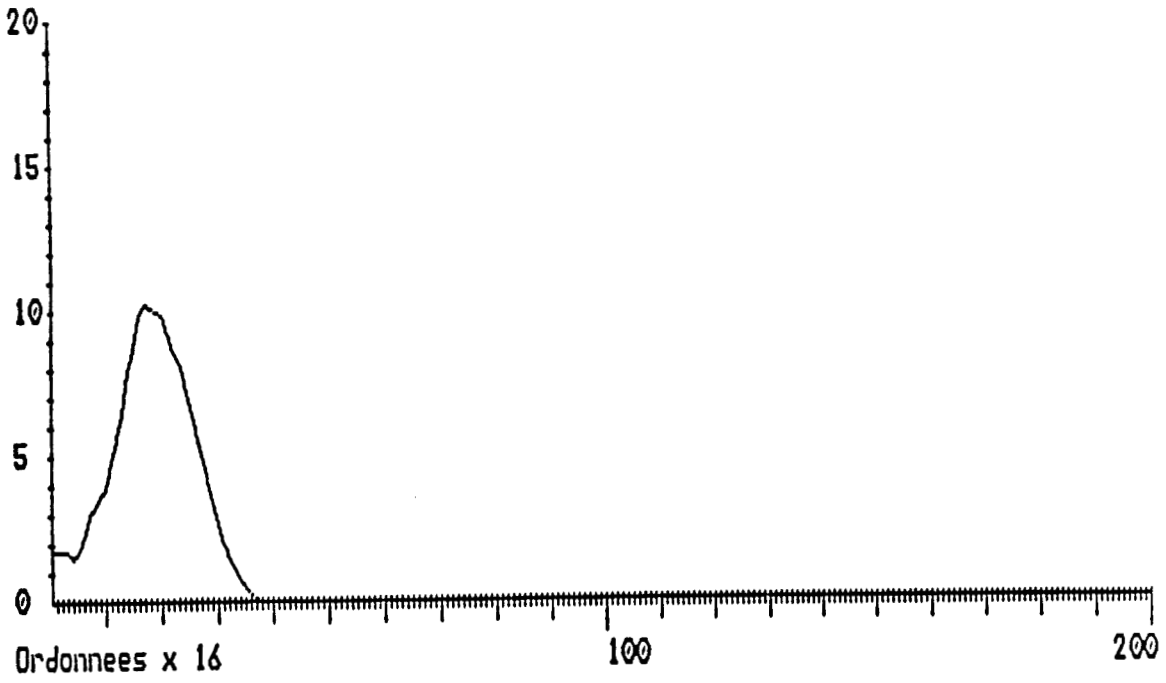
Nombre total de messages



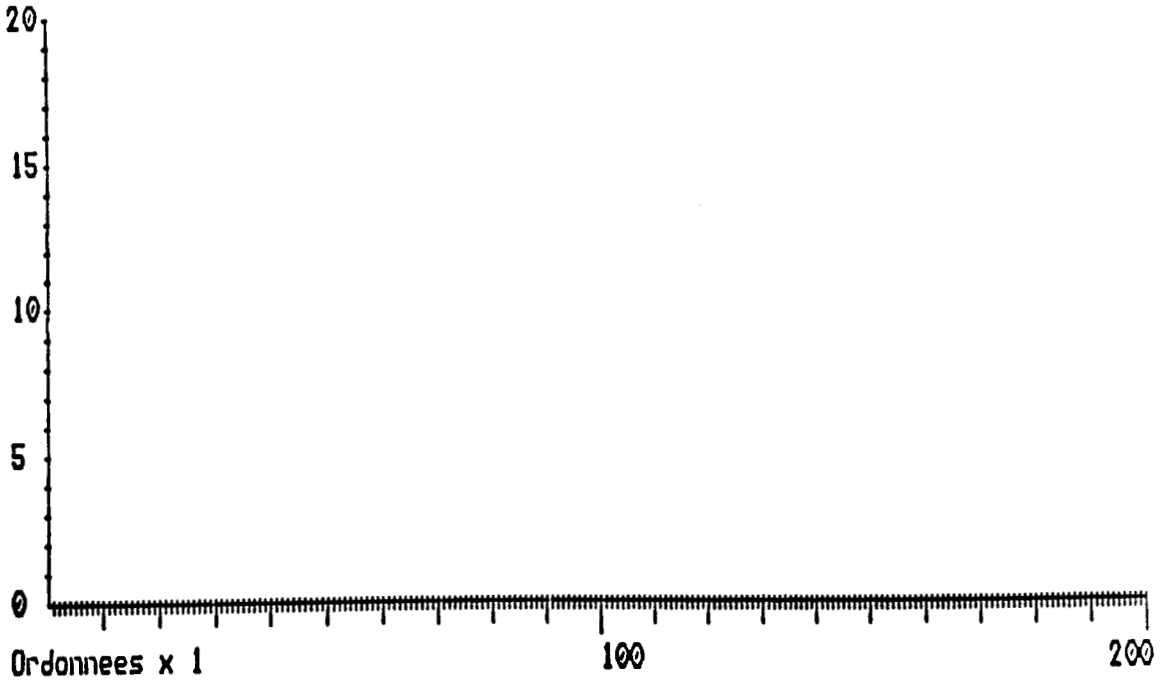
Nombre de messages en routage arborescent

Taux de collision : 4

$t_i = t_u / 2$



Nombre total de messages



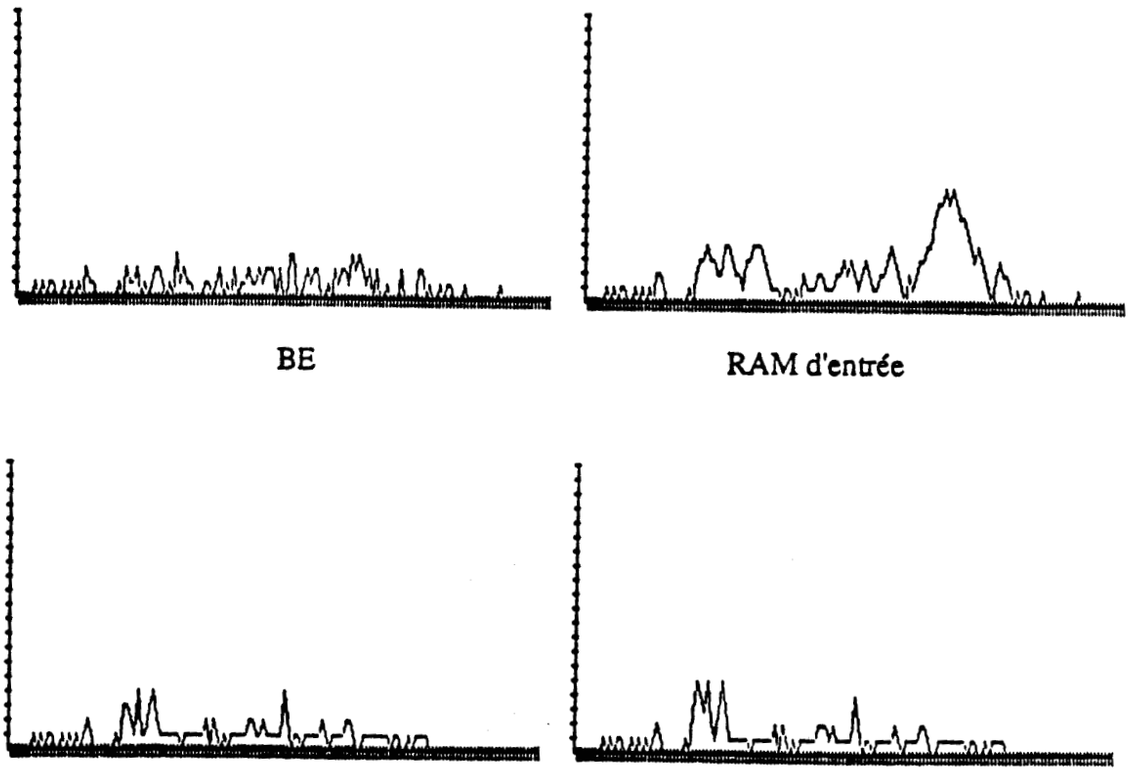
Nombre de messages en routage arborescent

