NUMERO D'ORDRE : 1097

ANNEE : 1993

LABORATOIRE D'INFORMATIQUE F E DE LILLE

50376
1993
97

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

Tiente HSU

Proposition d'une Architecture de Réseau d'Interconnexion à Reconfiguration Dynamique et Asynchrone

Thèse prévue pour le 19 Février 1993, devant la commission d'examen :

Président :	M. MERIAUX	Université de Lille 1
Rapporteurs :	P. BAKOWSKI	Université de Nantes
	P. FRISON	Université de Rennes 2
Directeur de Thèse	B. TOURSEL	Université de Lille 1
Examinateurs :	G. GONCALVES	Université de Lille 1
	M. MERIAUX	Université de Lille 1
	P. VINCENT	ENIC

UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE
U.F.R. d'I.E.E.A. Bât M3. 59655 Villeneuve d'Ascq CEDEX
Tél. 20.43.47.24 Fax. 20.43.65.66

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE

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. PARREAU, J. LOMBARD, M. MIGEON, J. CORTOIS, A.DUBRULLE

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

M. P. LOUIS

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CHAMLEY Hervé	Géotechnique
M. CONSTANT Eugène	Electronique
M. ESCAIG Bertrand	Physique du solide
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. LABLACHE COMBIER Alain	Chimie
M. LOMBARD Jacques	Sociologie
M. MACKE Bruno	Physique moléculaire et rayonnements atmosphériques

M. MIGEON Michel	EUDIL
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	Probabilités
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BRASSELET Jean Paul	Géométrie et topologie
M. BREZINSKI Claude	Analyse numérique
M. BRIDOUX Michel	Chimie Physique
M. BRUYELLE Pierre	Géographie
M. CARREZ Christian	Informatique
M. CELET Paul	Géologie générale
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. CROSNIER Yves	Electronique
Mme DACHARRY Monique	Géographie
M. DAUCHET Max	Informatique
M. DEBOURSE Jean Pierre	Gestion des entreprises
M. DEBRABANT Pierre	Géologie appliquée
M. DECLERCQ Roger	Sciences de gestion
M. DEGAUQUE Pierre	Electronique
M. DESCHEPPER Joseph	Sciences de gestion
Mme DESSAUX Odile	Spectroscopie de la réactivité chimique
M. DHAINAUT André	Biologie animale
Mme DHAINAUT Nicole	Biologie animale
M. DJAFARI Rouhani	Physique
M. DORMARD Serge	Sciences Economiques
M. DOUKHAN Jean Claude	Physique du solide
M. DUBRULLE Alain	Spectroscopie hertzienne
M. DUPOUY Jean Paul	Biologie
M. DYMENT Arthur	Mécanique
M. FOCT Jacques Jacques	Métallurgie
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie structurale
M. FRONTIER Serge	Ecologie numérique
M. GLORIEUX Pierre	Physique moléculaire et rayonnements atmosphériques
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie-Physique
M. GRANELLE Jean Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILBAULT Pierre	Physiologie animale
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. HENRY Jean Pierre	Génie mécanique
M. HERMAN Maurice	Physique spatiale
M. LACOSTE Louis	Biologie Végétale
M. LANGRAND Claude	Probabilités et statistiques

M. LATTEUX Michel	Informatique
M. LAVEINE Jean Pierre	Paléontologie
Mme LECLERCQ Ginette	Catalyse
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique atomique et moléculaire
M. LEROY Jean Marie	Spectrochimie
M. LHENAFF René	Géographie
M. LHOMME Jean	Chimie organique biologique
M. LOUAGE Françis	Electronique
M. LOUCHEUX Claude	Chimie-Physique
M. LUCQUIN Michel	Chimie physique
M. MAILLET Pierre	Sciences Economiques
M. MAROUF Nadir	Sociologie
M. MICHEAU Pierre	Mécanique des fluides
M. PAQUET Jacques	Géologie générale
M. PASZKOWSKI Stéfan	Mathématiques
M. PETIT Francis	Chimie organique
M. PORCHET Maurice	Biologie animale
M. POUZET Pierre	Modélisation - calcul scientifique
M. POVY Lucien	Automatique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. RAMAN Jean Pierre	Sciences de gestion
M. SALMER Georges	Electronique
M. SCHAMPS Joël	Spectroscopie moléculaire
Mme SCHWARZBACH Yvette	Géométrie
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
M. SLIWA Henri	Chimie organique
M. SOMME Jean	Géographie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. THIEBAULT François	Sciences de la Terre
M. THOMAS Jean Claude	Géométrie - Topologie
M. THUMERELLE Pierre	Démographie - Géographie humaine
M. TILLIEU Jacques	Physique théorique
M. TOULOTTE Jean Marc	Automatique
M. TREANTON Jean René	Sociologie du travail
M. TURRELL Georges	Spectrochimie infrarouge et raman
M. VANEECLOO Nicolas	Sciences Economiques
M. VAST Pierre	Chimie inorganique
M. VERBERT André	Biochimie
M. VERNET Philippe	Génétique
M. VIDAL Pierre	Automatique
M. WALLART Françis	Spectrochimie infrarouge et raman
M. WEINSTEIN Olivier	Analyse économique de la recherche et développement
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ABRAHAM Francis	
M. ALLAMANDO Etienne	Composants électroniques
M. ANDRIES Jean Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BALL Steven	Génétique
M. BART André	Biologie animale
M. BASSERY Louis	Génie des procédés et réactions chimiques
Mme BATTIAU Yvonne	Géographie
M. BAUSIERE Robert	Systèmes électroniques
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique atomique et moléculaire
M. BERNAGE Pascal	Physique atomique, moléculaire et du rayonnement
M. BERTHOUD Arnaud	Sciences Economiques
M. BERTRAND Hugues	Sciences Economiques
M. BERZIN Robert	Analyse
M. BISKUPSKI Gérard	Physique de l'état condensé et cristallographie
M. BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie végétale
M. BOHIN Jean Pierre	Biochimie métabolique et cellulaire
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie civil
M. BOIVIN Jean Claude	Spectrochimie
M. BOUCHER Daniel	Physique
M. BOUQUELET Stéphane	Biologie appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BROCARD Jacques	Chimie
Mme BROUSMICHE Claudine	Paléontologie
M. BUISINE Daniel	Mécanique
M. CAPURON Alfred	Biologie animale
M. CARRE François	Géographie humaine
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 électroniques optiques
M. COMYN Gérard	Informatique théorique
Mme CONSTANT Monique	Composants électroniques et optiques
M. COQUERY Jean Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique nucléaire et corpusculaire
M. COUTURIER Daniel	Chimie organique
M. CRAMPON Norbert	Tectolique géodynamique
M. CURGY Jean Jacques	Biologie
M. DANGOISSE Didier	Physique théorique
M. DE PARIS Jean Claude	Analyse
M. DECOSTER Didier	Composants électroniques et optiques
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean Paul	Informatique
M. DELORME Pierre	Physiologie animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
Mme DEMUYNCK Claire	Physique atomique, moléculaire et du rayonnement
M. DENEL Jacques	Informatique
M. DEPREZ Gilbert	Physique du solide - christallographie

M. LE MAROIS Henri	Vie de la firme
M. LEMOINE Yves	Biologie et physiologie végétales
M. LESCURE François	Algèbre
M. LESENNE Jacques	Systèmes électroniques
M. LOCQUENEUX Robert	Physique théorique
Mme LOPES Maria	Mathématiques
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU François	Sciences économiques
M. MAHIEU Jean Marie	Optique - Physique atomique
M. MAIZIERES Christian	Automatique
M. MANSY Jean Louis	Géologie
M. MAURISSON Patrick	Sciences Economiques
M. MERIAUX Michel	EUDIL
M. MERLIN Jean Claude	Chimie
M. MESMACQUE Gérard	Génie mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MOCHE Raymond	Modélisation,calcul scientifique,statistiques
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie organique
M. MORE Marcel	Physique de l'état condensé et cristallographie
M. MORTREUX André	Chimie organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
M. NIAY Pierre	Physique atomique,moléculaire et du rayonnement
M. NICOLE Jacques	Spectrochimie
M. NOTELET Francis	Systèmes électroniques
M. PALAVIT Gérard	Génie chimique
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. PERTUZON Emile	Physiologie animale
M. PETIT Daniel	Biologie des populations et écosystèmes
M. PLIHON Dominique	Sciences Economiques
M. PONSOLLE Louis	Chimie physique
M. POSTAIRE Jack	Informatique industrielle
M. RAMBOUR Serge	Biologie
M. RENARD Jean Pierre	Géographie humaine
M. RENARD Philippe	Sciences de gestion
M. RICHARD Alain	Biologie animale
M. RIETSCH François	Physique des polymères
M. ROBINET Jean Claude	EUDIL
M. ROGALSKI Marc	Analyse
M. ROLLAND Paul	Composants électroniques et optiques
M. ROLLET Philippe	Sciences Economiques
Mme ROUSSEL Isabelle	Géographie physique
M. ROUSSIGNOL Michel	Modélisation,calcul scientifique,statistiques
M. ROY Jean Claude	Psychophysiologie
M. SALERNO Françis	Sciences de gestion
M. SANCHOLLE Michel	Biologie et physiologie végétales
Mme SANDIG Anna Margarette	
M. SAWERYSYN Jean Pierre	Chimie physique
M. STAROSWIECKI Marcel	Informatique
M. STEEN Jean Pierre	Informatique
Mme STELLMACHER Irène	Astronomie - Météorologie
M. STERBOUL François	Informatique
M. TAILLIEZ Roger	Génie alimentaire
M. TANRE Daniel	Géométrie - Topologie
M. THERY Pierre	Systèmes électroniques
Mme TJOTTA Jacqueline	Mathématiques
M. TOURSEL Bernard	Informatique
M. TREANTON Jean René	Sociologie du travail

M. DERIEUX Jean Claude	Microbiologie
M. DERYCKE Alain	Informatique
M. DESCAMPS Marc	Physique de l'état condensé et cristallographie
M. DEVRAINNE Pierre	Chimie minérale
M. DEWAILLY Jean Michel	Géographie humaine
M. DHAMELINCOURT Paul	Chimie physique
M. DI PERSIO Jean	Physique de l'état condensé et cristallographie
M. DUBAR Claude	Sociologie démographique
M. DUBOIS Henri	Spectroscopie hertzienne
M. DUBOIS Jean Jacques	Géographie
M. DUBUS Jean Paul	Spectrométrie des solides
M. DUPONT Christophe	Vie de la firme
M. DUTHOIT Bruno	Génie civil
Mme DUVAL Anne	Algèbre
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FARVACQUE Jean Louis	Physique de l'état condensé et cristallographie
M. FAUQUEMBERGUE Renaud	Composants électroniques
M. FELIX Yves	Mathématiques
M. FERRIERE Jacky	Tectonique - Géodynamique
M. FISCHER Jean Claude	Chimie organique, minérale et analytique
M. FONTAINE Hubert	Dynamique des cristaux
M. FORSE Michel	Sociologie
M. GADREY Jean	Sciences économiques
M. GAMBLIN André	Géographie urbaine, industrielle et démographie
M. GOBLOT Rémi	Algèbre
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. GUELTON Michel	Chimie physique
M. GUICHAOUA André	Sociologie
M. HAIMAN Georges	Modélisation,calcul scientifique, statistiques
M. HOUDART René	Physique atomique
M. HUEBSCHMANN Johannes	Mathématiques
M. HUTTNER Marc	Algèbre
M. ISAERT Noël	Physique de l'état condensé et cristallographie
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et statistiques
M. JEAN Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KOENIG Gérard	Sciences de gestion
M. KOSTRUBIEC Benjamin	Géographie
M. KREMBEL Jean	Biochimie
Mme KRIFA Hadjila	Sciences Economiques
M. LANGEVIN Michel	Algèbre
M. LASSALLE Bernard	Embryologie et biologie de la différenciation
M. LE MEHAUTE Alain	Modélisation,calcul scientifique,statistiques
M. LEBFEVRE Yannic	Physique atomique,moléculaire et du rayonnement
M. LECLERCQ Lucien	Chimie physique
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Marc	Composants électroniques et optiques
M. LEFEVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
M. LEGRAND Michel	Astronomie - Météorologie
M. LEGRAND Pierre	Chimie
Mme LEGRAND Solange	Algèbre
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne

M. TURREL Georges	Spectrochimie infrarouge et raman
M. VANDIJK Hendrik	
Mme VAN ISEGHEM Jeanine	Modélisation,calcul scientifique,statistiques
M. VANDORPE Bernard	Chimie minérale
M. VASSEUR Christian	Automatique
M. VASSEUR Jacques	Biologie
Mme VIANO Marie Claude	
M. WACRENIER Jean Marie	Electronique
M. WARTEL Michel	Chimie inorganique
M. WATERLOT Michel	géologie générale
M. WEICHERT Dieter	Génie mécanique
M. WERNER Georges	Informatique théorique
M. WIGNACOURT Jean Pierre	
M. WOZNIAK Michel	Spectrochimie
Mme ZINN JUSTIN Nicole	Algèbre

Table des matières

Introduction

2 février 1993

De nos jours, les domaines d'application des ordinateurs ne cessent de se développer, notamment dans le domaine de l'intelligence artificielle, du traitement d'images, ou encore dans le domaine scientifique : analyse des milieux continus (analyse de structure, dynamique des fluides...), et traitement du signal (sismique, transformé de Fourier...). Ces développements s'accompagnent d'une augmentation importante du degré de complexité des logiciels utilisés et de leur taille. Et quel que soit l'accroissement des performances obtenues dans un système monoprocesseur (Augmentation de la fréquence, Architecture RISC), le recours au parallélisme reste la seule voie permettant d'obtenir la puissance nécessaire pour répondre aux nouveaux besoins.

L'évolution de la technologie dans le domaine de la densité d'intégration et les nouvelles architectures de processeurs de type RISC sont à la source d'un regain d'intérêt pour les architectures parallèles : les processeurs sont bon marché, facilement conditionnables et consomment peu d'énergie. Il est économiquement faisable de concevoir une architecture contenant un grand nombre de processeurs pour atteindre une performance pic dépassant les supercalculateurs d'aujourd'hui. L'approche supercalculateur, comme par exemple de type Cray, consiste à développer des architectures contenant quelques dizaines de processeurs très puissants. La mise en oeuvre de ce type d'architecture, qui nécessite l'utilisation de techniques très avancées (fréquence d'horloge rapide, nouveaux matériaux, processeurs dédiés, etc), est très coûteuse pour une augmentation de performance de l'ordre de 5 à 10 d'une version à l'autre. Après l'apparition du processeur Alpha, de fabrication série et très puissant, Cray se tourne également vers la conception des systèmes dotés d'un grand nombre de processeurs. Des architectures interconnectant un très grand nombre de processeurs, de plus en plus puissants, ont été réalisées, ou sont en cours de réalisation. On distingue 2 types d'architectures parallèles :

- Architectures à mémoire commune, **les multiprocesseurs** [LNBA89] :

tous les processeurs peuvent accéder à la totalité de la mémoire.

- Architectures à mémoire distribuée, **les multicalculateurs** [AS88] : chaque processeur dispose localement d'une mémoire privée.

Les architectures à mémoire commune

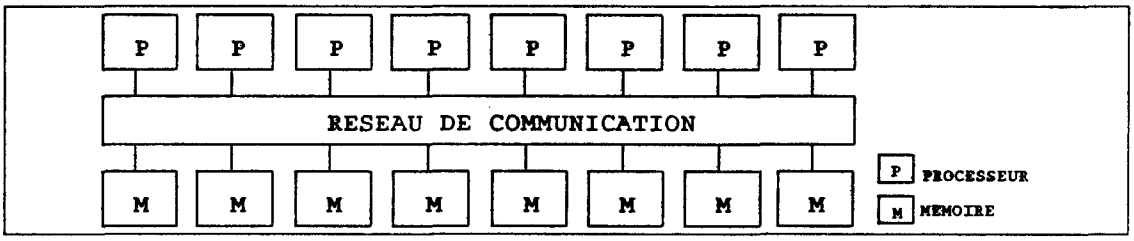

Figure 0.1 : Architecture à mémoire commune

Dans ce type d'architecture [Lit90], la mémoire joue un double rôle : d'une part le rôle classique de la mémoire contenant instructions et données propres au programme en cours d'exécution sur chaque processeur, et, d'autre part, un rôle de communication entre les processeurs par le biais des variables partagées. Cette mémoire commune est donc très sollicitée (quasiment à chaque instruction) et son accès forme un goulot d'étranglement bien connu. En pratique, les processeurs sont dotés d'une mémoire cache. L'utilisation du cache réduit la fréquence d'accès mémoire par les processeurs, mais pose des problèmes de cohérence des données source de trafic avec la mémoire, et le maintien de la cohérence augmente la complexité du matériel et du logiciel.

Par ailleurs, des réseaux à commutation de circuits entre les processeurs et les mémoires sont systématiquement employés. Ils assurent à la fois un débit important et un temps de latence minimal. Mais ces réseaux deviennent vite inefficaces quand le nombre de processeurs atteint quelques centaines d'unités, que ce soit pour le Crossbar (augmentation quadratique), ou pour des réseaux multiétages (complexité de la commande [NS81], [Len78], [And77]).