Taux de collision : 1

$t_i = t_u$

5.2.3 Taux d'occupation

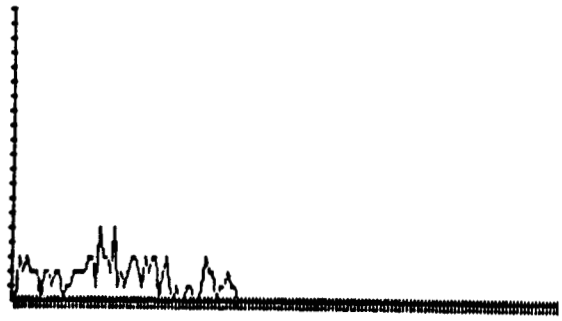
D'autres résultats permettent de se rendre compte du remplissage des mémoires et divers buffers de chaque noeud. Pour des taux de collisions raisonnables, c'est-à-dire inférieur à 0.5, ces derniers n'ont jamais été saturés. Leur capacité pour la simulation était de 16 messages, ce qui est assez faible devant la quantité totale de messages pouvant être présents à chaque instant dans le réseau . Ces résultats sont présentés ci-dessous par les courbes "Remplissages des buffers et RAM". Sur ces courbes, on trouve en abscisses : le temps ; en ordonnées : le nombre de messages.