Les langages utilisés dans ce type d'architecture sont dits *orienté procédure* (PASCAL Modula) [AS83], l'interaction des processus est basée sur des variables partagées. Ces langages fournissent des mécanismes élémentaires d'exclusion mutuelle pour résoudre le problème de l'accès concurrent à des variables partagées. Le point critique réside dans l'accès à la mémoire commune, qui contient ces variables partagées par les processeurs; cela se traduit par des conflits d'accès entre

les processeurs et par de fortes contraintes sur le réseau d'interconnexion processeurs mémoire : toutes les informations doivent transiter par ce réseau, ce qui limite sévèrement la taille de ces architectures.

Les architectures à mémoire distribuée

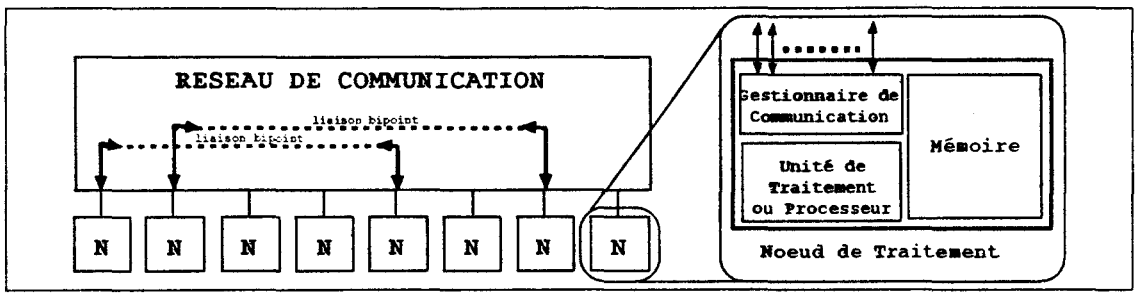

Figure 0.2 : Architecture à mémoire distribuée

Un multicalculateur est formé par un ensemble de noeuds autonomes interconnectés par des liaisons bipoints ; chaque noeud est composé d'un processeur, d'une mémoire locale privée et d'un circuit de gestion de communication inter-noeuds.

La mémoire contient les informations qui sont propres au processeur et les copies d'informations partagées. Chaque noeud exécute des tâches indépendamment de tous les autres noeuds. La communication et la synchronisation entre les noeuds reposent sur des primitives de passage de messages.

La programmation de telles architectures est difficile et nécessite que le programmeur gère explicitement la communication et la synchronisation dans ses programmes. Les langages utilisés sont du type *orienté message*, qui contrairement aux langages orientés procédures, n'ont pas de variables partagées. Le programmeur construit son programme comme un ensemble de processus s'exécutant en parallèle sur les différents processeurs. Il ne dispose que de 2 primitives **send** et **receive** pour définir et gérer les communications et la synchronisation interprocessus. Occam [Inm88] issu du modèle CSP (Communicating Sequential Processes) [Hoa78], Gypsy [GCKW79] et PLITS [Fel79] sont des exemples de langage orienté message.

Le mécanisme de communication par échanges de messages, généralement moins efficace que le partage de mémoire, se traduit par un grain de parallélisme moins fin pour les architectures à mémoire distribuée; le **grain** de parallélisme d'une application est défini ici comme étant le rapport entre la charge en calcul et la charge en communication.

Contrairement aux architectures à mémoire commune, les échanges d'informations les plus fréquents, acquisition de l'instruction et manipulation de données locales, s'effectuent à courte distance puisque mémoire et processeur résident sur le même noeud et n'utilisent pas le réseau pour communiquer entre eux. Le délai introduit par la transmission à travers le réseau ne pénalisera que les échanges relativement moins fréquents entre processus parallèles.

Cet allégement des contraintes sur le réseau permet d'envisager l'interconnexion d'un très grand nombre de processeurs et, de ce fait, permet d'apporter un accroissement des performances beaucoup plus significatif. Néanmoins, des difficultés persistent pour l'obtention des performances comme, par exemple, sur la transmission des données. Les performances de ces architectures sont en effet fortement limitées par leur capacité à communiquer rapidement.

Une autre difficulté réside dans l'adaptation de la structure de calcul d'une application à la structure de la machine. Dans une architecture à mémoire distribuée, l'ensemble des processeurs est interconnecté suivant une topologie donnée (grille, arbre, hypercube) [Fen81]. Chaque application peut être modélisée par *un graphe de processus* où, à chaque noeud, on fait correspondre un processus et les arcs entre les noeuds modélisent les canaux de communication. Le problème de l'adaptation du graphe des processus à la topologie de la machine cible est appelé *le problème de placement des tâches*. Ce problème est NP-complet, c'est à dire qu'il n'existe pas d'algorithme qui détermine la solution optimale en un temps polynômial, d'autant plus que le graphe de processus n'est pas de nature statique, puisque les processus sont créés ou tués de manière dynamique au cours de l'exécution.

L'augmentation des performances peut donc se faire en utilisant une topologie de réseau adaptée et/ou en optimisant les algorithmes de communication et de routage.

Le plan de l'exposé

Le but de cette thèse est de proposer un réseau permettant d'établir des liaisons physiques directes, pour supporter les communications entre les processeurs à la demande et de manière totalement asynchrone.

L'organisation de l'exposé est le suivant :

- Le chapitre 1 est composé de 2 parties. La première s'intéresse aux topologies des réseaux de communication utilisées dans les architectures à mémoires distribuée. Les différents modes de communication, et algorithmes

de routage, ainsi que leur coût en communication sont présentés. La deuxième partie expose les différents algorithmes de placement des processus dans ces architectures. Elle décrit les limites de ces algorithmes dans les topologie statique, et donne les intérêts de l'utilisation des topologies dynamiques pour le placement des processus.

- Le chapitre 2 présente les différentes machines reconfigurables existantes. L'attention est portée sur les réseaux reconfigurables, dont nous présentons les plus connus, parmi lesquels figure le réseau que nous utilisons. Nous exposons les raisons de ce choix et présentons une étude sur le dimensionnement idéal du réseau en fonction des paramètres du système comme le nombre de processeurs ou la taille ou la fréquence des messages.

- Le chapitre 3 présente le projet Architecture à Réseau Partagé (ARP), qui propose une architecture de réseau d'interconnexion à reconfiguration dynamique et asynchrone. L'approche modulaire adoptée consiste à réaliser le réseau par assemblage de modules de communication de base identiques. Les différentes unités du modules sont décrites, nous insisterons en particulier sur la partie contrôle qui constitue l'élément clef de l'architecture.

- Le chapitre 4 concerne la modélisation ARP. L'outil utilisé est un logiciel d'aide à la modélisation par files d'attente, le logiciel Qnap2. Après une description de Qnap, nous présentons la modélisation d'ARP à l'aide d'un ensemble de symboles que nous avons définis. Enfin, nous présentons les résultats des simulations d'ARP, qui donnent le comportement du système en mode de fonctionnement équilibré et saturé.

- Le chapitre 5 concerne la réalisation d'un module de communication de base sous solo1400. Une description détaillée du module de base est présentée, notamment l'automate de contrôle de l'unité de décision. La réalisation physique permet de à la fois de démontrer la faisabilité du projet et de déterminer les performances du circuit . Les résultats obtenus servent de données d'entrée à la simulation Qnap2.

- Chapitre 6 présente les directions futures du projet ARP. nous présentons les optimisations possibles, ainsi que des solutions architecturales, qui utilisent le module défini et réalisé dans le cadre de ce travail, comme brique de base pour réaliser des systèmes disposant de plus de connexions, et/ou ayant un débit du réseau plus important, et/ou permettant d'interconnecter un grand nombre de processeurs.

Chapitre 1

Communication et Placement dans les architectures parallèles

2 février 1993

1.1 La communication dans les architectures à mémoire distribuée

On distingue principalement 2 types de réseaux de communication : les réseaux à **topologie statique** et ceux à **topologie dynamique**. Dans un réseau à topologie statique, les liens entre les processeurs sont fixes et ne peuvent être modifiés pour une connexion directe vers d'autres processeurs. Dans un réseau à topologie dynamique, les processeurs sont connectés à un ensemble de commutateurs, ces derniers étant organisés suivant un graphe donné (par exemple une grille). L'établissement des communications entre les processeurs se fait pendant l'exécution, par le positionnement de ces commutateurs.

1.1.1 Les topologies statiques

Un réseau à topologie statique [Fen81] se caractérise par un graphe non complètement connecté dont les sommets sont les processeurs et les arêtes les liens de communication. Concernant ce graphe, on définit plusieurs propriétés [Dal90] :

- **Le degré** d'un sommet est le nombre de ses voisins, ou encore le nombre de ses liens incidents. Un graphe est dit régulier, si tous les sommets ont le même degré, qui est appelé le degré du graphe.
- **La distance** entre 2 sommets est la longueur du plus court chemin, évalué en nombre de liens traversés, qui les relie.

- **le diamètre** du graphe est le maximum des distances entre 2 sommets du graphe.
- **La largeur de bissection** est le nombre de liens, qui une fois coupés, sépare le réseau en 2 parties égales.

La topologie idéale doit avoir un faible diamètre, afin de minimiser le temps de communication entre 2 sommets quelconques, et un faible degré, pour réduire le nombre de connexions nécessaires, par conséquent le coût matériel, et doit permettre de relier un très grand nombre de sommets pour obtenir un gain de performance significatif. Bien évidemment, ces 3 objectifs ne sont pas indépendants; en particulier si le degré (Δ) et le diamètre (D) sont fixés, il existe une limite théorique au nombre de sommets du graphe, connue comme la borne de Moore et égale à

$$\boxed{1 + \Delta + \Delta(\Delta - 1) + \Delta(\Delta - 1)^2 + \ldots + \Delta(\Delta - 1)^{D-1}} \tag{1.1}$$

Le (Δ, D) problème a été largement exploré dans la littérature [Bon87]. Cependant, tous les graphes proposés comme solution ne sont pas d'un égal intérêt comme modèle de réseau d'interconnexions ; soit qu'ils n'offrent pas d'algorithme d'acheminement de messages (routage) commode, soit que leurs propriétés de tolérance aux fautes soient mal connues, soit, et peut être surtout, qu'ils n'aient pas de représentation géométrique simple autorisant leur réalisation dans un espace à deux ou trois dimensions.

Parmi les graphes les plus utilisés, on peut citer :

1.1.1.1 l'hypercube

C'est le graphe le plus connu. Un hypercube (figure 1.1) de dimension n peut être défini en considérant ses sommets comme étiquetés par les représentations binaires des nombres de 0 à 2^{n-1}; deux sommets sont connectés si leurs étiquettes diffèrent exactement d'un bit. Son degré et son diamètre sont égaux à n. Il peut être construit récursivement à partir d'hypercubes de dimension inférieure : si on considère deux hypercubes de dimensions n-1, dont les sommets sont étiquetés de 0 à 2^{n-1}, en joignant les sommets de même étiquette, on obtient un hypercube de dimension n. Du point de vue de la modularité, l'augmentation de la taille du réseau se fait toujours en doublant le nombre de sommets existants, et cette modularité est limitée par le nombre maximal de connexions initialement prévues sur chaque sommet. L'avantage d'un hypercube réside certainement dans son

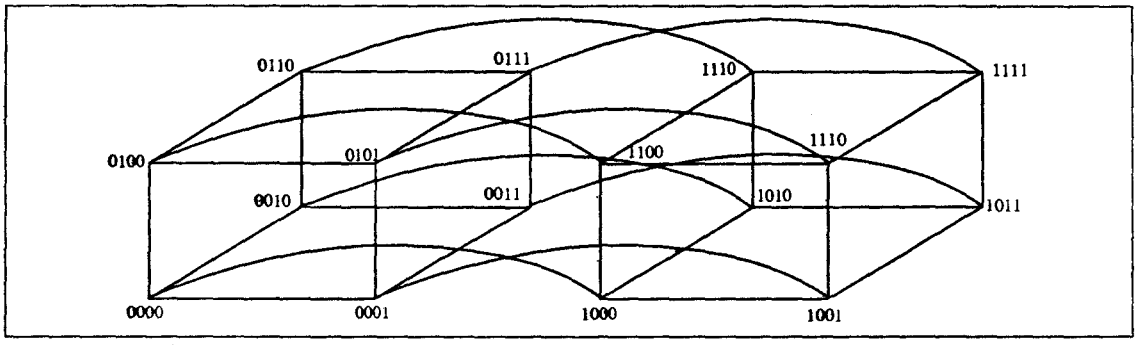

Figure 1.1 : hypercube 4D

algorithme de routage particulièrement simple, sans blocage et adapté à une réalisation cablée. Chaque sommet de l'hypercube est identifié par n bits, et chacun de ces bits représente une dimension du cube. Le routage consiste à changer les dimensions du cube dans un ordre défini à l'avance ; en général, on utilise l'ordre naturel croissant des dimensions, mais toutes les permutations sont valides.

1.1.1.2 La Grille

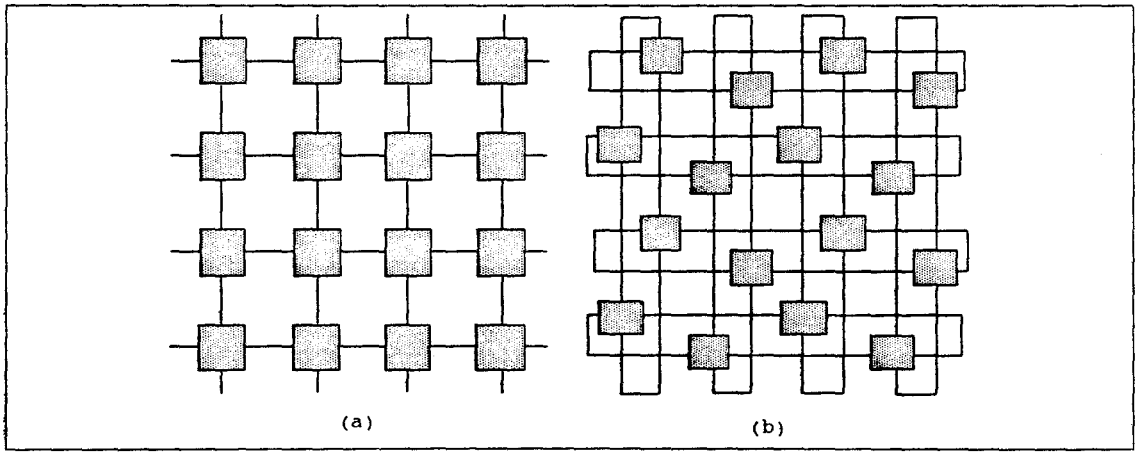

Figure 1.2 : Réseau en grille 2D (a) et en tore 2D (b)

La grille de dimension n et de côté k possédent k^n sommets que l'on peut voir comme les points de coordonnées entières comprises entre 0 et k-1 dans un espace euclidien de dimension n. Chaque sommet est connecté à ceux dont une coordonnée diffère exactement de 1 des siennes dans chacune des dimensions. La grille n'est évidemment pas un graphe régulier : le degré des sommets internes est 2n,

alors que, par exemple, le sommet (0,0,...,0) n'a que n sommets adjacents. Ce n'est pas le cas des grilles cycliques ou tores, où les sommets situés à l'extrémité sont interconnectés. Le tore est un graphe régulier, mais il n'est pas infiniment extensible à cause de la longueur des liaisons entre les sommets extrêmes. Un des intérêts essentiels de la grille est qu'elle est infiniment extensible suivant k et extensible suivant n jusqu'à la limite des interconnexions disponibles ; en ce sens, c'est la plus modulaire des topologies. Elle possède un algorithme de routage optimal analogue au précédent. Malheureusement son diamètre augmente en fonction des paramètres k et n, et est égal à $n(k-1)$.

1.1.1.3 L'arbre n-aire

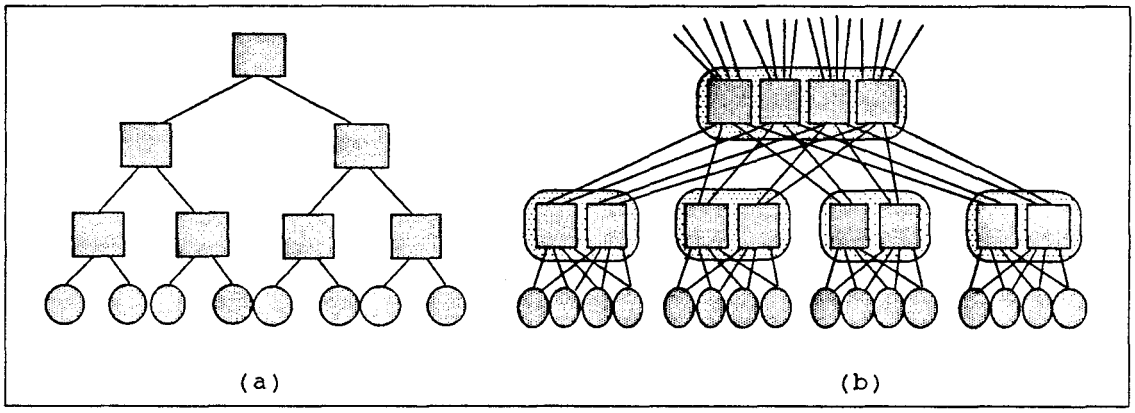

Figure 1.3 : L'arbre binaire complet (a) et le Fat-Tree de la CM5 (b)

Cette topologie suscite un intérêt nouveau au sein des grands constructeurs. En effet, les modèles de programmation à parallélisme de données et SPMD (Single Program and Multiple Data) exigent que les systèmes multicalculateurs soient capables d'effectuer efficacement des mécanismes de communications globales, comme la diffusion ou réduction, et la structure même de l'arbre semble bien convenir à ce type de communication. Mais, l'un des inconvénients majeurs est le débit que doit supporter la racine dans le cas des communications intensives entre sous-arbre droit et sous-arbre gauche.