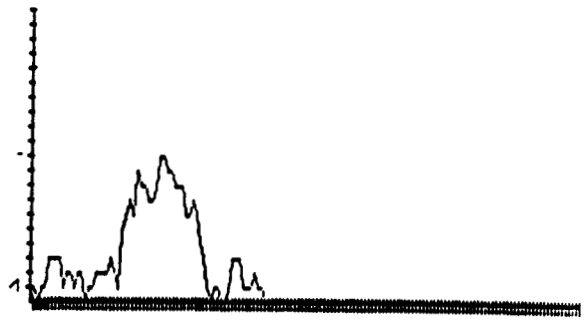
Remplissage des buffers et RAM

Taux de collision : 7

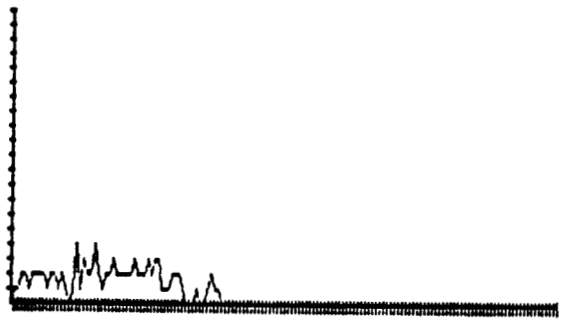
$t_t = t_u$



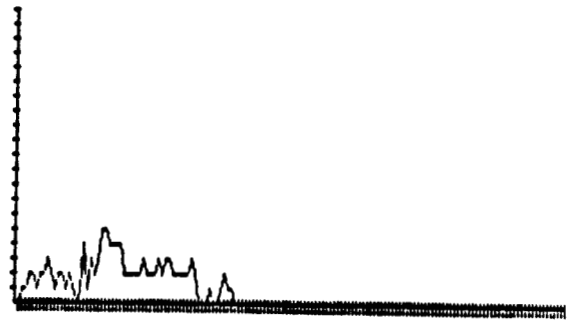
BE



RAM d'entrée



RAM de sortie

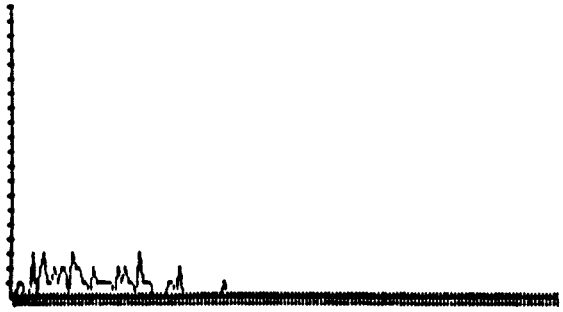


BS

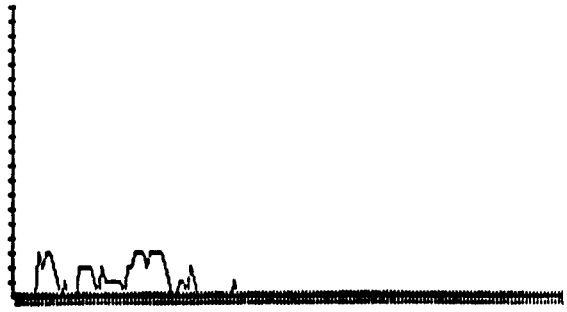
Remplissage des buffers et RAM

Taux de collision : 7

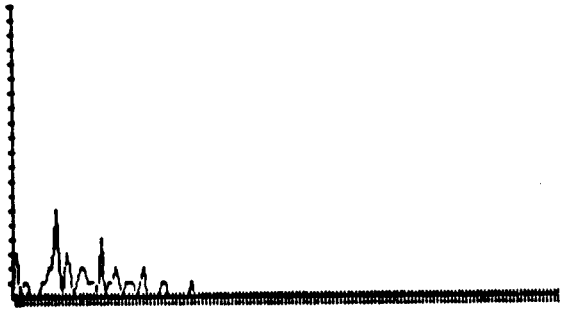
$t_t = t_u / 2$



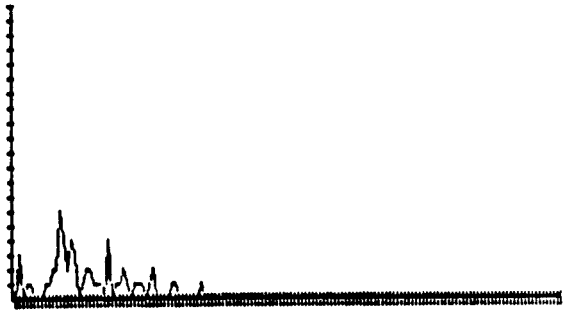
BE



RAM d'entrée



RAM de sortie

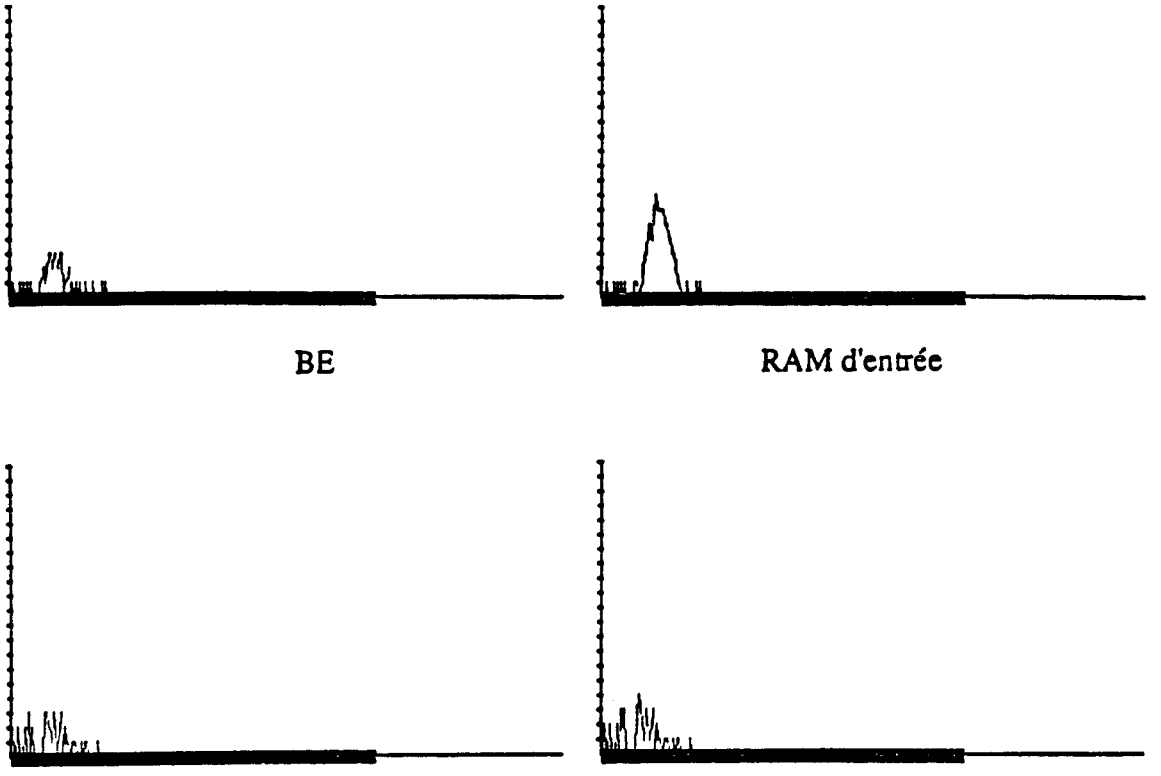


BS

Remplissage des buffers et RAM

Taux de collision : 6

$t_t = t_u$



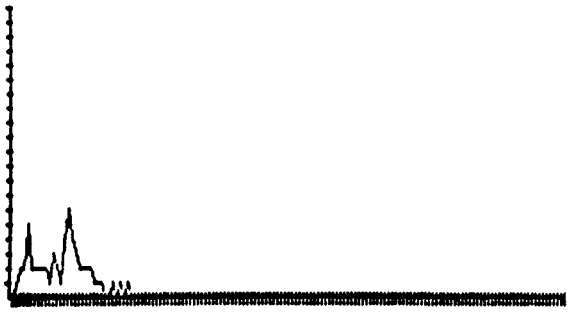
BE RAM d'entrée

RAM de sortie BS

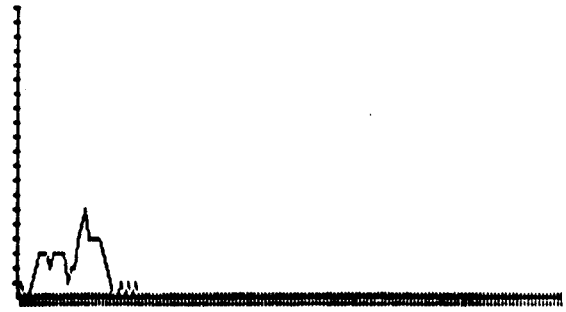
Remplissage des buffers et RAM

Taux de collision : 4

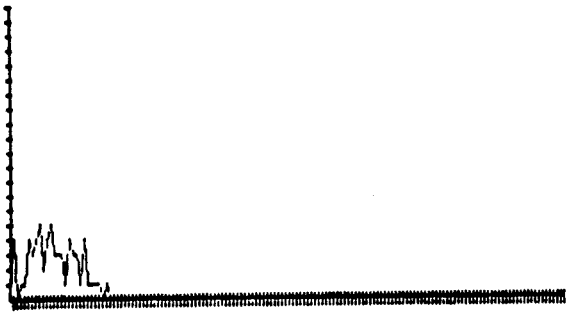
$t_t = t_u$



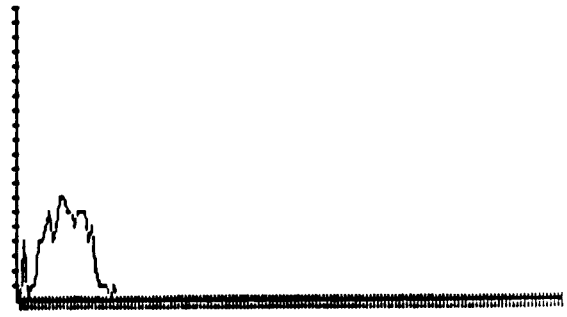
BE



RAM d'entrée



RAM de sortie

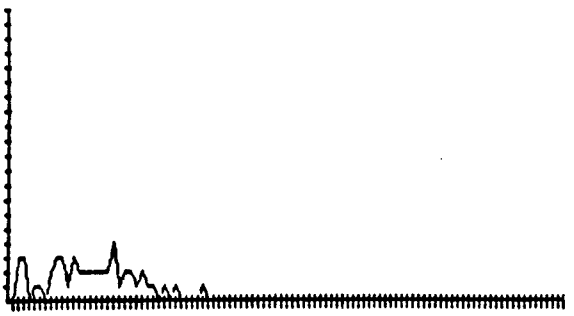


BS

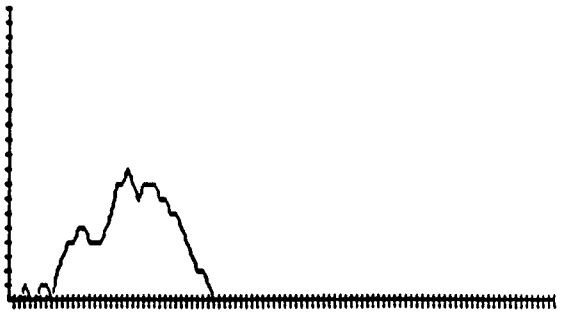
Remplissage des buffers et RAM

Taux de collision : 4

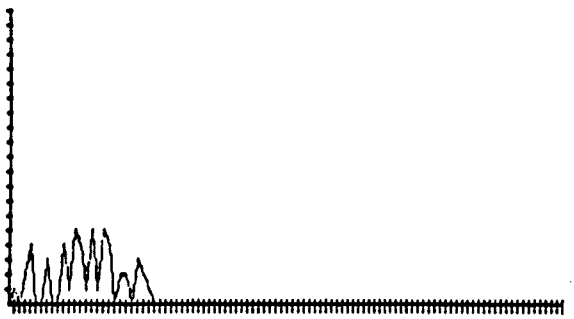
$t_t = t_u / 2$



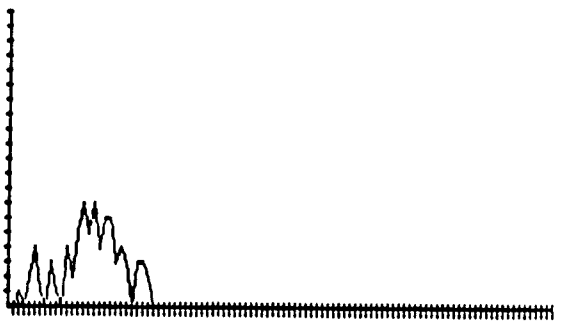
BE



RAM d'entrée



RAM de sortie



BS

Remplissage des buffers et RAM

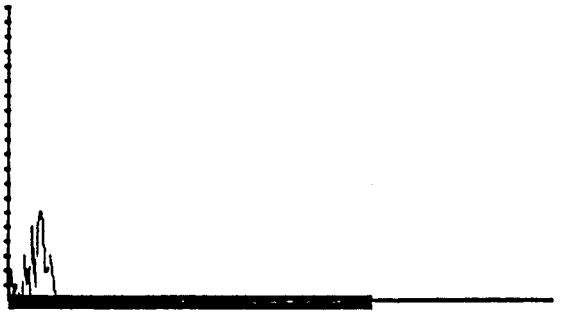
Taux de collision : 2

$t_l = t_u$

BE

RAM d'entrée

RAM de sortie



BS

Remplissage des buffers et RAM

Taux de collision : 1

$t_t = t_u$

5.2.4 Conclusion

Les résultats obtenus indiquent que la technique de hachage à deux niveaux avec sa répartition des redirections est valable : on n'a jamais observé de saturation locale, même pour des taux de collisions élevés. De plus les taux de parallélisme obtenus sont importants.

La dégradation des performances en temps d'exécution lorsque le taux de collisions augmente indique toutefois que le choix de la fonction de hachage globale devra être fait avec le plus grand soin.

Une confirmation de ces résultats a été apportée par d'autres travaux effectués dans le cadre du projet N-ARCH. Les résultats de ces travaux sont présentés dans le chapitre suivant.

CHAPITRE 6
COMPARAISON ET ÉVALUATION.

Dans ce chapitre, nous présentons deux études effectuées dans le cadre du projet N-ARCH et dont les résultats sont à rapprocher de ceux présentés au chapitre précédent.

6.1 Emulation sur Transputer