1.1.1.4 Conclusion

Nombreux sont les multicalculateurs munis de ce type de réseau d'interconnexion à topologie statique, celle-ci étant choisie définitivement lors de la conception de la machine. Les réseaux d'interconnexion fréquemment utilisés sont l'hypercube

(utilisé par connexion machine [Hil85] CM1 et CM2 et NCUBE), la grille (MEGA [GR89], MOSAIC [AS88], PARAGON [Cor91a]) et l'arbre n-aire pour la CM5 [Cor91b] [LAD+92], qui utilise 3 arbres différents pour supporter les différents types de données de la machine : fat-tree [Lei85] pour le réseau de données, un arbre binaire complet pour le réseau de contrôle et un arbre binaire pour le réseau de diagnostique. Dans ce type de topologie, où les connexions sont fixes, les fonctions d'acheminement des messages appelées *routage* sont donc primordiales. Avant de présenter les topologies dynamiques, nous allons d'abord décrire les différentes techniques de communication dans ce type de topologie.

1.1.2 Le routage

Nous étudions ici les différents moyens de communication entre les noeuds d'un réseau à topologie statique. Les processeurs communiquent en échangeant des messages le long des liens de communication. Quand un message est expédié par un noeud source vers un noeud destinataire, qui n'est pas directement lié au noeud source, le message doit transiter à travers des noeuds intermédiaires. Chaque noeud doit être capable de réexpédier les messages qui lui parviennent, mais qui ne lui sont pas destinés, sur un chemin approprié à la destination finale. L'ensemble des dispositifs permettant de réaliser cette fonction constitue **la stratégie de routage** du réseau. Celle-ci résulte de plusieurs choix, qui sont loin d'être totalement indépendants.

- l'algorithme de routage,
- la technique de commutation,
- la gestion des conflits.

1.1.2.1 L'algorithme de routage

C'est l'algorithme qui détermine le chemin logique emprunté par le message, au sens de chemin dans le graphe sous-jacent. Les routages étudiés dans la littérature sont caractérisés par les propriétés suivantes :

1. **Routage distribué/centralisé** : la décision du routage est locale au noeud intermédiaire traversé par un message. Ceci est à opposer au routage centralisé dans lequel un noeud maître gère toutes les routes, solution coûteuse dans le contexte du massivement parallèle.

2. **Routage adaptatif** : la fonction de routage permet l'utilisation d'un ensemble de chemins entre la source et la destination, les chemins étant choisis en fonction du trafic. On cherche alors à répartir au mieux la charge sur les canaux. Ces routages permettent également de résister aux pannes [Dua91].

3. **Routage déterministe** : le chemin emprunté par un message ne dépend que de sa destination, sans tenir compte du trafic. Ce type de routage est implémenté sur le Torus Routing Chip [DS86].

4. **Routage aléatoire** : il consiste à émettre le message vers une destination tirée au hasard, puis à effectuer un routage normal vers le destinataire. Cette solution vise à diminuer les risques de points de saturation en effectuant une distribution uniforme des communications. Elle diminue les risques de conflits et supprime malheureusement aussi les propriétés de localité du réseau. Cette solution sera offerte par le C104 d'Inmos [Inm91], le crossbar qui permettra d'interconnecter des T9000.

1.1.2.2 La technique de commutation

Elle définit le mode de réalisation physique du chemin logique. On trouve une présentation des diverses techniques de commutation dans l'introduction de [KK79] :

1. Le mode **commutation de messages** ou **store and forward** (figure 1.4.b). Dans ce mode de transmission, le noeud intermédiaire, ou son gestionnaire de communication, reçoit le message et le stocke (éventuellement en mémoire), puis le renvoie au noeud suivant. Le message est donc reçu puis redirigé à chaque étape de communication. Ce mode de communication se rencontre sur toutes les machines à base de transputers (comme la série T de FPS), sur le NCUBE et sur iPSC 1 ou 2. Il peut être effectué directement par un routeur (iPSC2) ou par programmation (FPS).

2. Les modes par circuits : un noeud intermédiaire n'a pas à stocker le message ; une connexion directe s'établit au niveau du gestionnaire de communication, et le message est directement redirigé vers le noeud suivant. Ce mode de communication est celui élaboré sur les dernières versions de la série iPSC d'Intel : iPSC860. On distingue plus finement trois manières de réaliser pratiquement les communications dans ce mode.

   (a) **Commutation de circuits** : (figure 1.4.a) L'émetteur envoie l'entête du message constitué de l'adresse du destinataire (ou de toutes les données permettant de joindre celui-ci). L'entête est routé de la source

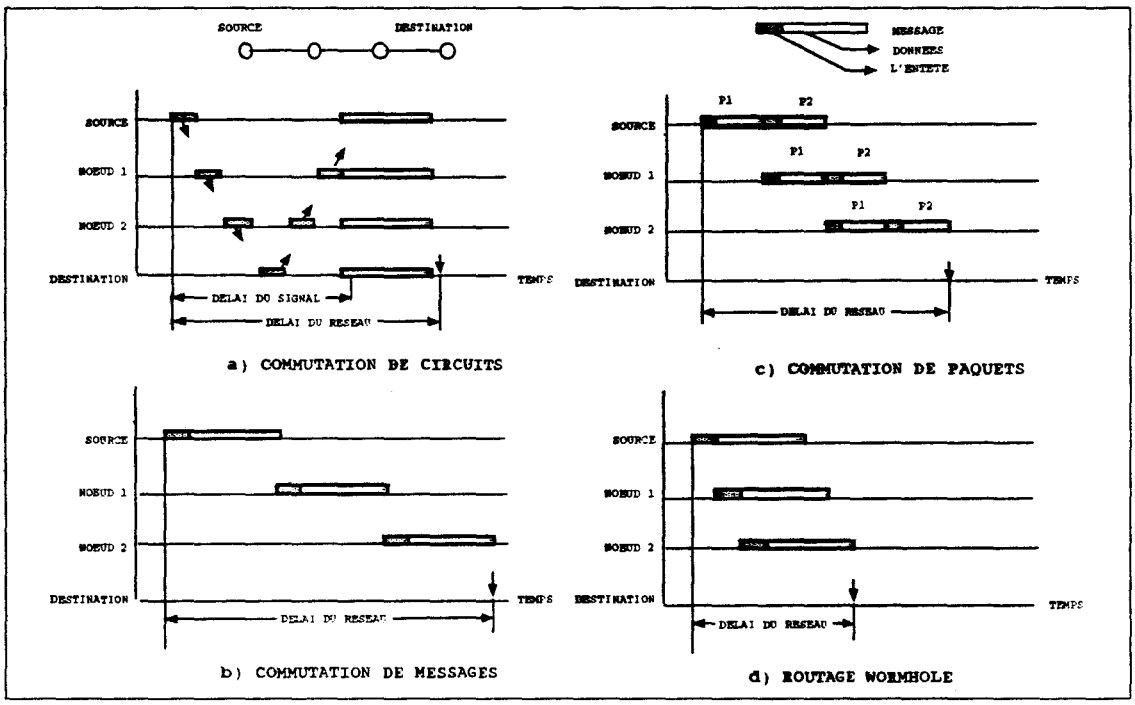

Figure 1.4 : Différentes techniques de commutation

vers le destinataire en établissant les connexions sur tous les noeuds intermédiaires. Le destinataire envoie alors un accusé de réception, et enfin, le corps du message suit le circuit créé de la source jusqu'au destinataire. D'autres communications qui voudraient emprunter une partie d'un circuit déjà établi sont bloquées jusqu'à la libération de la ligne. Le **direct connect** qui est proposé par Intel sur les hypercubes de la série iPSC/2 et iPSC i860 est un routeur de ce type.

(b) **Routage Wormhole :** (figure 1.4.d) le schéma est identique à la *commutation de circuits* mais le corps du message suit l'entête dès le début de l'établissement du circuit. En cas de blocage (si arrivé à un noeud intermédiaire, tous les autres liens de communication sont utilisés), le message est alors stocké sur tout le long du circuit en attendant de pouvoir continuer sa progression vers sa destination. Le wormhole a été choisi par Inmos pour la nouvelle série des Transputers T9000 et des crossbars C104 [Inm91]. Le routeur du C104 réalise également un multiplexage du lien physique de communication, les canaux virtuels ainsi créés permettent l'entrelacement des messages.

(c) **Virtual cut through routing [KK79] :** il s'agit d'un mode de communication destiné à améliorer le routage *Wormhole*. En cas de

blocage, le message n'est pas stocké tout au long du circuit, mais sur le dernier noeud. Ceci permet de libérer plus rapidement les canaux de routage, mais suppose des registres de tailles infinies au niveau du gestionnaire de communication, ce qui ne permet pas de les intégrer facilement avec le noeud. Cette technique n'a pas été implantée à notre connaissance.

Il ne faut pas confondre le routage Wormhole avec le routage à commutation de paquets (figure 1.4.c) qui est une optimisation du routage à commutation de messages. Les paquets résultent d'un découpage du message; envoyés en pipeline le long du chemin entre l'expéditeur et le destinataire, ils permettent, d'une part, de minimiser le temps de latence et, d'autre part, de diminuer la taille de la mémoire nécessaire à la mémorisation du message. Un paquet contient en plus des données, des informations de routage (entête), alors que dans le cas du routage Wormhole l'entité qui transite entre les noeuds, une fois le chemin établi, a une taille correspondant à la largeur du canal de transmission, cette entité est appelée **flit** [Dal90] et ne contient que des données à transmettre.

1.1.2.3 La gestion des conflits

Puisque le réseau n'est pas complètement connecté, chaque lien physique sera en général partagé par plusieurs chemins. Les conflits pour l'attribution des liens doivent donc être arbitrés. Ceux-ci peuvent être effectués au niveau des liens individuels, ou au niveau du chemin global. Lors d'un conflit d'accès à un canal de communication par 2 messages, un des messages recevra effectivement le canal, et le sort réservé au deuxième message dépendra de l'algorithme de résolution de conflits.

1. **Par mémorisation** : le deuxième message est mémorisé au niveau du noeud en attendant que le canal se libère. Cette méthode exploite au mieux les ressources de communication, mais elle nécessite une capacité de mémorisation importante au niveau de chaque noeud. Cette capacité dépasse souvent celle du noeud, le message supplémentaire est alors alloué dans la mémoire du processeur, ce qui augmente le temps de latence.

2. **Par blocage** : voir routage Wormhole.

3. **Par routage forcé** : cette technique décrite dans [GR89] est utilisée dans la machine Mega. Le but est de ne jamais bloquer un message, et dans le cas où le message arrivé à destination ne peut être consommé tout de suite, il est

re-routé et sera accepté ultérieurement. Le problème est d'assurer l'absence de famine, c'est à dire d'être sûr que tout message rerouté arrivera bien à destination. Des techniques similaires sont mises en oeuvre sur la CM-2.

4. **Par canaux virtuels** : l'interblocage est le résultat d'une dépendance cyclique dans la gestion des communications entre processus, les noeuds se bloquant mutuellement. Dans [DS87b], les auteurs démontrent que, s'il existe un graphe acyclique qui parcourt tous les éléments du réseau, alors, l'interblocage ne peut pas se produire. Les canaux virtuels sont utilisés pour construire un graphe acyclique. Le routage est déterministe puisque le message doit toujours suivre le même chemin pour atteindre sa destination; par conséquent, ce type de routage ne tolérera que très peu les pannes. Cette solution nécessite, d'une part, une mémorisation des messages en attente et, d'autre part, une unité chargée d'effectuer le partage du canal physique par plusieurs canaux logiques, la commutation des canaux logiques introduit forcément des délais d'attente.

1.1.3 Les topologies dynamiques

Il existe un deuxième type de réseau de communication dans lequel le support physique est fixé, mais où des commutateurs ou connecteurs permettent de modifier le schéma de connexion du réseau. Ce sont les réseaux à topologie dynamique ou reconfigurable. Ces réseaux ont été très étudiés dans le cadre des architectures à mémoire commune (STARAN [Bat76], TRAC [PKM+82], BBN Butterfly [RT86]). Un certain nombre de définitions ont été introduites permettant de les caractériser.

1. Un réseau pour lequel il est possible d'établir une connexion de toute entrée à toute sortie est dit **réseau de Banyan**.

2. Un réseau est dit **sans blocage** si on peut toujours établir une liaison entre toute entrée libre et toute sortie libre, et cela sans modifier les liaisons déjà établies : c'est le cas du réseau de crossbar où toute entrée est directement connectée à toute sortie.

3. Un réseau est dit **réarrangeable** s'il a le même nombre d'entrées que de sorties et si on peut toujours établir une liaison entre toute entrée libre et toute sortie libre, quitte à modifier des liaisons déjà établies : c'est le cas des réseaux de Clos [Clo53] et de Benès [Ben62].

Parmi les réseaux les plus utilisés on trouve :

1.1.3.1 Le crossbar

Le crossbar m vers r obtenu par juxtaposition de r multiplexeurs indépendants m vers un est le plus simple et le plus performant des réseaux de commutation. Chaque connexion ne traverse qu'un seul multiplexeur et ne pose pas de problème de choix du chemin à utiliser. Le coût de réalisation d'un crossbar (nombre de portes) est malheureusement proportionnel au carré du nombre de liens. La taille limite d'un crossbar monolithique est de l'ordre de la centaine, certains constructeurs réalisent des crossbars de taille supérieure par assemblage des éléments plus petits (RPP 128 × 128 par assemblage de 256 crossbars 8 × 8, VP500 224 × 224). L'autre façon de repousser cette limite physique est de construire des réseaux multi-étages.

1.1.3.2 Les réseaux multi-étages

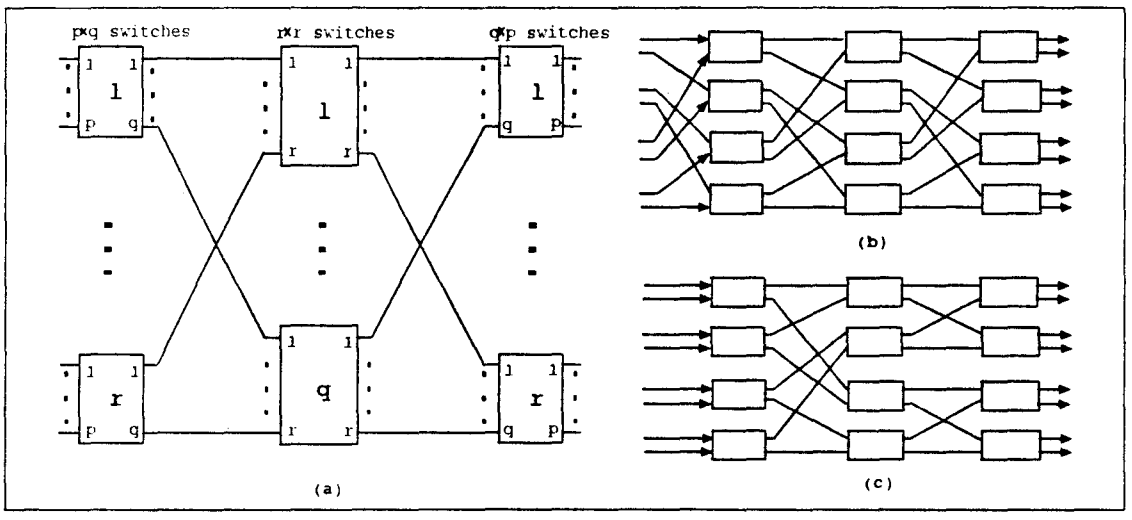

Figure 1.5 : Les réseaux multi-étages : (a) Clos à trois étages C(p,q,r), (b) Oméga, (c) Baseline.

Ce sont des réseaux où les commutateurs sont organisés en étages, les sorties de l'étage i étant directement connectées aux entrées de l'étage i+1. Les commutateurs les plus utilisés sont des crossbar 2 vers 2, c'est le cas des réseaux de Benès, Omega, Baseline etc... Dans ce type de réseau, le nombre d'étages est fonction du nombre de points d'entrées n : il est en général en $O(log_2 n)$. Bien qu'il réalise une économie en nombre de portes par rapport au crossbar, le délai de transmission des données ainsi que la complexité de la commande sont largement augmentés. Ce sont les raisons pour lesquelles ces réseaux ne sont utilisés que

dans des architectures de faible dimension. Un autre schéma de connexion a été proposé par Charles Clos ; c'est un réseau constitué de connecteurs $n \times q$, $r \times r$ et $q \times n$. Chacun des connecteurs est un réseau de crossbar. Un réseau de Clos a un nombre d'étages constant et égal à trois, il est réarrangeable et sans blocage. Ce type de réseau est utilisé dans les architectures issues du projet Supernode [MW90].