Un émulateur du réseau N-ARCH a été réalisé au moyen de transputers et un noyau de la machine a été développé en OCCAM par S. Niar [NIAR89]. Une série de tests et de mesures de performances a également été réalisée.

6.1.1 Configuration matérielle

Le matériel utilisé pour réaliser les tests a été le suivant :

- un micro-ordinateur de type PC-AT, muni d'une carte Inmos B04 qui contient un transputer et joue le rôle d'interface avec le réseau. C'est aussi sur cette carte que se fait le développement et la mise au point des programmes OCCAM qui seront chargés plus tard sur le réseau.

- 4 cartes Inmos B03 : chacune de ces cartes contient 4 transputers munis chacun de 256 K de mémoire. Sur chaque carte, les 4 transputers sont configurés en anneau. On dispose ainsi de 8 liens libres par carte (2 par transputer). On peut donc travailler sur des hypercubes de degré 0 à 4 suivant le nombre de transputers et de cartes utilisés.

6.1.2 Méthode de test

Un langage de simulation a été défini, pour lequel un programme est constitué d'un ensemble d'expressions, elles-même constituées

- d'un **ensemble d'arguments** qui sont les paramètres nécessaires au calcul de l'expression ;
- d'un **délai**, qui représente le temps nécessaire au calcul effectif du résultat lorsque les arguments ont été reçus.

Afin de pouvoir simuler l'exécution de programmes de taille importante, un processus OCCAM génère automatiquement des programmes de test à partir d'un certain nombre de paramètres choisis par l'utilisateur.

Un programme peut être représenté par un arbre de dépendance et c'est cet arbre qui est réparti dans les différents noeuds du réseau au moyen de fonctions de hachage.

6.1.3 Résultats

Les résultats obtenus ici concernent essentiellement l'accélération de l'exécution, en appelant accélération le rapport (temps d'exécution sur n processeurs)/(temps d'exécution sur 1 processeur).

6.1.3.1 Courbes d'exécution sans collision

On s'est arrangé dans ce cas pour qu'aucune collision ne se produise : le hachage local n'intervient jamais et le temps de routage des messages est donc réduit. L'accélération a été mesurée pour 5 programmes de tailles différentes.Les accélérations obtenues sont représentées sur la figure suivante :

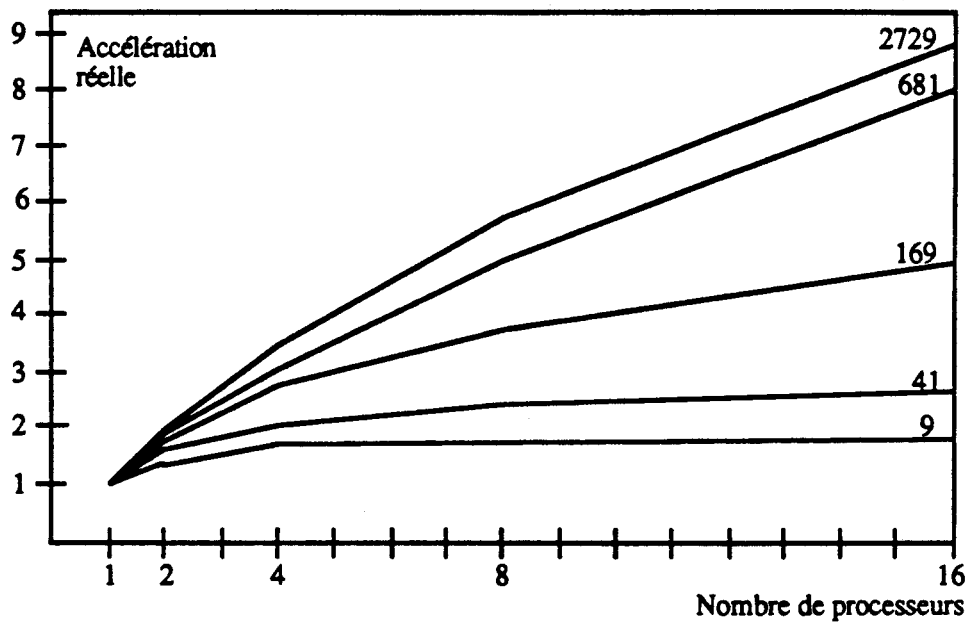


Figure 6.1

Sur cette figure, les nombres indiqués aux extrémités des courbes représentent la taille, en nombre d'expressions, des programmes.