1.1.4 Le coût des communications

Nous allons modéliser ici le coût en communication des différents modèles :

1.1.4.1 modèle store and forward

Pour ce modèle, on définit le temps de communication $T_{voisin}(L)$ d'un message de longueur L entre deux processeurs voisins comme la somme :

1. d'un temps d'initialisation (ou *start up*) β correspondant à des procédures d'envoi/accusé de réception ou d'initialisation de registres (*buffers*) mémoire au niveau d'un processeur.
2. d'un temps de propagation $L\tau$ directement proportionnel à la taille L du message communiqué.

$$\boxed{T_{voisin}(L) = \beta + L\tau} \tag{1.2}$$

La bande passante b d'un lien est définie par $b = \frac{1}{\tau}$. C'est la quantité de données susceptible de parcourir ce lien par unité de temps.

Pour ce modèle, la fonction de routage est gérée par logiciel. Pour une communication entre 2 processeurs non directement connectés, un temps $\beta + L\tau$ est nécessaire à chaque traversée de processeur intermédiaire. Le temps total maximal dépend directement du diamètre D du graphe et est égal à :

$$\boxed{T_{stf}(L) = D(\beta + L\tau)} \tag{1.3}$$

1.1.4.2 modèle wormhole

On ne s'intéressera ici qu'au modèle wormhole, qui est le modèle effectivement le plus utilisé dans les systèmes actuels. Le temps de communication T_w est :

$$\boxed{T_w(L) = a + D\delta + L\tau} \tag{1.4}$$

où α est le temps requis pour l'initialisation du processus de communication (start up time), δ le temps de commutation d'un noeud intermédiaire et τ le temps de propagation d'un octet sur le lien. Pour le routage wormhole, il faut un temps α pour initialiser le transfert, un temps égal à $D\delta$ pour établir le chemin, et ensuite, il ne reste plus qu'à recevoir tout le message en un temps $L\tau$. Pour que cette expression soit correcte, on prend comme hypothèse que l'entête du message n'est pas comptée dans L. Enfin, on ne doit pas confondre le temps T_w avec le temps obtenu par utilisation d'une technique de pipeline en store and forward. Si le message était découpé en P paquets, le temps de communication serait alors de $(D+P-1)(\beta+\frac{L}{P}\tau)$ (où $\beta=\alpha+\delta$). Pour un découpage optimal du message en $P=\sqrt{\frac{L\beta(D-1)}{\beta}}$ paquets, on a un temps de $(\sqrt{(D-1)\beta}+\sqrt{L\tau})^2$. En comparaison avec le coût du modèle wormhole donné par l'équation 1.4, le routage wormhole est plus avantageux que le modèle store and forward pour de longs messages.

1.1.4.3 modèle crossbar

Le temps de communication d'un crossbar est :

$$\boxed{T_{cross}(L)=\gamma+L\tau} \tag{1.5}$$

où γ est le temps nécessaire pour l'établissement de la connexion. Ce temps comprend un temps de contrôle pour choisir le canal, un temps physique pour la traversée des multiplexeurs et un temps dû au retard de la propagation du signal électrique. A titre d'exemple, le temps de propagation des signaux électriques sur les conducteurs des circuits imprimés usuels est de l'ordre de $6ns$ par mètre, et le temps de traversée d'une porte logique de base dépend de la largeur du canal, qui est de l'ordre d'une nanoseconde actuellement. L'intérêt d'un tel réseau est, d'une part, que le temps de communication reste constant quelque soit le couple de processeurs, et, d'autre part, qu'une communication ne concerne que les processeurs intéressés. On économise les temps de commutation des noeuds intermédiaires si les temps d'initialisation sont égaux.

On ne mentionnera pas le temps de communication des réseaux multi-étages car il s'obtient aisément en se basant sur le modèle crossbar : un commutateur est un crossbar.

1.2 Le placement des tâches

Pour exécuter une application sur une machine parallèle, l'étape de traduction du programme source, écrit dans un langage de haut niveau en un code binaire (compilation , édition de lien) et utilisé dans les machines séquentielles, n'est plus suffisante. Puisque tout programme parallèle consiste en une collection de tâches (processus) coopérantes, avant toute exécution, il est nécessaire de savoir placer sur les divers processeurs, suivant une certaine stratégie, ces processus qui interagissent entre eux uniquement par échanges de messages. Pour cela, plusieurs critères peuvent prévaloir, à savoir :

- *les distances entre les processeurs supportant les processus communiquant*; le temps de communication entre 2 processeurs communiquant dans un réseau à topologie statique est déterminé non seulement en fonction de la longueur du message, mais aussi de la distance D qui les sépare.
- *l'équilibrage de la charge de travail des processeurs*; l'évaluation de la charge d'une application et sa répartition sur les processeurs permettent d'obtenir un parallélisme d'exécution et par conséquent de réduire le temps de traitement.
- *l'équilibrage de la charge des liens de communication*; ce critère est primordial puisqu'une surcharge d'un ensemble de liens de communication est souvent à l'origine des situations de conflits, qui ont pour conséquence d'augmenter le temps de latence.

Le placement doit aussi tenir compte des contraintes imposées par le programme comme les contraintes temporelles dans les applications temps réels, et par la machine comme la limitation de la taille mémoire d'un processeur ou la spécificité de certains processeurs (processeur chargé d'effectuer des opérations d'entrées/sorties).

La plupart des outils d'aide à la programmation d'architectures parallèles obligent l'utilisateur à définir lui-même son placement, ce qui n'est pas en général tâche facile. Ainsi, le programmeur ne peut ignorer l'organisation générale de l'architecture au moment de l'écriture de son programme ou à son exécution, d'où une forte dépendance de l'architecture cible. Cette méthode permet d'exploiter au mieux les ressources et la puissance de la machine, mais, si ce programme doit être exécuté sur une autre configuration du réseau d'interconnexion, alors il doit en général être réécrit ce qui met en cause la portabilité des programmes. C'est l'intérêt d'un placement automatique des processus qui permet au programmeur d'écrire ses programmes dans un langage à haut niveau, et c'est l'algorithme de

placement qui se charge de répartir les tâches. Les performances sont inférieures mais les programmes sont portables.

1.2.1 Définition

Soit un système parallèle formé d'un ensemble T de n processeurs $(t_1, t_2, \ldots, t_n)$ qui communiquent entre eux par l'intermédiaire d'un réseau d'interconnexion statique ou dynamique. Soit un programme composé d'un ensemble P de k processus $(p_1, p_2, \ldots, p_k)$ communiquant entre eux par échanges de messages. Le problème d'allocation peut être formellement décrit par une fonction de l'ensemble des processus vers l'ensemble des processeurs. Il existe donc n^k placements possibles ; on suppose qu'un processus est alloué à un et un seul processeur.

Le problème dans sa généralité est NP_complet, c'est à dire qu'il n'existe pas d'algorithmes exacts pour la résolution du problème, pour lesquels le temps maximum de résolution est borné par une fonction polynomiale de la taille du problème.

On distingue deux types de méthodes d'allocation suivant l'instant où le placement est décidé. Alors que **l'allocation statique** décide du placement des processus à la compilation ou au chargement du programme, **l'allocation dynamique** fait un placement à l'exécution par migration de processus.

1.2.2 Le placement statique

Nombreux sont les travaux de recherche sur le problème d'allocation des processus. La plupart des algorithmes d'allocation proposés sont statiques. Il faut aussi noter que peu d'entre eux ont été réellement mis en oeuvre dans des systèmes réels. Les travaux dans ce domaine se divisent en deux catégories dépendant de la topologie de l'architecture cible :

- topologie statique,
- topologie dynamique.

1.2.2.1 Placement dans une architecture à topologie statique

C'est dans ce domaine qu'il existe le plus de travaux, dans [MT91] les auteurs présentent une synthèse et une classification de ces différents travaux. La plupart des stratégies d'allocation statiques sont basées sur l'une des approches suivantes :

- la théorie des graphes,
- la programmation mathématique,
- l'utilisation d'heuristique.

Les approches par la théorie des graphes modélisent le problème par un graphe regroupant processus et processeurs. Elles tentent à minimiser le coût total de communication inter-processeurs par application des techniques de la théorie des graphes telles que : coupe minimale/flot maximal, recherche d'un homomorphisme faible, etc...

Les méthodes utilisant la programmation entière formulent le problème comme un problème d'optimisation combinatoire. Elles utilisent les techniques de programmation mathématique : programmation dynamique, Branch and Bound. etc...

Les deux premières approches donnent des solutions optimales mais sont gourmandes en temps CPU. L'utilisation d'heuristique permet d'obtenir des algorithmes rapides, mais fournit des solutions approchées. Malgré tout, ces algorithmes de placement statiques présentent des limites car ils ne tiennent pas compte du comportement dynamique des machines et des programmes lors de leur exécution sur une architecture cible :

- Il est supposé que l'état de charge du système distribué reste inchangé durant chaque exécution d'un programme. De telles situations peuvent exister dans le cas où les processeurs sont dédiés à des applications particulières comme les applications graphiques et les processus de contrôle dans les architectures temps réel. Par contre, si plusieurs programmes utilisateurs s'exécutent simultanément, il faut un mécanisme permettant de prendre en compte l'état courant du système ; d'où l'intérêt d'un algorithme d'allocation dynamique.

- Dans ce type d'algorithme, il est également supposé que l'architecture est fiable. La panne d'un processeur ou d'un lien pendant l'exécution d'un programme n'est pas prise en compte.

- Des algorithmes opérant sous ces conditions sont limités à des configurations statiques, c'est à dire qu'une configuration dynamique du réseau de processeurs n'est pas prise en compte et fait dégrader ainsi les performances du système dans le cas où la décision de reconfiguration apparaît par exemple dans la couche de communication.

- Les paramètres estimés dépendent des données d'entrée du programme; d'où l'inefficacité de représenter ces paramètres par des constantes. Dans le cas des applications où les coûts de communication et d'exécution ne dépendent pas des données d'entrée, comme par exemple dans quelques applications temps réel, ceux-ci peuvent être efficacement obtenus en faisant des expériences de simulation.

- Le placement statique de processus ne peut être appliqué pour des modèles de programmation dynamique où le nombre de processus et la structure du réseau de communication ne peuvent être connus à priori. Les processus créés dynamiquement doivent être placés à l'exécution en tenant compte éventuellement de l'état courant du système.

1.2.2.2 Placement dans une architecture à topologie dynamique

Dans le placement précédent, les algorithmes utilisés déterminent un placement optimal *en fonction* de la topologie statique de l'architecture cible.

Si le placement obtenu est optimal, il est uniquement pour la machine cible donnée, et n'est pas forcément le meilleur, puisque la topologie du graphe des processus propre à l'application n'est pas forcément adaptée à la topologie du réseau utilisée : les structures arborescentes sont particulièrement adaptées à l'évaluation de langages fonctionnels (FP par exemples [Bac78]) tandis que ceux en anneaux sont utilisés dans de nombreux algorithmes de service pour résoudre des problèmes tels que la synchronisation ou l'exclusion mutuelle [Ray85]. En fait, *on adapte le graphe des processus à la machine.*

L'approche inverse consiste à déterminer un graphe optimal puis à adapter la topologie de la machine cible sur ce graphe. La machine cible doit donc être capable de modifier sa topologie à la demande. Parmi les machines à topologie reconfigurable, on peut citer : la machine RPP [MMF+89], la machine CHiP [Sny82] [YA85] et les machines issues du projet Supernode [MW90].

Le problème du placement statique serait trivial, si chaque noeud pouvait être directement connecté à tous les autres noeuds, on aurait alors un réseau complètement interconnecté. Malheureusement, ces machines présentent des contraintes de package et de coût matériel qui limitent le nombre des connexions à un petit nombre (environs dix ou moins). Chaque processeur ne dispose donc que d'un nombre limité de connexions vers le réseau. Le réseau n'est donc pas capable de réaliser toutes les topologies possibles, mais à part certaines applications particulières (par exemple des applications de type neuronale [Bin90]) le graphe des processus optimal d'une application se présente comme un réseau non complète-

ment connecté. En plus, cette limitation est beaucoup moins contraignante que celle dûe à la nature statique des réseaux précédents. Dans [LS88], les auteurs présentent un algorithme qui détermine un placement optimal d'un graphe des processus donné en fonction du nombre de connexions disponibles par processeur.

Cette approche présente plusieurs avantages :

- le réseau reconfigurable permet de générer un ensemble de topologies. Il offre, d'une part, la possibilité d'avoir la meilleure machine possible pour l'exécution d'une application donnée, et, d'autre part, il permet de traiter une gamme plus étendue d'applications.

- un réseau de processus peut évoluer au cours de l'exécution, rendant le placement initial inefficace, ce qui introduit une baisse des performances dans une machine à topologie statique. Du fait du caractère reconfigurable de son réseau de communication, la machine peut évoluer en modifiant sa configuration. On peut éviter de faire migrer des processus, ce qui est généralement coûteux en temps de communication.

Nous verrons plus loin qu'il existe 2 types de machines reconfigurables. On les différencie par le moment où on effectue la reconfiguration. On parle de **reconfiguration synchrone** lorsque les connexions ne sont modifiables qu'à des moments précis : soit déterminés à l'avance, soit nécessitant des points de rendez-vous de tous les processeurs, et de **reconfiguration asynchrone** lorsqu'une connexion peut à tout moment être établie ou détruite à la demande des processeurs concernés.

1.2.3 Le placement dynamique

Le placement statique présente certaines limites : déterminé lors de la compilation, il ne prend pas en compte l'état courant du système, comme la création des processus dynamiques, l'évolution du graphe initial des processus, la dépendance des variables d'entrées et la panne d'un ou des éléments du système. Pour limiter une dégradation des performances dûe à ces facteurs, on a recours à la technique de placement dynamique.

Les algorithmes de placement dynamique sont en général plus complexes, puisqu'ils utilisent des informations sur l'état courant du système. Cependant, l'utilisation de ces informations peut donner des performances meilleures que celles des méthodes statiques [RF87].

Le placement dynamique est réalisé par l'association de 2 unités : une unité d'information et une unité de contrôle. L'unité d'information maintient l'information concernant l'état du système qui est utilisé par l'unité de contrôle pour effectuer le placement proprement dit des processus.

Les principaux problèmes posés dans la conception de l'unité d'information sont l'estimation de la charge des processeurs, et le trafic de communication induit par le protocole d'échange d'information. Le mécanisme de mesure de la charge d'un processeur constitue l'élément essentiel de l'algorithme de placement, étant donné que les états de charge des divers processeurs sont les seules sources d'information pour les autres éléments composant le processus d'allocation.

1.2.3.1 Mesure de la charge d'un processeur

L'estimation de la charge d'un processeur est un problème difficile, pour lequel aucune solution satisfaisante n'a été communément approuvée jusqu'à maintenant. La notion de la charge du processeur comprend non seulement la charge en traitement mais aussi la charge en communication.

Les décisions de placement ou de migration sont prises en fonction de l'état actuel du système. La plupart des algorithmes supposent qu'à tout instant l'état de chaque processeur peut être représenté par la valeur d'un paramètre unique, appelé **indicateur de charge**. Les caractéristiques d'un indicateur de charge idéal sont les suivantes [FZ86] :

1. la capacité d'estimer la charge courante ;
2. l'approximation de ce que sera la charge dans un **futur proche**, car le temps de réponse d'un processus affecté à un processeur dépend de la charge future, et non de la charge actuelle du processeur ;
3. la stabilité, afin de ne pas prendre en compte les fluctuations à court terme ;
4. la capacité à prendre en compte, le cas échéant, le caractère personnel des machines.

Dans [GDM91], une synthèse est faite sur les différentes méthodes d'évaluation de l'indicateur de charge. La plupart de ces méthodes ne prennent en compte que la charge en traitement du processeur alors que la charge en communication, qui nécessite des temps de traitement dans certains processeurs comme le Transputer, peut s'avérer non négligeable.

L'évaluation de la charge en traitement peut se faire suivant des paramètres comme le nombre ou la taille des processus. Ces paramètres sont facilement évaluables par des fonctions systèmes. Quant à l'évaluation de la charge en communication, elle est difficilement réalisable dans une architecture à passage de messages, puisqu'elle nécessite la connaissance des chemins empreintés par les messages.

Etant donné que la mesure de la charge d'un processeur se produit très souvent et que la charge reflète l'état courant du processeur, son évaluation doit être efficace afin de permettre à l'unité de contrôle d'effectuer un meilleur placement. Mais, cette efficacité se fait au prix d'une augmentation du coût de communication dans le réseau. La difficulté est de réaliser un compromis qui dépend de plusieurs facteurs ; un des plus importants concerne les propriétés du médium de communication.

1.2.3.2 Les algorithmes de placement

Le résultat d'une exécution de l'algorithme de placement dynamique est une prise de décision : faut-il placer ou migrer un processus, et sur quel processeur? On distingue 2 grandes classes d'algorithmes de placement : les centralisés et les distribués.

Dans les algorithmes centralisés, un seul processeur maître connaît l'état global du système. L'algorithme centralisé détermine l'état global du système en interrogeant chaque processeur sur son état courant. La méthode permet d'avoir une vue globale du système, mais crée au voisinage du processeur maître une surcharge en communication.

Parmi les algorithmes distribués, on distingue deux méthodes de résolution :

- Drafting algorithm,
- méthode du gradient.

Dans le **Drafting Algorithm** [NXG85], chaque processeur détermine son état de charge, qu'il caractérise par l'un des 3 états : faiblement, normalement et fortement chargé. L'algorithme consiste à équilibrer la charge de tous les processeurs, tout en minimisant la charge en communication nécessaire à l'application de l'algorithme. L'équilibrage des charges, c'est à dire la migration de certaines tâches d'un processeur fortement chargé vers les processeurs faiblement chargés n'a lieu qu'à la demande soit des processeurs fortement chargés, soit des processeurs faiblement chargés.

Dans la méthode du gradient, chaque processeur interagit uniquement avec ses voisins immédiats. Un équilibre global est atteint par propagation et raffinement de l'information sur la charge locale.

1.3 Conclusion

Le réseau de communication d'une architecture massivement parallèle constitue l'élément limitateur de performance d'une machine. L'amélioration de la performance d'exécution d'une application peut être obtenue en minimisant le coût en communication grâce à un bon placement des processus. Un placement efficace des processus est difficile à réaliser sur une architecture à topologie statique.

Par contre, les topologies dynamiques présentent des caractéristiques intéressantes qui permettent d'améliorer la qualité du placement : le coût de communication plus faible, facilités de placement statique des processus, qui permettent d'exploiter efficacement une gamme plus importante d'applications et qui présentent de bonnes performances pour les communications globales. Mais la réalisation de ces systèmes interconnectant un grand nombre de processeurs pose des problèmes de réalisation, de coût (crossbar) et de commande du réseau (réseau multi-étage).

Des progrès énormes ont été réalisés dans le domaine de l'intégration : la largeur de canal est passé de 50 microns en 1960 à 0.8 micron en 1990 et on a de bonnes raisons de penser que cette valeur atteindra la valeur de 0.1 micron en l'an 2000, cette diminution s'accompagne naturellement d'une augmentation en $O(n^2)$ en nombre de transistors intégrables. Parallèlement la largeur maximale de la surface intégrable ne cesse d'augmenter, elle est passée de 2 mm en 1960 à 13 mm en 1990. Ainsi, cette augmentation s'accompagne également d'une augmentation en $O(n^2)$ du nombre de transistors [HJ91]. Par ailleurs, des progrès sont réalisés dans le domaine de la transmission série des données : 1.8 Gb/s pour l'autobahn du bus VME ou sur fibre optique. Ces progrès techniques sont à la source d'un regain d'intérêt pour les systèmes à topologie dynamique.

En effet l'augmentation du nombre de transistors intégrables permet de réaliser un réseau de taille plus importante ; un crossbar 224×224 sert de réseau de communication au supercalculateur VPP500 de Fujitsu. Du fait de l'encombrement et de la limite en nombre de pattes du circuit [FWT82], la largeur de canal est souvent réduite à un. Dans un réseau statique un noeud dispose de beaucoup moins de points de sortie (moins de dix), on dispose donc des canaux plus larges. Pour obtenir un débit équivalent, il faut que la liaison série possède un débit largement supérieur.

Le projet **A**rchitecture à **R**éseau **P**artagé (ARP) consiste à étudier et à définir de quelle manière il est possible de définir et d'utiliser un réseau à commutation de circuits réduit à reconfiguration dynamique asynchrone, partagé par les processeurs, à commande distribuée et aisément extensible (de manière linéaire en nombre de circuits utilisés). Nous allons, dans ce qui suit, exposer les différentes réflexions qui nous ont conduit à la définition de notre architecture de réseau.

Chapitre 2

Architectures et réseaux reconfigurables

2 février 1993

Actuellement, les concepteurs et les constructeurs annoncent des performances des machines parallèles de l'ordre d'une centaine de GFlops. Mais, sur des applications réelles, ces machines ne dépassent guère quelques dizaines de GFlops. Ceci est dû à la non optimisation du code pour chaque processeur, mais surtout aux communications entre processeurs.

2.1 Problèmes liés à la communication

Il existe deux approches pour améliorer les performances de l'exécution d'une application sur une machine parallèle donnée : soit en réécrivant le programme en tenant compte des spécificités de la machine cible, soit en utilisant des algorithmes efficaces de placement des processus. La première approche n'est pas envisageable puisque les applications doivent rester transportables pour éviter de réécrire le code chaque fois que l'on change de machine.

Ceci implique donc l'utilisation des algorithmes de placement automatique. Pour obtenir la meilleure exécution d'une application donnée, un bon algorithme de placement doit, à la fois, réaliser l'équilibre des charges en calcul et des charges en communication des processeurs et augmenter la vitesse d'acheminement des messages dans le réseau. La répartition des charges en calcul présente de bonnes solutions dans la littérature [FP88] [MT91] [HS91], il n'en est pas de même pour celle liée à la communication, car la charge en communication est beaucoup plus difficile à évaluer, donc à répartir (§1.2.3.1). Le but final des algorithmes de placement est de réduire le temps d'exécution d'une application sur une machine donnée. Il est nécessaire, pour cela, de prendre en compte un paramètre crucial : le mouvement des données.

2.1.1 Le mouvement des données

Nous introduisons ce paragraphe par une citation de J.E. Smith dans [SHH90] "Le mouvement des données est le problème le plus difficile à résoudre pour l'utilisation efficace des machines massivement parallèles. Les performances des futures machines seront finalement mesurées à partir de la vitesse de mouvement des données dans le système et au travers du réseau de communication".

En effet, la vitesse d'exécution des processeurs et la taille des données traitées augmentent beaucoup plus vite que la vitesse de mouvement des données entre les processeurs communicants. Le rapport entre communication et calcul détermine la granularité des algorithmes. On peut distinguer 2 types d'algorithmes [CR87]:

- les algorithmes **Compute-bound** : où le nombre de calculs élémentaires est plus grand que le nombre de transferts de données en entrée-sortie.
- les algorithmes **I/O-Bound** : où le nombre de transferts de données en entrée-sortie est plus grand que le nombre de calculs élémentaires.

Nous ne nous intéresserons, dans le cadre de ce travail, qu'aux machines exploitant des algorithmes du type compute-bound. En effet, l'exécution des algorithmes I/O-Bound est la plupart du temps effectuée sur des machines dédiées, et seuls ceux qui possèdent un graphe des processus réguliers (systolique, traitement d'images) peuvent être traités efficacement. Les performances importantes des architectures dédiées sont dûes à l'adaptation quasi-parfaite de la structure de calcul à la structure spécifique d'une application. Cependant, ces machines souffrent de leur adaptation et ne permettent pas l'exécution efficace d'une application quelconque.

Le propre des machines à vocation généraliste est de pouvoir exécuter un grand nombre d'applications de manière efficace. L'obtention des performances nécessite de trouver des solutions pour pallier l'écart entre le temps d'exécution et le temps de communication.

Naturellement, nous pouvons résoudre ce problème en réduisant la puissance des processeurs, mais dans ce cas, à quoi bon avoir des processeurs aussi performants que le processeur Alpha ?! L'autre solution consiste :

- d'une part, à limiter la charge du réseau de communication. Les accès aux données locales sont beaucoup moins coûteux que ceux aux données distantes, qui nécessitent des communications par le réseau. En regroupant le *contexte* (les données locales et le code) d'un processus dans sa mémoire

locale, le réseau de communication n'est alors utilisé que pour transmettre les données partagées.

- et, d'autre part, à augmenter la vitesse de mouvement des données. La performance d'un réseau de communication lors d'une exécution est liée à la qualité du placement et à la vitesse d'acheminement des messages, qui dépend de la topologie du réseau, de l'algorithme de routage et de la technologie utilisée.

2.1.2 La nécessité d'un placement dynamique

La plupart des machines massivement parallèles à mémoire distribuée utilisent un réseau à topologie statique comme support de communication. Pour exploiter au mieux la puissance de calcul de la machine, l'utilisateur est contraint de placer les processus communicants sur des processeurs physiquement connectés, et ce, afin de minimiser le temps de communication. Dans le paragraphe consacré au placement, nous avons vu qu'un tel placement n'est pas efficace tout au long de l'exécution, étant donnée la nature non déterministe des applications.

Pour améliorer les performances, un placement dynamique, c'est à dire une migration de certains processus, est nécessaire, ce qui s'accompagne d'un déplacement important de données dans le réseau. La migration d'un processus consiste à déplacer le code du processus, les données, la pile et les informations de contrôle. Cette opération, très coûteuse en temps de communication, ne sera réalisée que si le gain en performance, apporté par le nouveau placement, recouvre le temps nécessaire à sa réalisation. Les algorithmes de placement dynamique doivent à la fois équilibrer la charge en traitement et minimiser la charge en communication. C'est à dire qu'une décision de migration d'un processus donné peut être prise soit parce que le processeur associé est beaucoup plus chargé en traitement par rapport aux autres processeurs, soit parce qu'il entraîne une charge en communication importante du fait de sa position géographique.

De manière générale, l'équilibre de la charge en calcul nécessite une migration des processus. Par contre, la réduction de la charge en communication peut être réalisée soit par migration, soit par reconfiguration du réseau. La deuxième solution évite d'introduire dans le réseau de communication une quantité importante de données correspondant au contexte du processus à migrer.

Par ailleurs, nous avons vu que les réseaux à topologie dynamique comme le crossbar (§1.1.4) possèdent un temps de communication constant quelle que soit la position géographique du processeur adressé, et, que, lors d'une communication, les processeurs concernés disposent du moyen le plus efficace qui soit : une liaison

physique directe. La distance dans un tel réseau est constante et égale à un. Tous les processeurs sont alors virtuellement voisins. Cette propriété simplifie fortement le placement statique.

En placement dynamique, il peut être intéressant de modifier seulement une partie du graphe des processus existant ; pour réaliser cette modification, certaines connexions doivent être détruites ou créées, et une migration d'un processus nécessite l'établissement d'une connexion provisoire entre le processeur source et le processeur destinataire ; le réseau de communication utilisé doit donc offrir la possibilité de modifier les connexions sans changer celles qui existent (réseau sans blocage), et d'établir une connexion entre un couple de processeurs libres (réseau de Banyan).

Les machines utilisant un réseau de communication reconfigurable ont fait l'objet de nombreuses études, qui ont donné naissance à des machines reconfigurables. Nous allons, dans ce qui suit, présenter quelques unes de ces machines.

2.2 Les machines reconfigurables existantes

Parmi les machines reconfigurables, nous donnons ici une description rapide des trois plus représentatives, qui se différencient par leurs réseaux à topologie reconfigurable : celui de Supernode est un réseau multi-étages, le réseau de CHiP est composé d'une grille de commutateurs, et la machine RPP utilise un crossbar complet.

2.2.1 Le projet Supernode/Méganode

Le projet [MW90] a été entrepris dans le cadre du projet de recherche européen ESPRIT P1085. La machine (ou module) Supernode de base est organisée autour d'un seul étage de communication composé de crossbars "Supernode" (SN-Switch 72×72). On distingue deux modes de fonctionnement des modules de base : le fonctionnement autonome et l'association en machines hiérarchisées.

En fonctionnement autonome, chaque module peut interconnecter jusqu'à 64 transputers T800 d'Inmos. L'ensemble du module est géré par un contrôleur T414 (noté C sur la figure 2.1) qui commande le réseau via son bus de mémoire externe. Un bus de contrôle (non représenté) sert de support pour le protocole de communication entre le contrôleur et les transputers de travail (notés T). Toute demande de reconfiguration passe par le contrôleur via ce bus, il permet également au contrôleur d'effectuer des primitives de synchronisation et de diffusion.

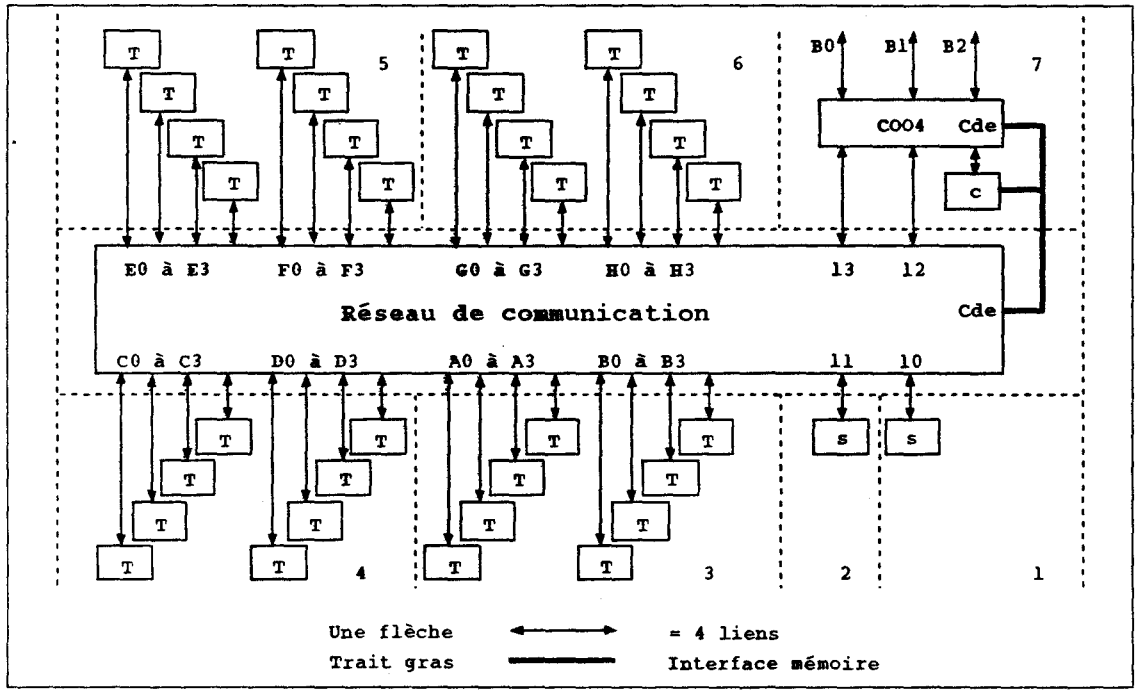

Figure 2.1 : Organisation d'un noeud simple

En fonctionnement hiérarchisé, le nombre de Transputers par module est réduit de moitié. Les liens, qui étaient destinés à la connexion des Transputers, ainsi libérés sont utilisés par un réseau de commutation de niveau supérieur qui regroupe tous les modules de base. L'ensemble du réseau de communication constitué des crossbars supernode et C004 sont interconnectés entre eux pour former un réseau de Clos.

L'ensemble est coordonné par un transputer superviseur qui pilote le deuxième niveau de commutateurs. Les contrôleurs de module sont à leur tour connectés en tant qu'esclaves sur un bus de contrôle de niveau supérieur dont le transputer superviseur est le maître.

Ce type de machine a été conçue à l'origine pour permettre la reconfiguration statique. Si la reconfiguration dynamique asynchrone peut se faire facilement sur le module de base, il n'en est pas de même pour la version hiérarchisée. Car, en plus de la complexité intrinsèque de l'algorithme de pilotage des réseaux de Clos, elle présente une difficulté de gestion des deux niveaux de contrôleurs, ce qui nécessite une gestion centralisée et ralentit la vitesse de reconfiguration.

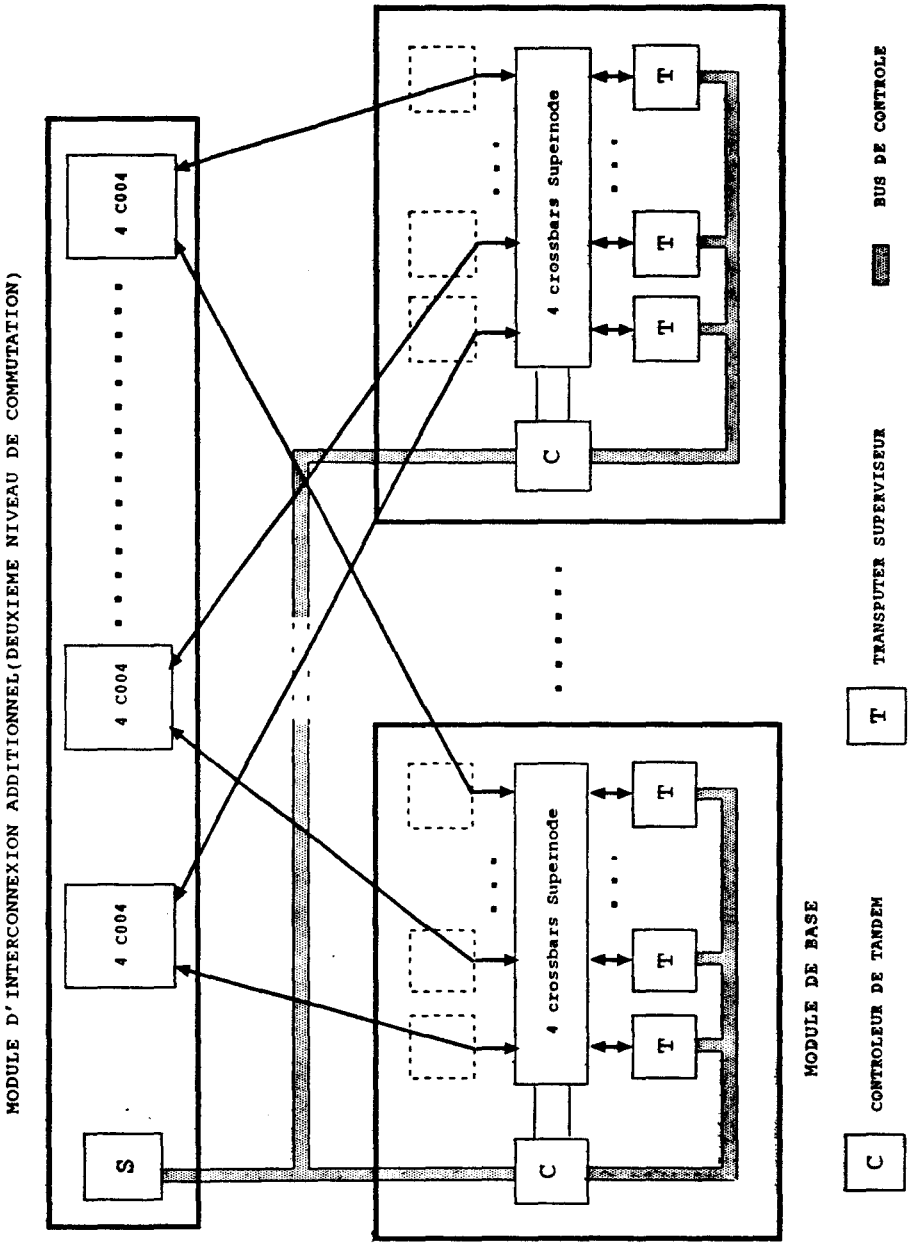

Figure 2.2 : Vue d'ensemble d'une machine hiérarchisée

2.2.2 La machine CHiP

La machine CHiP [Sny82] (Configurable Highly Parallel), développée à l'université de Purdue, est une machine statiquement reconfigurable. Elle est constituée d'un réseau de processeurs interconnectés par l'intermédiaire d'une matrice de circuits d'aiguillages (switch) programmables. Chaque circuit programmable est capable d'établir une connexion statique entre les chemins de données qui arrivent sur lui. Une mémoire locale au circuit programmable lui permet d'enregistrer plusieurs modèles de configuration couramment utilisés.