On remarque que plus la taille du programme est importante, plus l'accélération est grande : si le nombre de processeurs est suffisant, l'apparition d'un plus grand nombre d'expressions évaluables en parallèle est exploitée et augmente le parallélisme.

Autre constatation : l'existence d'un seuil au-delà duquel l'introduction de davantage de processeurs n'apporte plus de gain. Il n'y a pas assez de parallélisme dans le programme pour occuper les processeurs supplémentaires.

Une autre série de mesures a été réalisée, toujours dans les mêmes conditions, c'est-à-dire en l'absence de collision. Elle a pour but d'évaluer l'influence de la granularité sur l'accélération. Pour cela on a fait varier le délai des expressions c'est-à-dire leur temps de calcul effectif (cf. 6.1.2). Comme on peut le constater sur la figure 6.2, l'augmentation du niveau de grain permet d'obtenir des accélérations plus importantes. Ceci s'explique par le fait qu'un niveau de grain plus important implique une quantité de traitement local également plus élevée et donc des communications faibles devant le temps d'exécution. En contrepartie, un niveau de grain plus important diminue le parallélisme de l'exécution et accroît la difficulté d'obtenir une bonne répartition de la charge entre les processeurs.

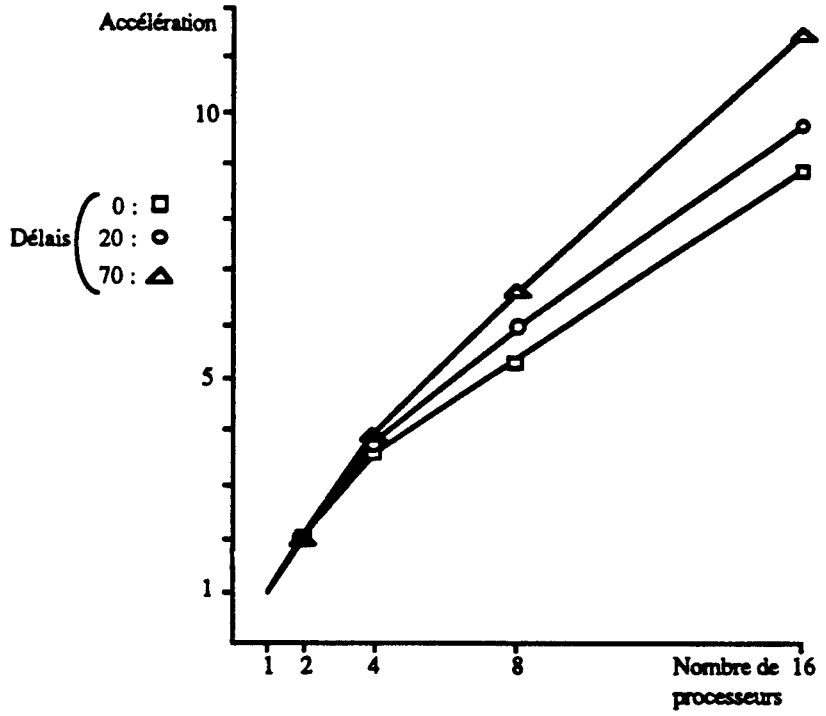


Figure 6.2

6.1.3.2 Exécution en présence de collisions

Pour la simulation d'exécution avec collisions, les résultats obtenus (tableaux 6.1 et 6.2) l'ont été avec un hypercube d'ordre 4 et les programmes, déjà évoqués ci-dessus, comportant respectivement 169 et 681 expressions. Comme on pouvait s'y attendre, ces résultats montrent un allongement du temps d'exécution avec le nombre de collisions et donc l'importance du hachage global dans la vitesse d'exécution. Dans les tableaux ci-dessous, la qualité de ce hachage apparait en première colonne : meilleur est le hachage global, plus grand est le nombre de processeurs sur lequel celui-ci répartit les expressions.

Nombre de processeurs			Accélération	Temps d'exécution	
adressés par le hachage global	utilisés	ne contenant que des expressions en collision		avec collisions	sans collisions
1	5	4	1	307	376
2	8	6	1.31	233	229
4	4	0	2.13	144	144
8	8	0	3.07	101	101
16	16	0	4.14	74	74

Tableau 6.1

Nombre de processeurs			Accélération	Temps d'exécution	
adressés par le hachage global	utilisés	ne contenant que des expressions en collision		avec collisions	sans collisions
1	16	15	1	1076	1542
2	16	14	1.12	954	879
4	16	12	1.49	719	512
8	16	8	2.33	460	314
16	16	0	5.60	191	191

Tableau 6.2

6.2 Simulation par événements significatifs

6.2.1 Description de la méthode de simulation

La méthode de simulation par événements significatifs [TARB70] a été mise au point à l'IMAG de Grenoble dans le but d'être intégrée à des outils de CAO de circuits intégrés. Elle peut toutefois être parfaitement utilisée dans d'autres cadres, et l'a été ici par J.C. Marti [MART89] pour simuler le fonctionnement, au niveau macroscopique, du réseau N-ARCH.

Cette méthode ne prend en compte que les événements que sont l'apparition ou la disparition des processus simulés et saute directement de l'un de ces événements, dits significatifs, à son suivant dans l'ordre chronologique. Il n'y a pas de découpage du temps en unités dont la longueur serait dictée par la durée de vie des différents processus entrant en jeu.

Cette méthode consiste en fait à générer une suite de couples (temps, événement) classée par ordre chronologique. Une tête de suite permet à tout instant l'accès à l'événement chronologiquement le plus proche.

Cet événement sera extrait puis analysé. Cette analyse permettra éventuellement de prévoir l'arrivée de nouveaux événements qui seront donc intercalés dans la suite à leurs places respectives dans l'ordre chronologique.

Les instants auxquels sont prévus les événements peuvent être fixes (exemple : commutation de circuit) ou aléatoires (exemple : arrivée ou extraction d'un message dans une file d'attente).

Le traitement de chaque événement comporte obligatoirement celui des interactions entre les processus, représentées par un graphe d'états des événements significatifs. Explicitons ce fonctionnement sur un exemple : la gestion d'une file d'attente (buffer d'entrée d'un noeud par exemple), pour laquelle deux événements significatifs seront retenus :

- arrivée d'un message dans la file ;
- extraction d'un message de la file.