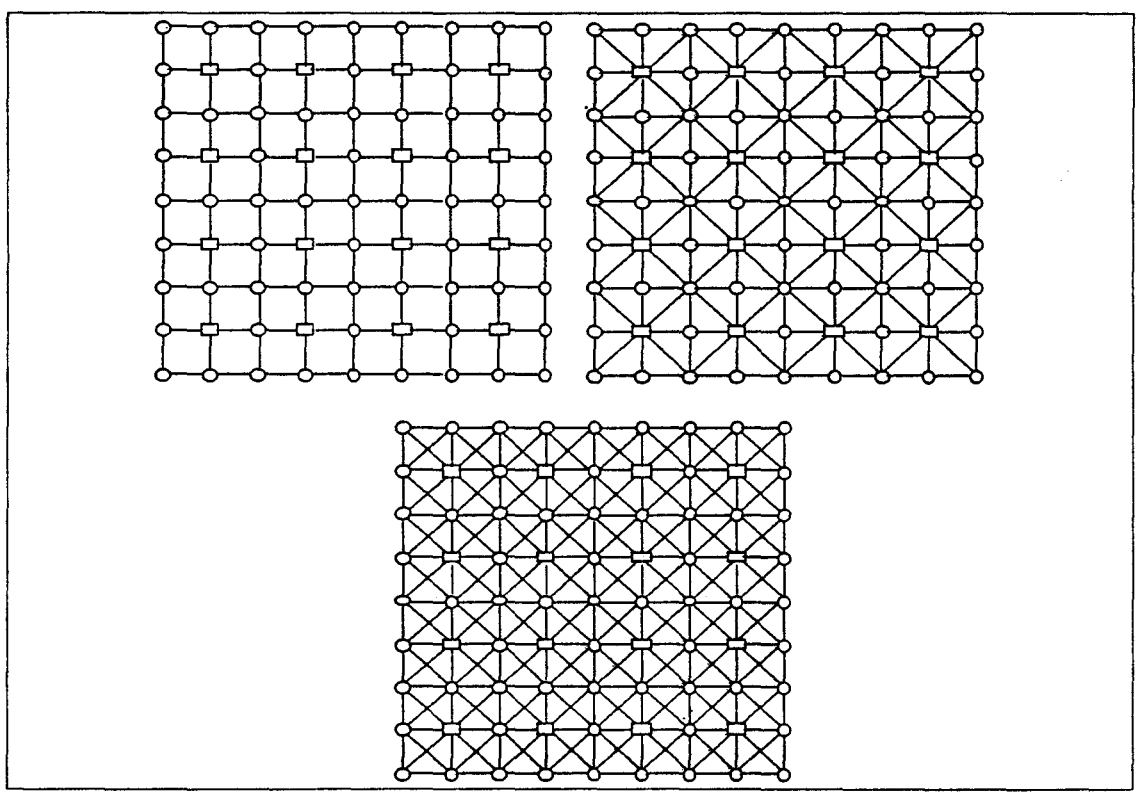

Figure 2.3 : Structures de la machine CHiP : les cercles représentant les circuits d'aiguillage, et les carrés, les processeurs

A la compilation, le compilateur produit non seulement un code exécutable pour le processeur, mais aussi des données destinées à la configuration des circuits programmables. La mémoire du circuit et la mémoire du programme du processeur sont chargées en parallèle par l'intermédiaire d'un autre réseau d'interconnexion. L'ensemble des processeurs configurés travaillent de façon synchrone.

La machine CHiP est particulièrement adaptée pour des applications systoliques et en VLSI. Elle permet également d'établir plusieurs partitions, autorisant ainsi

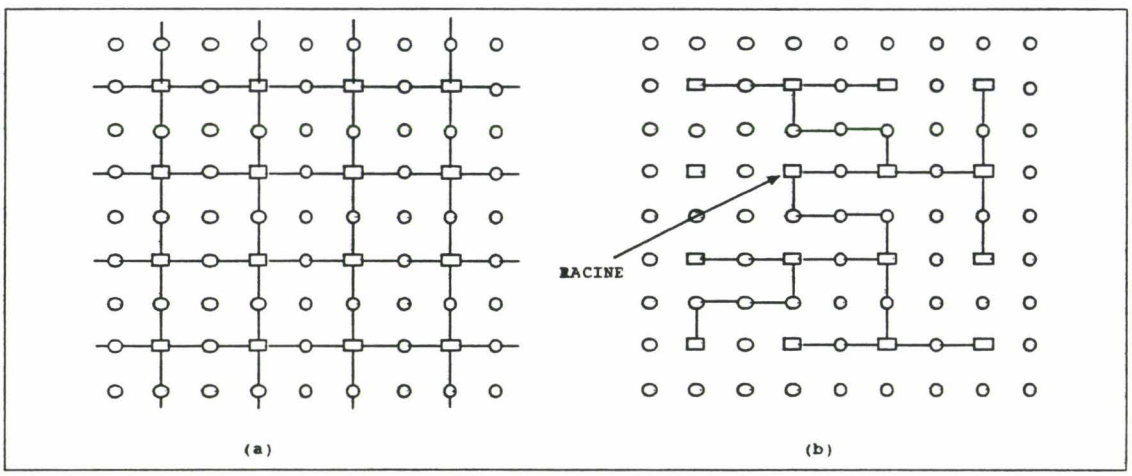

Figure 2.4 : La machine CHiP configurée en mailles (a) et en arbre binaire (b)

à plusieurs applications de s'exécuter simultanément. Elle tolère bien les pannes au prix d'une augmentation de temps de communication (plus de circuits à traverser).

Mais la machine CHiP ne permet pas la reconfiguration dynamique. Fondamentalement, les processeurs sont interconnectés en grille et les techniques de routage dans les machines à passage de messages, utilisant la même topologie, ont beaucoup évolué. Plus particulièrement, le routage Wormhole, en termes de temps de communication, approche les performances du réseau de la machine CHiP, puisqu'une fois que le chemin est établi par l'entête, le corps du message progresse suivant un chemin déterminé. Et le routage Wormhole permet, de plus, une utilisation plus souple du réseau de communication.

2.2.3 La machine RPP

La machine RPP [MMF+89] [MFM+89](Reconfigurable Parallel Processor) développée à l'université de Kyushu, est une machine MIMD à 128 processeurs. Le réseau de communication est un crossbar 128×128. La machine RPP peut se comporter comme un système à mémoire distribuée, dans ce cas le crossbar sert de médium de communication entre processeurs. Elle peut également être configurée en un système à mémoire commune où le crossbar interconnecte les bancs mémoires aux processeurs.

L'élément principal de la machine est son réseau de communication. Le crossbar 128×128, trop grand pour être intégré dans un seul boîtier, est constitué par un assemblage de 256 crossbars $(8 \times 8) \times 8$ (car les données sont sur 8 bits), dont on

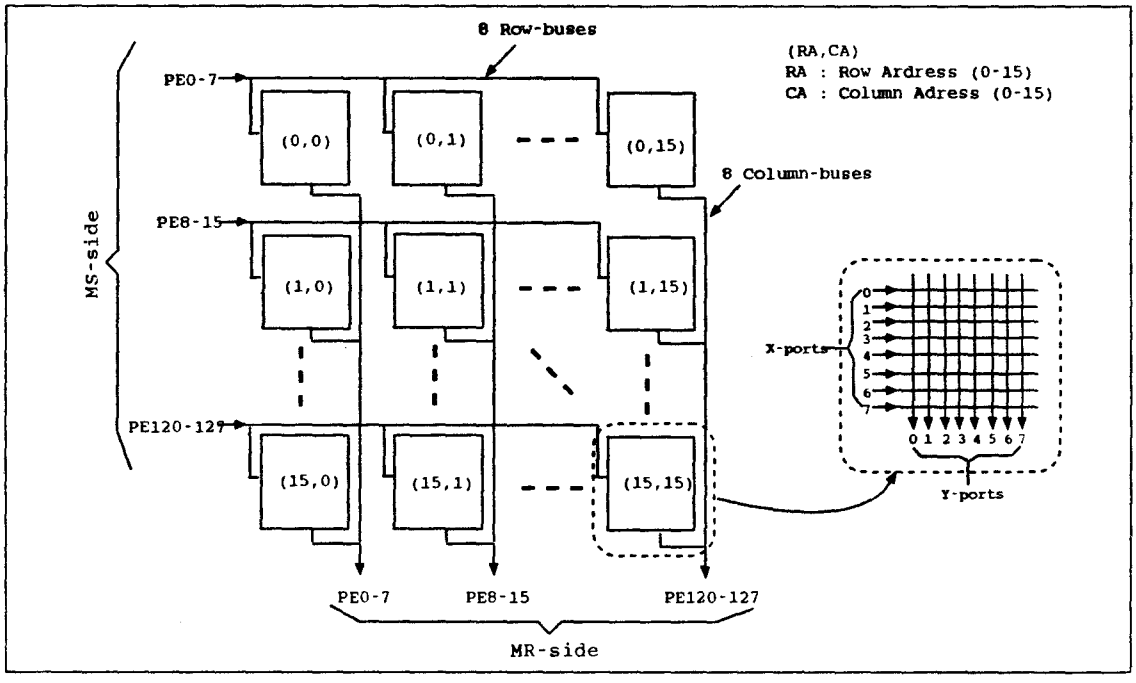

Figure 2.5 : L'arrangement du crossbar 128 × 128 de la machine RPP

voit le schéma de principe (figure 2.5). Le réseau fonctionne suivant 3 modes :

- *le mode demande* : les connexions sont effectuées de façon asynchrone à la demande des processeurs.
- *le mode preset* : les connexions sont établies de façon synchrone. Il est possible de mémoriser plusieurs configurations et de fixer la durée de chaque configuration. Ainsi la topologie du réseau peut changer statiquement selon les différentes topologies prédéterminées à des moments fixes.
- *le mode partitions* : l'ensemble des 128 crossbars peut être partitionné en 16 groupes et chaque groupe peut fonctionner suivant les 2 modes précédents.

La machine RPP apporte de bons éléments de réponse sur l'étude des machines reconfigurables dynamiquement. La modularisation du crossbar par des crossbars identiques de petite taille autorise une bonne extensibilité jusqu'à 128. Mais, nous pouvons nous interroger sur la dimension d'un tel système. Chaque bus ligne doit avoir au moins une largeur de 144 (par processeur, on comptabilise 8 lignes pour les données, 4 lignes pour la sélection du crossbar, 3 pour la sortie, et 3 lignes de contrôle). La résolution des contentions en mode demande nécessite un arbitrage sur 2 niveaux : résolution au niveau du module crossbar sur les conflits d'accès à

la même sortie et résolution des conflits d'accès à la même entrée du processeur par les sorties des crossbars de la colonne.

2.2.4 Comparaison des systèmes présentés

Le tableau 2.1 récapitule les différentes caractéristiques des 3 machines. Ces 3 machines possèdent un réseau de communication permettant la reconfiguration statique. La reconfiguration dynamique n'est possible que sur Supernode et RPP. Du point de vue de la performance, la meilleure solution est apportée par la machine RPP grâce à un arbitrage distribué : l'arbitrage est fait sur chaque crossbar pour le choix de la sortie, et également au niveau de chaque entrée processeur pour le choix de l'entrée. Le schéma d'arbitrage autorise des connexions en parallèle, mais ceci au prix d'une complexité matérielle élevée. Par ailleurs, la machine RPP, de par sa conception, ne permet pas une extension facile. Quant à la machine Supernode, elle réalise un bon compromis entre le temps de latence et le coût par l'utilisation d'un réseau de Clos réduit. Le réseau de Clos permet une bonne extensibilité, mais il est pénalisé par la lenteur du réseau hiérarchique de contrôleurs et par la complexité de la commande du réseau de Clos lui-même.

Tableau 2.1 : Comparaison entre Supernode, CHiP et RPP

	reconfiguration dynamique	extensibilité	coût matériel	latence	arbitrage
Supernode	moyenne	bonne	moyen	moyen	centralisé
CHiP	non	facile	faible	mauvais	centralisé
RPP	bonne	non	important	bonne	distribué sur 2 niveaux

La réalisation d'un bon réseau à reconfiguration dynamique nécessite la prise en compte de tous ces paramètres, soit en réalisant un compromis (cas supernode), soit en privilégiant certains paramètres par rapport aux autres, comme la performance. Dans le projet ARP, nous prenons comme critères principaux, la performance et la facilité de reconfiguration dynamique, et nous apportons quelques éléments de réponse pour réduire le coût matériel et permettre l'extensibilité.

2.3 Le réseau de communication d'ARP

Nous avons vu que les architectures reconfigurables présentent des propriétés intéressantes qui permettent d'améliorer la qualité du placement (§1.3). Le projet ARP, présenté dans le cadre de ce travail, définit une architecture à mémoire distribuée, où les processeurs communiquent par un réseau à reconfiguration dynamique et asynchrone. La commande du réseau est faite de manière distribuée, et le réseau est extensible.

Le but final du projet ARP est d'étudier quelles sont les améliorations que peut apporter une telle architecture sur le placement (statique ou dynamique) des processus. Mais, ceci sort du cadre de ce travail qui consiste à présenter l'architecture elle-même, à démontrer sa faisabilité et à donner ses performances.

Un des éléments clés d'une architecture reconfigurable est son réseau de communication. Nous allons, dans un premier temps, fixer le choix du type de réseau reconfigurable.

2.3.1 Le choix du réseau reconfigurable

Nous avons introduit les réseaux reconfigurables ou à topologie dynamique dans le §1.1.3. Il existe trois catégories de réseaux : le réseau multi-étages, le réseau multibus et le réseau crossbar. Les trois types de réseaux sont couramment utilisés dans les machines reconfigurables actuelles (cf §2.2), et en règle générale, aucun des trois types de réseaux n'est prépondérant. Ils possèdent chacun des qualités et des défauts, le choix dépendant uniquement des objectifs et des contraintes fixés.

Nous allons, dans un premier temps, détailler les différents réseaux existants, puis nous exposerons les différentes raisons qui nous ont conduit au choix de notre réseau.

2.3.1.1 Les réseaux multi-étages

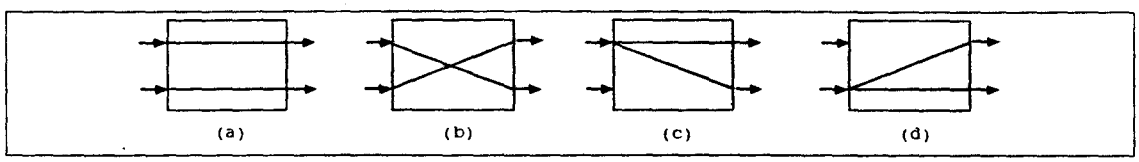

Figure 2.6 : Les différents états d'un commutateur 2×2

L'élément de base des réseaux multi-étages est le commutateur $n \times m$ (de n entrées vers m sorties), dont le plus utilisé est le commutateur 2×2. Ce dernier a été très largement étudié [CF80] [Agr82]. Ce commutateur réalise 2 fonctions de base : continuer sur les mêmes lignes (figure 2.6.a) et permuter les lignes (figure 2.6.b). Mais, dans certains cas, des fonctions supplémentaires, telles que la diffusion supérieure (figure 2.6.c) et la diffusion inférieure (figure 2.6.d) [Law75], lui sont ajoutées.