Si, statistiquement, on peut prévoir un certain taux d'arrivée de messages, cela signifie que chaque nouvelle arrivée se produit après un certain délai moyen, soit T_m ce délai. Alors, l'arrivée d'un message dans la file à un instant donné t permet de prévoir l'arrivée suivante au temps t + dt, avec $dt = -T_m \times Log(Va)$, où Va désigne une variable aléatoire à valeurs dans [0, 1]. Le graphe d'états est le suivant :

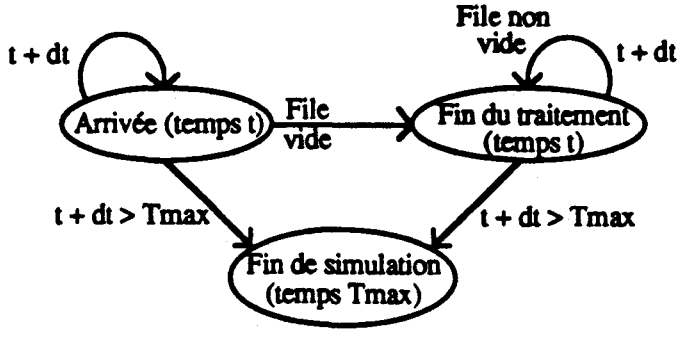


figure 6.3

Si la file est vide, alors on peut prévoir l'extraction du message qui y arrive. Sinon, pour chaque message traité, on peut prévoir (prévision statistique ou à temps fixe) l'instant de la fin du traitement du message suivant. On voit ainsi comment le traitement d'un événement appelle la création d'un autre, qui sera intercalé, sous sa forme (temps, événement), à sa place dans la suite.

6.2.2 Les événements de la simulation

Les événements significatifs retenus sont les suivants :

- l'arrivée en file d'attente mémoire d'un message en lecture ou écriture ;
- la fin du service mémoire, c'est-à-dire l'achèvement d'un cycle de lecture/écriture ;
- le début d'occupation d'un processeur pour un calcul ;
- la libération d'un processeur en fin de traitement ;
- les émissions et réceptions de messages au niveau du réseau ;
- la fin de simulation.

Comme pour la simulation présentée au chapitre 5, les programmes sont vus non pas à travers ce qu'ils "font" réellement mais en fonction du nombre et du type des messages qu'ils vont créer et faire circuler à travers le réseau.

6.2.3 Les résultats

Les résultats ont été obtenus pour trois sortes de "programmes" différents.

Programme 1 :

Il s'agit en fait de programmes indépendants lancés simultanément sur tous les noeuds du réseau. Le fonctionnement est donc équivalent à celui de processus parallèles partageant des données réparties. Deux cas de figure ont été étudiés :

a) les paquets-requêtes émis par les noeuds ne comportent de demandes que pour un opérande et les objets demandés sont implantés localement ;

b) les requêtes portent sur un ensemble d'objets pour lesquels la part d'implantation locale est plus réduite que ci-dessus.

Dans les deux cas, on a fait varier la longueur du temps de traitement pour évaluer son importance par rapport au temps de routage.

Comme on pouvait s'y attendre, on obtient des taux de parallélisme très élevés et ce d'autant plus que la localité des opérandes est importante. L'allongement du temps de

traitement accroît également le taux de parallélisme, ce qui s'explique, comme pour les résultats obtenus au chapitre précédent, par le fait que, les communications étant plus espacées dans le temps, celles-ci encombrent moins les liaisons en un instant donné.

Programme 2 :

Ici, un noeud particulier envoie des messages d'activation aux autres noeuds, qui exécutent chacun un programme, de façon indépendante comme dans le cas précédent ; puis ces noeuds renvoient des résultats au noeud "maître" qui achève son travail. Trois réseaux ont été testés :

- un réseau à sept noeuds de degré quatre ;
- un réseau à sept noeuds complètement interconnecté (degré 6) ;
- un réseau à quinze noeuds.

Dans chaque cas on a fait varier :

- la taille des paquets ;
- le temps d'exécution après retour des opérandes.

L'étude des résultats permet plusieurs constatations :

Le taux de parallélisme augmente avec le nombre de connections, la durée de routage et la densité du trafic étant moins importantes.

Comme pour le programme 1, l'augmentation de la taille des paquets accroît le parallélisme.

Par contre, le taux de parallélisme rapporté au nombre de processeurs diminue quand le nombre de processeurs augmente. Ceci confirme l'existence d'un seuil de parallélisme au delà duquel l'accroissement du nombre de processeurs ne se traduit plus par un accroissement des performances et peut même aller jusqu'à une dégradation de celles-ci.

Programme 3 :

C'est celui qui amène le fonctionnement le plus proche de celui simulé au chapitre 5. Ici, la liste des noeuds sur lesquels sont effectuées les recherches est très variable et définie dans les paramètres liés à l'instruction elle-même. Le nombre de noeuds sollicités varie d'un site à l'autre, d'une instruction à l'autre. Ceci est à rapprocher de ce qui se passe dans la simulation décrite au chapitre 5 : le traitement d'un message amène la création d'un nombre variable de messages, à destination de divers noeuds. Quant aux collisions elles sont simulées de manière aléatoire au moyen d'une fonction de Poisson.

Un facteur limitatif par rapport au chapitre 5 est que les instructions sont assez fortement dépendantes les unes des autres.

Pour ce genre de programme, le taux de parallélisme obtenu est en général assez faible et il diminue lorsque le taux de collisions augmente.

Dans les trois cas ci-dessus, les courbes obtenues en ce qui concerne l'activité des processeurs sont en tout point comparables à celles présentées dans le chapitre précédent : à une première phase de "montée en régime" succède une phase plus ou moins stable de rendement maximal puis une phase où les processeurs sont de moins en moins sollicités. Pour le programme 3, avec un réseau comportant 7 noeuds, on obtient par exemple la courbe représentée sur la figure 6.4, sur laquelle on a fait figurer en abscisses le temps d'exécution et en ordonnées le nombre de processeurs actifs.

Il en est de même pour les messages en circulation dans le réseau, pour lesquels on obtient, dans le cas du programme 2, toujours avec un réseau à sept noeuds, les courbes présentées sur le graphique de la figure 6.5., sur laquelle on a fait figurer en abscisses le temps d'exécution et en ordonnées le nombre de messages.