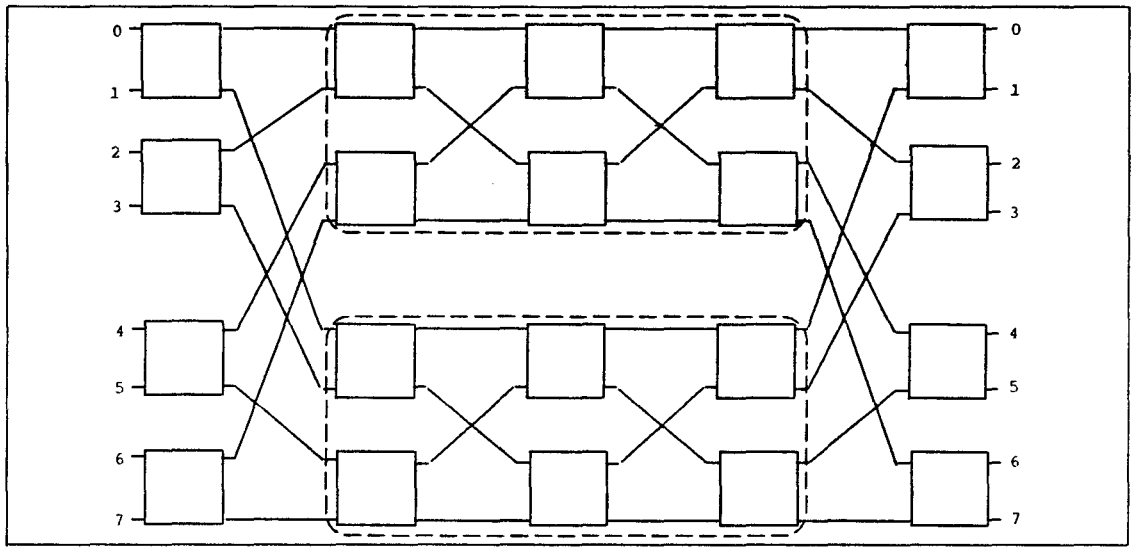

Figure 2.7 : Le réseau de Benès de dimension 2

Ce commutateur est à la base de la majorité des réseaux multi-étages réarrangeables : Benès, Omega, Butterfly, Baseline [Fen81]. Ces réseaux ont été développés à l'origine pour interconnecter les processeurs aux mémoires dans le contexte des machines à mémoire commune [Bat76] [GGK+83]. Les réseaux réarrangeables bloquants (Omega, Butterfly) sont les plus souvent utilisés car ils réalisent des configurations régulières, comme la permutation et le shuffle qui suffisent à l'exécution de la plupart des algorithmes de calcul comme la Transformée de Fourrier Rapide, la transposition de matrice et l'évaluation polynomiale [Law75] [Bat76] [Lei92]. Par ailleurs, ces réseaux utilisent un nombre de commutateurs très inférieur à celui des réseaux non bloquants, et possèdent une commande plus simple.

Mais, ils deviennent difficilement utilisables dans le contexte des machines à mémoire distribuée, où les configurations ne sont pas toujours régulières. Les réseaux multi-étages non bloquants permettent de combler cette lacune. Le réseau non bloquant le plus connu est le réseau de Benès [Ben62] ; la figure 2.7 montre un réseau de Benès de dimension 2, qui comporte un nombre de commutateurs égal

à $\frac{n}{2}(2Log_2 n - 1)$, où n est le nombre d'entrées au lieu de n^2 commutateurs pour un crossbar [Fra81]. Malgré ce gain en complexité matérielle, le réseau de Benès présente quelques difficultés de mise en oeuvre.

L'algorithme de commande du réseau s'exécute la plupart du temps de manière centralisée [And77] [Len78]. Il est possible de commander le réseau étage par étage [NS81], mais l'algorithme n'aboutit pas toujours à la configuration demandée. Il n'est pas possible de commander un tel réseau de façon distribuée, à moins de contourner la difficulté en faisant une commutation de messages [GGK+83]. Par ailleurs, les réseaux sont réarrangeables, ce qui implique une modification des connexions existantes. Un réseau multi-étages est très difficilement extensible puisque, comme dans un hypercube, l'extension du réseau oblige à doubler le nombre d'entrées et à modifier la totalité ou une partie des connexions existantes.

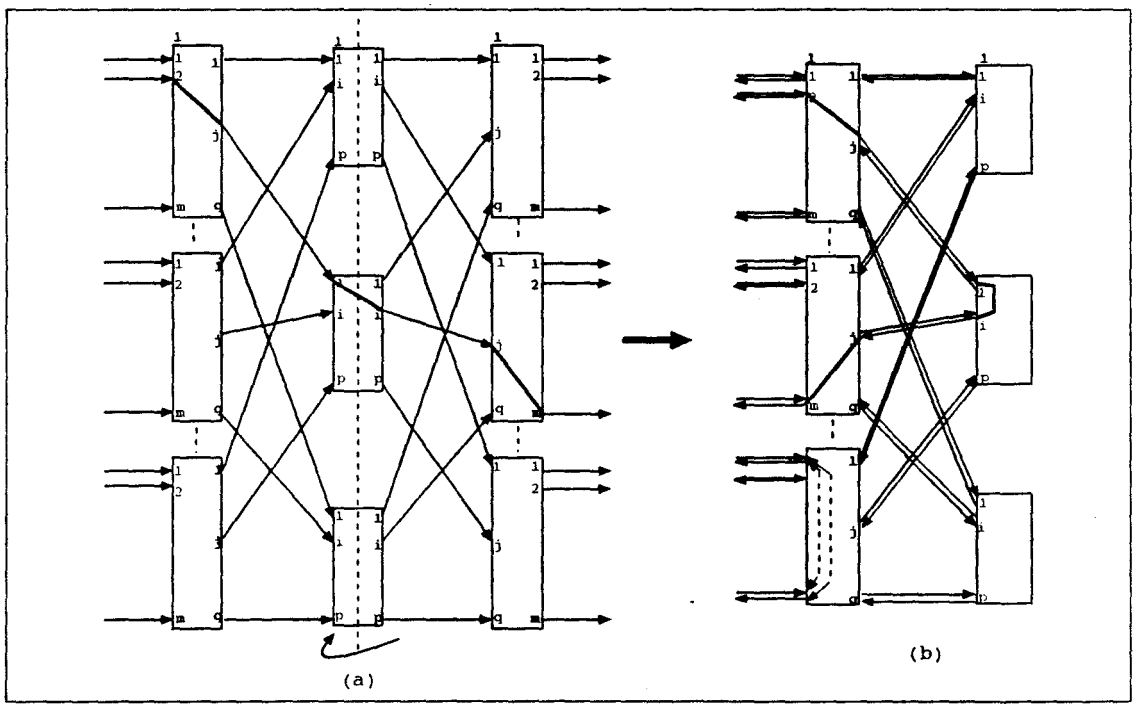

Figure 2.8 : Le réseau de Clos et le réseau unilatéral de Clos des machines Supernode

Le réseau de Clos [Clo53] est constitué par une association de 3 sortes de commutateurs ou crossbars : $m \times q$, $p \times p$ et $q \times m$. Le réseau de Clos est réarrangeable si $q \geq p$, et il est de plus non bloquant si $q \geq 2p - 1$.

Le réseau de Clos présenté dans la figure 2.8.a, dont les liens d'entrée et de sortie occupent les deux extrémités, peut être qualifié de bilatéral. Le réseau de Clos peut être transformé en réseau unilatéral (entrées et sorties du même

côté) : il suffit de replier le réseau de Clos initial suivant son axe de symétrie et de remplaçer chaque paire de crossbars superposés (m vers q et q vers m) des premier et troisième étages par un crossbar unique de m+q vers m+q. Le réseau de Clos ainsi obtenu possède les mêmes propriétés de réarrangeabilité et d'absence de blocage. L'intérêt de ce nouveau réseau est de permettre de relier des entrées et sorties appartenant à un même crossbar sans faire intervenir le deuxième niveau de commutateur.

2.3.1.2 Le multibus

Le bus est le support de communication le plus simple à utiliser, mais le débit possible sur ce bus limite rapidement le nombre de processeurs du système à une dizaine de processeurs. Pour des raisons physiques [LHSP90], la fréquence de fonctionnement n'augmentera pas en proportion avec la technologie et le nombre de points de branchement restera limité. Sur le Futurebus, le nombre de charges possibles (cartes), que l'on peut y raccorder, est limité à 20 [Jon91] pour un débit de l'ordre de 100 Mo/s [ES88]. L'augmentation du nombre de processeurs ne peut se faire qu'en multipliant le nombre de bus.

Un exemple de réalisation est donné dans [MHW87] pour une architecture à mémoire commune, où la mémoire est découpée en bancs et où les processeurs accèdent aux bancs mémoires par un multibus. Une fois connecté, un processeur dispose d'une liaison directe avec le banc mémoire choisi, mais cette solution nécessite un arbitrage sur 2 niveaux : un arbitrage pour l'accès des processeurs à un bus et un second pour l'accès aux bancs mémoires.

L'approche multibus conserve les avantages du bus et augmente le nombre de processeurs du système. A un instant donné, un bus est soit libre, soit occupé par un couple (processeur, mémoire). Supposons que chaque processeur dispose d'une entrée et d'une sortie séparées, qui lui permettent d'émettre et de recevoir en même temps. Si nous réservons un bus par processeur, pour que chaque processeur puisse disposer d'un débit maximal, le réseau devient alors un crossbar dont le coût est en $O(n^2)$, avec n le nombre de processeurs.

Une solution plus économique consiste à considérer un bus comme un ensemble de segments où l'on peut effectuer plusieurs communications différentes simultanément. C'est l'approche bus sécable [Gec77] [PBG90] [PKG92].

Les processeurs sont disposés en ligne au dessus du réseau. Celui-ci est construit par la mise en parallèle de lignes constituées de segments. Ces segments sont reliés par des interrupteurs statiques configurables à 4 ports. Ces interrupteurs ont un port d'entrée et un port de sortie connectés respectivement à la sortie et à l'entrée du processeur, et deux ports d'entrées-sorties connectés aux segments situés sur

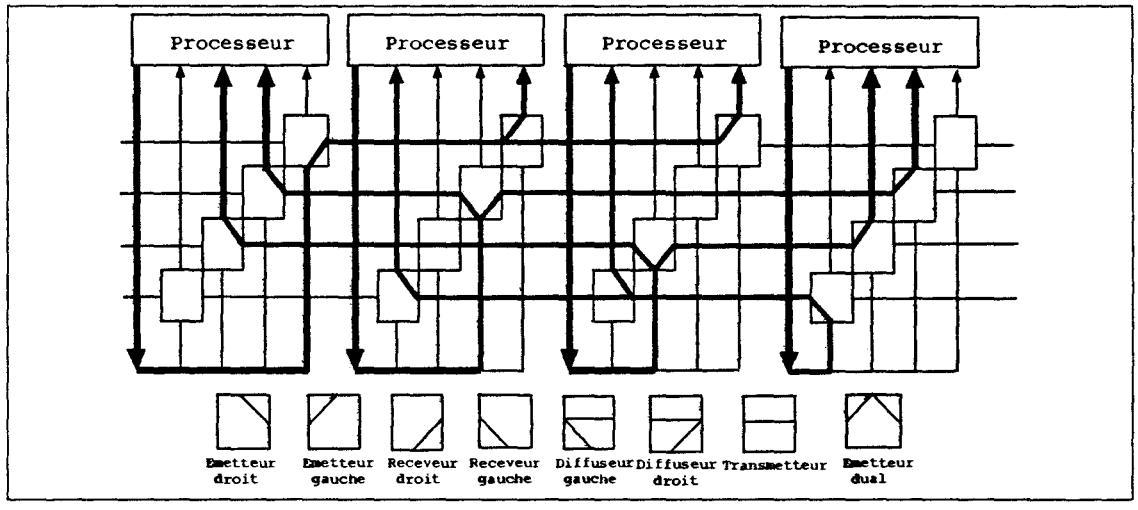

Figure 2.9 : Architecture à bus sécable et les commutateurs du bus

les côtés droit et gauche. Dans chaque interrupteur, toutes les combinaisons de connexion d'un port utilisable en entrée vers des ports utilisables en sortie sont possibles. L'approche par bus sécable permet une utilisation efficace du bus dans un contexte totalement synchrone, mais l'établissement des liaisons nécessite une connaissance de l'état global du réseau, donc de l'utilisation d'une unité superviseur.

2.3.1.3 Les réseaux crossbar

La forme générale d'un réseau de crossbar à N entrées et M sorties se présente comme une matrice $N \times M$, où les ports d'entrée sont situés à gauche et les ports de sortie en bas. La connexion de l'entrée i et de la sortie j nécessite juste l'activation du commutateur(i,j) situé à l'intersection de la ligne i et de la colonne j.

Le réseau est non bloquant, car il existe toujours un chemin entre tout couple d'entrée et de sortie libres, et la connexion de ce couple ne modifie pas les connexions déjà établies. Par ailleurs, il n'existe qu'un seul chemin entre tout couple d'entrée et de sortie; cette propriété facilite la commande des commutateurs, mais rend le réseau vulnérable aux pannes; la panne du commutateur(i,j) rend la sortie j inaccessible depuis l'entrée i. Nous désignerons, dans ce qui suit, ce réseau comme **un réseau de crossbar CCU** (à chemin de connexion unique).

Dans le cas où le nombre d'entrées est égal au nombre de sorties, une deuxième forme de réseau peut être définie : la matrice de commutateurs est en $N \times \frac{N}{2}$, les

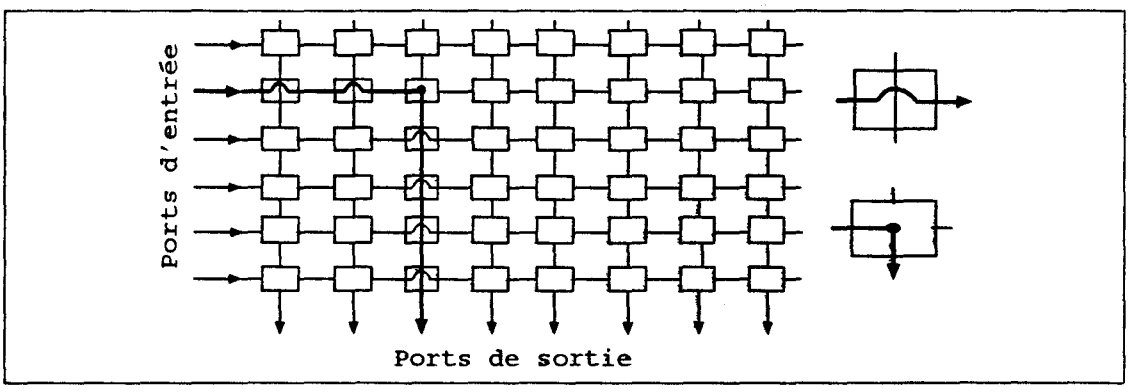

Figure 2.10 : Le réseau de crossbar à chemin de connexions unique

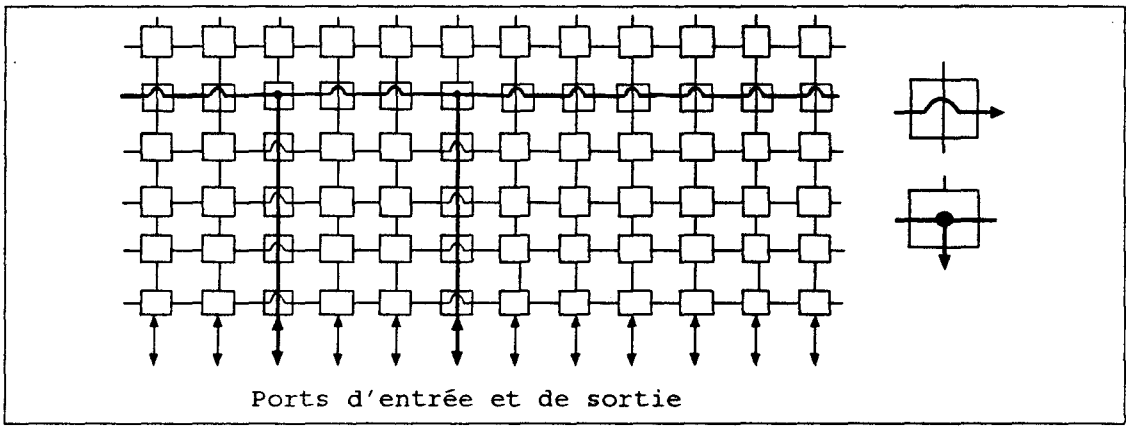

Figure 2.11 : Le réseau de crossbar à chemins de connexions multiples

ports d'entrée et les ports de sortie sont confondus et peuvent être représentés en bas du réseau. La connexion d'une entrée i vers une sortie j nécessite l'activation de commutateurs(k,i) et (k,j), où $1 \leq k \leq \frac{N}{2}$, et tous les commutateurs(k,x) sont désactivés avec $x \neq i, j$. Nous désignerons ce réseau comme **un réseau de crossbar CCM** (à chemins de connexions multiples), puisque chaque couple d'entrée et de sortie peut disposer de plusieurs choix possibles pour réaliser la connexion. Par ailleurs, les commutateurs, différents des commutateurs précédents, offrent la possibilité de diffuser vers un sous-ensemble de sorties.