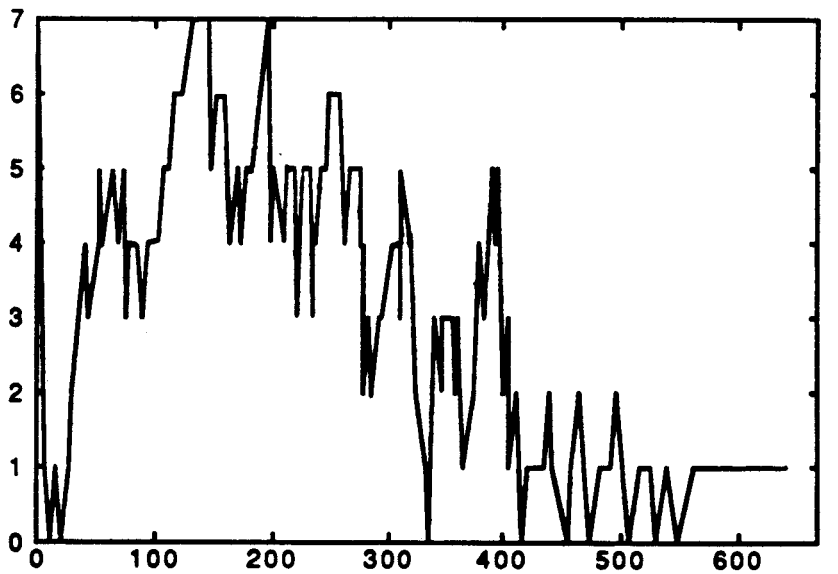


figure 6.4

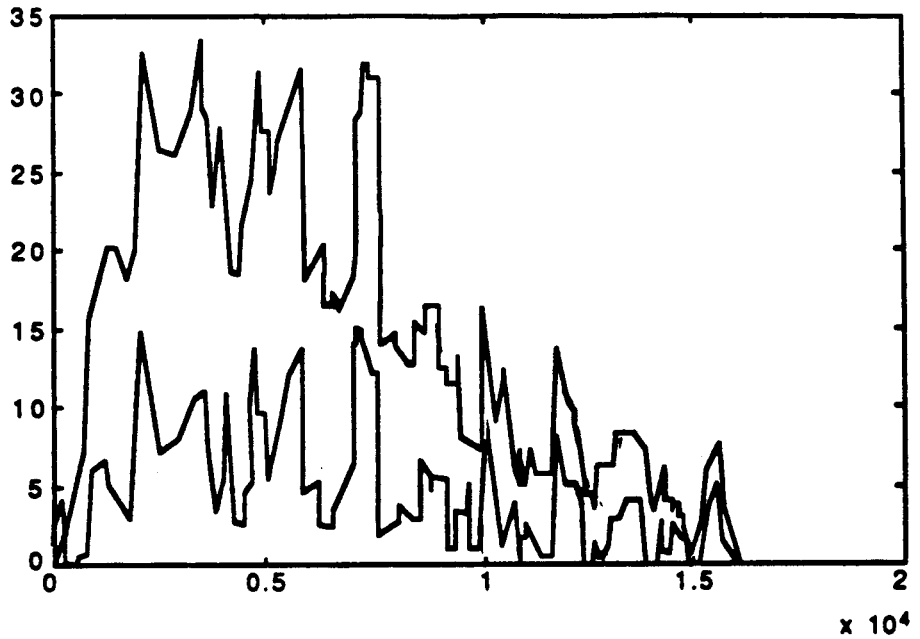


figure 6.5

Sur cette dernière figure, la courbe supérieure représente l'évolution du nombre total de messages, la courbe inférieure représentant quant à elle les messages en cours de routage après une ou plusieurs collisions.

Dans cette simulation, on a également obtenu des résultats à propos des taux de remplissages des buffers d'entrée des mémoires. Là encore, les résultats sont comparables à ceux présentés plus haut : le nombre maximum de paquets en attente est en général inférieur à 10, et souvent de l'ordre de 2 à 4.

En définitive, les résultats obtenus lors de ces deux études corroborent, et complètent, ceux présentés au chapitre 5, à la fois pour les taux de parallélisme importants que l'on peut obtenir, sur l'absence de saturation locale et sur l'importance de la fonction de hachage global.

CONCLUSION

Après avoir proposé une technique originale de répartition par hachage sur une machine multiprocesseurs sans mémoire partagée, nous avons, en exhibant 2 types de réseaux pouvant convenir, montré qu'il est possible de se plier aux contraintes imposées à la topologie du réseau par cette technique.

Les résultats obtenus en simulation confirment la validité de cette technique, au niveau résolution des collisions, par sa faculté à éviter les saturations locales et autres phénomènes de "bouchon". Comme pour toutes les techniques de hachage, la fonction de hachage proprement dite devra toutefois être judicieusement choisie.

Le projet, devenu LogArch, a maintenant pris une nouvelle voie en direction de la programmation logique plutôt que fonctionnelle. Dans ce nouveau cadre, le schéma d'évaluation choisi laisse valables les résultats présentés ici. Ceux-ci pourront maintenant être affinés, le cadre étant devenu plus strict et le mode de fonctionnement de la machine défini avec plus de précision.

TERMINOLOGIE

Le réseau est modélisé par un graphe dont les sommets représentent les noeuds du réseau et les arêtes les liaisons.

Un **graphe** est une paire G={S, A} formée d'un ensemble S de **sommets** et d'un ensemble A d'**arcs**. Ici nous utiliserons plutôt le terme **arête** qui désigne un arc non orienté, les liaisons dans le réseau étant bidirectionnelles. A peut être considéré comme un sous-ensemble de SxS.

Le degré d'un sommet désigne le nombre d'arêtes le contenant.

Le degré d'un graphe est celui du sommet qui a le plus grand degré.

Un graphe est **régulier** si tous ses sommets ont le même degré.

Un **chemin** (ou **chaîne,** si le graphe considéré n'est pas orienté) dans un graphe G={S, A} entre deux sommets s_i et s_j est une suite de sommets $s_i=s_{i_0}, s_{i_1}, s_{i_2},..., s_{i_k}=s_j$ telle que $(s_{i_{n-1}}, s_{i_n}) \in A \; \forall n \in [1, k]$.

La **longueur** d'un chemin désigne le nombre d'arêtes le constituant.

La **distance** entre deux sommets s_i et s_j est la longueur du plus court chemin joignant s_i à s_j.

Le **diamètre** d'un graphe G désigne le maximun des distances entre toutes les paires de sommets de G.

Un **chemin élémentaire** est un chemin ne passant pas deux fois par un même sommet.

Un **chemin hamiltonien** est un chemin élémentaire passant par tous les sommets du graphe.

Un graphe est **connexe** s'il existe toujours une chaîne entre deux sommets donnés.

Un **cycle** est un chemin dans lequel le sommet d'arrivée est le même que le sommet de départ.

Un **arbre** est un graphe connexe sans cycle.

On peut définir des cycles élémentaire et hamiltonien de la même façon que les chemins.

BIBLIOGRAPHIE

[ARDE81] B.W. ARDEN & H. LEE.

Analysis of Chordal Ring Network.

IEEE Transactions on Computers, Vol C30, N°4, Avr 81, pp 291-295.

[BAKO87] P. BAKOWSKI & A. PAWLAK.

LIDO : A Silicon Compiler Preprocessor.

North-Holland, Microprocessing and Microprogramming 20 (1987),

pp 167-172.

[BELL70] J.R. BELL.

The Quadratic Quotient Method : a Hash Code Eliminating Secondary Clustering.

Communications of the ACM, Février 70, Vol 13, N°2, pp 107-109.

[BHUY84] L.N. BHUYAN & D.P. AGRAWAL.

Generalized Hypercube and Hyperbus Structures for a Computer Network.

IEEE Transactions on Computers, Vol C33, N°1, Avr 84, pp 323-333.

[BOUR88] A.BOURZOUFI.

Etude des fonctions de hachage dans un réseau hypercube de type N-ARCH.

Mémoire de DEA. Laboratoire d'Informatique Fondamentale de Lille.

[BRAN86] J.E. BRANDENBURG & D.S. SCOTT.

Embeddings of Communications Trees and Grids into Hypercubes.

Document INTEL.

[CRAM85] J.A. CRAMMOND & C.D.F. MILLER.

An architecture for parallel logic languages.

Dept of Computer Science.Heriot-Watt University.79 Grasmarket. Scotland.

[DELO85] C. DELORME.

Grands Graphes de Degré et Diamètre Donnés.

European Journal of Combinatorics, Vol 6, 1985, pp 291-302.

[DESH86] S.R. DESHPANDE & R.M. JENEVEIN.

Scalability of a Binary Tree on a Hypercube.

Proceedings of the 1986 Int. Conf. on Parallel Processing, pp 661-668.

[DEVE88] N. DEVESA, G. GONCALVES, M.P. LECOUFFE & B. TOURSEL.

A Controlled Reduction Model of Functional programs on a Distributed Associative Network.

Euromicro Congress 88. Zurich (Suisse). 1988

[GOOD81] J.R. GOODMAN & C.H. SEQUIN.

Hypertree : A Multiprocessor Interconnection Topology.

IEEE Transactions on Computers, Vol C30, N°12, Dec 81,pp 923-933.

[HANN89] I. HANNEQUIN, G. GONCALVES, B. TOURSEL, P. LECOUFFE.

PROLOG : A New Parallel Evaluation Scheme.

North-Holland, Microprocessing and Microprogramming 27 (1989),

pp 391-396.

[HUGH82] J. HUGHES.

Super-Combinators : A New Implementation Method for Applicative Languages.

Proc. 1982 ACM Symposium on LISP and Functionnal Programming pp 1-10.

[IQBA86] M.IQBAL, J. SALTZ & S.BOKHARI.

A Comparative Analysis of Static and Dynamic Load Balancing Strategies.

Proceedings of the 1986 International Conference on Parallel Processing, pp 1040-1047.

[KNUT73] D.E. KNUTH.

The Art of Computer Programming. Volume 3 / Sorting and Searching.

Addison-Wesley Publishing Company.

[LI&M86] P.P. LI & A.J. MARTIN.

A Versatile Interconnection Network.

Proceedings of the 1986 Int. Conf. on Parallel Processing, pp 20-27.

[LUM&71] V.Y. LUM, P.S.T. YUEN & M. DODD.

Key-to-Address Transform Techniques : A Fundamental Performance Study on Large Existing Formatted Files.

Communications of the ACM, Avril 71, Vol 14, N°4, pp 228-239.

[MARC81] R. MARCOGLIESE & R. NOVARESE.

Module and data allocation methods in distributed systems.

Proceedings of the 2nd International Conference on Distributed Computing Systems, pp 50-59.

[MART89] J.C. MARTI.

Simulation du réseau N-ARCH.

Publication interne n° 65. LIFL. Janvier 89.

[MEMM82] G. MEMMI & Y. RAILLARD.

Some New Results About the (d,k) Graph Problem.

IEEE Transactions on Computers, Vol C31, N°8, Août 82,pp 784-791.

[MORR68] R. MORRIS.

Scatter Storage Techniques.

Communications of the ACM, Janvier 68, Vol 11, N°1, pp 38-44.

[MOUY86] A.F. MOUYART.

Communication personnelle, 1986.

[NIAR89] S. NIAR.

Contribution à l'étude des architectures d'ordinateurs parallèles : structure de la machine N-ARCH et émulation sur un réseau de Transputers.

Thèse de doctorat. LIFL. Octobre 1989.

[NICO89A] J.C. NICOLAS.

Une technique de hachage adaptée à la répartition d'une relation sur un réseau de processeurs et son utilisation dans un algorithme de jointure parallèle.

Publication interne n° 68. LIFL.

[NICO89B] J.C. NICOLAS.

Répartition d'un schéma logique d'objets complexes sur un réseau de processeurs.

Publication interne n° 73. LIFL.

[POLY86] C.D. POLYCHRONOPOULOS & U. BANERJEE.

SpeedUp Bounds & Processor Allocation for Parallel Programs on Multiprocessors.

Proceedings of the 1986 Int. Conf. on Parallel Processing, pp 961-968.

[PREP81] P. PREPARATA & J. VUILLEMIN.

The Cube-Connected Cycles : A Versatile Network for Parallel Computation.

Communications of the ACM, Mai 81, Vol 24, N°5, pp 300-309.

[RAVI87] K. RAVIKANTH, P.S. SASTRY, K.R. RAMAKRISHNAN & Y.V. VENTAKESH.

A reduction Architecture for the Optimal Scheduling of Binary Tree.

Frontiers in Computing, Dec. 87, Amsterdam.

[SEIT85] C.L. SEITZ.

The Cosmic Cube.

Communications of the ACM, Jan 85, Vol 28, N°1, pp 22-33.

[TARB70] J.C. TARBOURIECH.

Un programme de simulation de circuits logiques.

Colloque international sur la microélectronique avancée. Paris 70.

[TOUR86] B. TOURSEL.

Topologie du réseau associatif dans N-ARCH : Peut-on utiliser des Hypercubes?

Projet N-ARCH, Note n°2, Mars 86.

[TURN79] D.A. TURNER.

A New Implementation Technique for Applicative Languages.

Software Practice and Experience, Vol.9, Sept.79, pp 31-49.

TITLE:
PROPOSAL FOR A METHOD OF LOAD DISTRIBUTION ON A NETWORK OF PROCESSORS : TOPOLOGICAL CONSEQUENCIES AND SIMULATION.

ABSTRACT :
In multiprocessors architectures, one of the most important problem is load distribution and data allocation. We propose an original distribution method, consisting in a particular hashing technique. This method induces topological constraints for the interconnection network. Thus, we propose two differents topologies well suited to these constraints. Then we present simulation results showing the validity of the above distribution technique.

KEYWORDS :
Parallel architectures, multiprocessors, interconnection network, hashing, load distribution, data allocation, simulation.

TITRE :
PROPOSITION D'UNE METHODE DE REPARTITION DE LA CHARGE SUR UN RESEAU DE PROCESSEURS : CONSEQUENCES SUR LA TOPOLOGIE ET SIMULATION.

RESUME :
Dans les machines multiprocesseurs, un des problèmes les plus importants réside dans la répartition à la fois de la charge, des programmes et des données dans les processeurs. Nous proposons une méthode originale de répartition, consistant en une technique particulière de hachage. Cette méthode induit des contraintes quant à la topologie du réseau d'interconnexion de la machine. Nous proposons deux topologies différentes répondant convenablement à ces contraintes puis présentons les résultats de travaux de simulation montrant la validité de la technique de répartition préalablement définie.

MOTS CLES :
Ordinateurs parallèles, multiprocesseurs, réseau d'interconnexion, hachage, distribution de charge, répartition des données, simulation.