Ce réseau tolère mieux la panne que le crossbar CCU. Il a été démontré que le crossbar CCM reste non bloquant avec au plus $\frac{N}{2} - 1$ commutateurs en panne, et reste réarrangeable avec au plus $\frac{N^2}{4} - N + 1$ commutateurs en panne [VC92].

2.3.1.4 Le réseau ARP

Nous avons porté notre choix sur le crossbar CCM pour plusieurs raisons :

- La disposition des commutateurs du crossbar CCM en matrice facilite sa réalisation ; un crossbar CCM $N \times M$ peut être réalisé par un assemblage de crossbars CCM $K \times M$, avec $K < N$. Cette propriété apporte une extensibilité du réseau de communication.
- Le crossbar CCM est un réseau non bloquant et la connexion de 2 éléments i et j du réseau nécessite uniquement l'activation des commutateurs (k,i) et (k,j). Par conséquent, la réalisation des connexions peut être faite de façon totalement distribuée.
- Le nombre d'éléments de commutation à traverser reste constant quels que soient les processeurs à connecter.
- Dans un crossbar CCM, chaque port d'E/S peut se connecter **en réception** sur n'importe quelle ligne de communication donnée. Par conséquent, il est possible de réaliser des diffusions dans un tel réseau en attribuant une même ligne de communication à tous ou à un ensemble de ports d'E/S.

2.3.2 Le réseau de communication du projet ARP

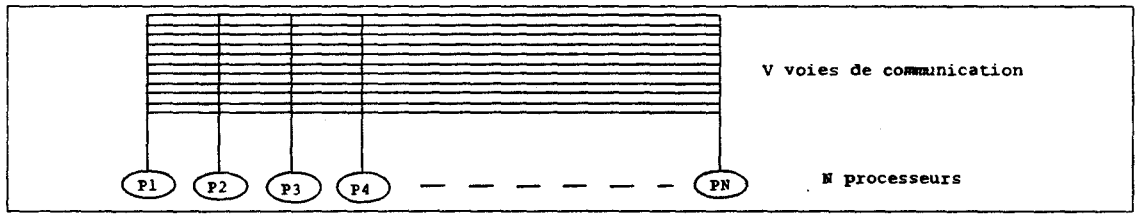

Figure 2.12 : Vue simplifiée de l'ARP

Le projet ARP définit une architecture qui globalement peut être vue comme un ensemble de N processeurs de puissance P Mips (Millions d'instructions par seconde) se partageant V voies de communication de débit D_{res} Mbits/s (Méga bits par seconde).

Il est intéressant, dans un premier temps, d'étudier la performance d'une telle architecture, et de déterminer un ordre de grandeur des paramètres N, P, V et D_{res}. La détermination du paramètre D_{res} nous informe sur le mode de transmission à utiliser, par conséquent, sur les aspects physiques à considérer. Le partage

des voies de communication introduit des phénomènes de conflits d'accès aux processeurs et aux voies disponibles. Une section est consacrée aux conséquences causées par ceux-ci.

2.3.2.1 Performances et contraintes du réseau communication

Pour déterminer la capacité du réseau à supporter la charge en communication des processeurs, nous pouvons appliquer le principe de globalité [Fra92] : on compare la quantité d'informations susceptibles de circuler dans tout le réseau par unité de temps à la quantité d'informations totale devant circuler pour effectuer la communication. On parle de la *la bande passante* de tout le réseau comparée à la *bande passante* nécessaire à la communication.

Nous définissions le **grain de parallélisme** par le rapport entre L et I. I est le nombre d'instructions exécutées localement avant qu'un processeur n'entre en communication, et L la longueur moyenne (exprimée en bits) des messages échangés. Nous considérons uniquement l'aspect purement matériel, les messages échangés entre les processus résidants sur le même processeur ne sont pas comptabilisés, puisque dans ce cas la communication est locale. Le débit de communication des N processeurs peut alors s'exprimer de la façon suivante :

$$\boxed{B_p = \frac{N \times L \times P}{I}} \tag{2.1}$$

Et la bande passante du réseau est :

$$\boxed{B_r = V \times D_{res}} \tag{2.2}$$

Pour que le réseau de communication soit convenablement dimensionné, il faut que le débit en communication soit inférieur à la bande passante du réseau. Ce qui donne l'inégalité suivante :

$$\boxed{B_p \leq B_r \Longleftrightarrow \frac{N \times L \times P}{I} \leq V \times D_{res}} \tag{2.3}$$

De cette inégalité, nous en déduisons une expression donnant la bande passante D_{res} nécessaire, en fonction des autres paramètres :

$$\boxed{D_{res} \geq \frac{N}{V} \times \frac{L}{I} \times P} \tag{2.4}$$

Les paramètres N et V sont fixés par des contraintes technologiques. La valeur de P dépend du processeur utilisé. Le respect de l'inégalité dépend donc du choix des valeurs de L et I, et par conséquent du choix du grain de parallélisme.

Le rêve des architectes est de concevoir un réseau de communication dans lequel le processeur pourra atteindre en un temps identique aux données locales et distantes. L'exécution d'une instruction nécessite en général 2 opérandes, le réseau idéal devrait donc être capable de fournir au processeur 2 données à chaque cycle de calcul. Parmi les machines à mémoire distribuée actuelles, aucune d'elles ne dispose d'un réseau de communication capable d'un tel débit.

Le projet PTAH vise à concevoir une architecture réalisant cet objectif [CB91], mais les applications visées sont du type calcul numérique et déterministe dont on connaît parfaitement le graphe des processus. Le réseau de PTAH est reconfigurable synchrone et une compilation est nécessaire pour définir toutes les configurations du réseau de PTAH au cours de l'exécution. La topologie de ce réseau se modifie en suivant ces configurations à des moments précis déterminés à la compilation.

La réalisation de 2 communications pour un calcul par processeur, dans une machine à mémoire distribuée à vocation généraliste, est difficile à atteindre, étant donnée la technologie actuelle : les processeurs possèdent actuellement une puissance de calcul dépassant les 100 Mips (T9000, Alpha) sur des données de 64 bits. Pour atteindre cet objectif, un canal de communication doit avoir une bande passante d'au moins 12,8 Gbits ! Un tel débit n'est pas possible actuellement en transmission série, par contre il peut être obtenu en transmission parallèle (par exemple un canal à 64 lignes de 200 Mbits/s), mais, dans le contexte des machines parallèles, cette solution n'est pas envisageable économiquement.

La solution, permettant d'atténuer les effets de cette limitation, constitue la base même du modèle de fonctionnement des architectures à mémoire distribuée et locale : le code est chargé dans la mémoire locale du processeur et le réseau n'est utilisé que pour les échanges, moins fréquents, des messages de synchronisation ou contenant des données communes. Les valeurs de I et de L dépendent non seulement des applications mais aussi du placement des processus. En théorie, il n'existe pas de valeurs "standards" pour I et L. En pratique, nous disposons pour cela de quelques règles empiriques :

La régle d'Amdahl/Case permet de donner un ordre de grandeur du débit des canaux : un système informatique est équilibré, s'il dispose d'environ 1 Mo de capacité mémoire et d'environ 1 Mbits/s de bande passante d'E/S pour chaque MIPS de performance d'UC. Si nous appliquons cette règle dans le cas du T9000, il faudrait un débit supérieur à 100 Mbits/s pour chaque canal.

La propriété la plus importante, qui est le plus souvent exploitée, est la **localité des références** : les programmes ont tendance à réutiliser les données et les instructions qu'ils ont utilisées récemment. Une constatation largement répandue est qu'un programme passe 90% de son temps d'exécution sur seulement 10% des instructions.

Des progrès, dans le domaine de la densité d'intégration de la mémoire dont la taille double tous les trois ans [HP92], ont permis de réaliser des mémoires plus compactes et de capacité plus importante : la taille de la mémoire principale varie actuellement de 16-64 Mo, tandis que celle de la mémoire cache varie de 16-64 Ko (CM5, Paragon). Une conséquence de la propriété de localité [SC91] est que, si la mémoire locale du processeur est suffisamment grande pour stocker le code et les données d'un processus, alors le processeur n'effectuera qu'un petit nombre d'entrées et de sorties vers le réseau au cours de l'exécution.

Le tableau 2.2 donne, à titre indicatif, quelques caractéristiques telles que la puissance de calcul et la bande passante des entrées et sorties des processeurs des machines existantes ou futures [Got92] [Cor91a] [LAD+92] [Qui92] :

Tableau 2.2 : Performances des machines actuelles et futures

	Année	Processeur	Calcul		Communication	
			Mips	Mflops	Latence	Débit
iPSC/1	1985	80286/80287	1	0.05	1.7 ms	10 Mo/s
iPSC/2	1988	80386/80387	4	0.3	370 μs	22 Mo/s
Kyushu RPP	1989	SPARC	10	1.6	150 ns	20 Mo/s
iPSC/860	1990	i860XR	40	60	70 μs	22 Mo/s
KSR1	1992	Sharp	40	40	? μs	32 Mo/s
PARAGON	1992	i860XP	50	75	25 μs	200 Mo/s
CM5	1992	SPARC/C.Vect	128	128	5 μs	20 Mo/s
Parsytec GC5	?	T9000	200	25	10 μs	80 Mo/s

Ce tableau ne fait que confirmer la différence qui existe entre la puissance de calcul et la bande passante du réseau. La majorité des machines citées possèdent un réseau de communication à topologie statique, leur temps de latence est plus élevé que celui des réseaux à topologie dynamique, comme par exemple la machine RPP qui utilise un crossbar. Mais, les réseaux à topologie statique autorisent un débit plus important, car, dans ces réseaux, chaque processeur ne dispose que d'un faible nombre de connexions courtes, puisqu'ils ne communiquent directement qu'avec des processeurs situés physiquement dans son voisinage. De plus, ces

connexions courtes, en faible nombre, permettent une transmission en parallèle des données, malgré une fréquence d'émission plus faible dûe aux limitations liées aux transmissions sur bus commun.

Comme il n'est pas possible de déterminer la valeur de L et de I dans le cas général, nous ne nous intéresserons qu'aux applications compute-bound, c'est à dire aux applications où le rapport $\frac{L}{I}$ est strictement inférieur à un.

Pour des raisons d'économie de broches, nous nous limitons, dans un premier temps, à des liaisons séries bidirectionnelles. Les performances des liaisons séries ont beaucoup progressé. Elles permettent d'atteindre, aujourd'hui, des débits de l'ordre du Gbits/s. Des circuits pour la transmission série sont commercialisés ou annoncés (Taxi 150 Mb/s, Hot Rod 800 Mb/s, Autobahn 1.8 Gb/s) [Lit92].

Une étude et une mise en oeuvre d'une liaison série ont été réalisées dans le cadre du projet M3S [Sai91]. L'auteur a réalisé une liaison série à 800 Mb/s avec des circuits en AsGa du commerce. La solution M3S peut s'appliquer, dans notre cas, à condition de résoudre le problème de la transmission de l'horloge d'échantillonnage (car il n'est pas possible, dans notre cas, de disposer d'une horloge globale. Il est donc nécessaire de transmettre l'horloge en même temps que les données) et de fournir une puissance en sortie capable d'alimenter $N-1$ entrées (cela se fera nécessairement par des amplificateurs au détriment de la performance).

2.3.2.2 Problèmes de conflits d'accès au réseau de communication

L'approche que nous avons adoptée fait apparaître 2 sources de conflit: tout d'abord, plusieurs requêtes peuvent être faites simultanément à un même processeur , mais de plus, le nombre de requêtes à satisfaire simultanément peut être supérieur, à un moment donné, au nombre de liens.

On rencontre des problèmes similaires dans des systèmes à mémoire commune disposés en plusieurs bancs mémoires où les processeurs accèdent aux bancs mémoire par un nombre limité de bus [MHW87]. Ces architectures multi-bus sont caractérisées par 3 paramètres : N (nombre de processeurs), M (nombre de bancs mémoire), B (nombre de bus). Dans la plupart de ces architectures, nous avons $B < M$, car cela permet de réduire le coût matériel, mais surtout, des études ont montré que, à cause de ces deux types de conflits, les bus ne sont jamais totalement occupés [LVA82] [LVF83] [DB85] [CS91]. Dans les 4 études, le pourcentage de bus occupés ne dépasse pas 0.63%.

Dans les architectures à mémoire distribuée, si nous modélisons les requêtes faites par les processeurs comme une séquence de suites de Bernouilli indépendantes,

et si N est le nombre de processeurs, le but sera de déterminer une expression donnant le nombre moyen de requêtes de communication des processeurs et la bande passante du réseau.

Soit r le *taux des requêtes*, c'est à dire la probabilité qu'un processeur arbitraire P_i émette une requête de communication vers un autre processeur, le nombre de requêtes de communication est donné par Nr. Des requêtes peuvent être bloquées à cause des conflits vers le même processeur et ceci quel que soit le nombre de canaux disponibles. Le réseau de communication composé de C canaux permet de fournir une bande passante BW. La présence des conflits signifie que toutes les requêtes Nr ne vont pas être satisfaites, ce qui implique $BW < Nr$.

La probabilité d'une requête issue de P_i vers un processeur particulier P_j est $\frac{r}{N-1}$ indépendamment de i ou de j; la probabilité pour qu'il n'y ait aucune requête de P_i vers P_j est $1 - \frac{r}{N-1}$. Donc, la probabilité pour qu'il n'y ait aucun processeur émettant des requêtes vers un processeur P_j est $(1-\frac{r}{N-1})^{N-1}$. Soit E_j l'évènement qui signale la présence d'une requête vers P_j, la probabilité pour qu'il y ait au moins une requête vers P_j est notée par :

$$\boxed{\Pr[E_j] = 1 - (1 - \tfrac{r}{N-1})^{N-1}} \tag{2.5}$$

pour tout j. Si les évènements E_j, j = 1, ..., N, sont supposés indépendants, s'il existe suffisamment de canaux pour que chaque couple de processeurs puisse toujours disposer d'un canal de communication et si chaque processeur peut émettre et recevoir en même temps, alors le nombre moyen de communications peut être exprimé par :

$$\boxed{BW = \textstyle\sum_{j=1}^{N} \Pr[E_j] = N[1 - (1 - \tfrac{r}{N-1})^{N-1}]} \tag{2.6}$$

Dans le cas où N est suffisamment grand, une borne inférieure de la bande passante BW est donnée par :

$$\boxed{BW \approx N(1 - e^{-r})} \tag{2.7}$$

Le tableau 2.3 donne le pourcentage d'occupation du réseau de communication en fonction du taux des requêtes de demande de communication r. La conclusion, que l'on peut tirer à partir de ces résultats, est que le problème de conflit d'accès au même processeur constitue la principale source de limitation de la bande passante du réseau. La deuxième source de conflits (limitation du nombre

de canaux) n'introduit aucune conséquence ici, étant donné qu'il existe autant de canaux que de processeurs.

Dans le cas où le processeur effectue très fréquemment des requêtes de demande de communication, à chaque cycle, par exemple ($r = 1$), la charge maximale du réseau de communication ne dépasse pas 63% ; ce pourcentage varie avec le taux des requêtes. La troisième ligne du tableau, qui est le rapport entre la première et la deuxième ligne, donne le pourcentage des requêtes satisfaites. Nous voyons qu'elle approche les 90% pour des fréquences peu élevées.

Tableau 2.3 : Le pourcentage d'occupation du réseau de communication $\frac{BW}{N}$ en fonction du taux des requêtes r, et le pourcentage des requêtes satisfaites

r	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{BW}{N}$	0.09	0.18	0.26	0.33	0.39	0.45	0.5	0.55	0.59	0.63
$\frac{\frac{BW}{N}}{r}$	0.9	0.9	0.87	0.82	0.78	0.75	0.71	0.69	0.66	0.63

Dans le cadre du projet ARP, le domaine visé d'applications est du type compute-bound, c'est à dire que le taux des requêtes est peu élevé. La conséquence immédiate est que, avec un nombre de voies V inférieur au nombre de processeurs, le réseau crossbar d'ARP est capable théoriquement de satisfaire la majorité des requêtes en communication des processeurs.

2.3.2.3 Le partage des voies de communication

Nous venons de voir que le taux des requêtes détermine le pourcentage d'occupation du réseau, ainsi que le pourcentage de succès des requêtes. Ce taux peut être défini par $\frac{L}{I}$ (avec $L < I$), qui exprime la longueur du message demandé par instruction exécutée. Et son inverse donne le nombre d'instructions exécutées avant que le processeur n'entre en communication. Si nous réécrivons l'équation 2.4 sous la forme :

$$\boxed{V \leq N \times \frac{P}{D_{res}} \times \frac{L}{I}} \tag{2.8}$$

nous obtenons alors une borne inférieure pour le nombre V de voies en fonction des autres paramètres. Si le processeur choisi est le T9000 (P=200Mips) et le débit $D_{res} = 200$ Mb/s. alors le tableau 2.4 donne les différentes valeurs de V en fonction de N et $\frac{L}{I}$. La valeur en gras est obtenue par l'équation 2.8, et l'autre valeur de V est obtenue par l'équation 2.7 en posant $V = BW$ et $\frac{L}{I} = r$.

Tableau 2.4 : Le nombre maximal de voies utilisées, en tenant compte des conflits (sans conflits), en fonction du nombre de processeurs et du taux des requêtes

		N			
		128	256	512	1024
$\frac{L}{I}$	0.01	1 (**1**)	2 (**2**)	5 (**5**)	10 (**10**)
	0.05	6 (**6**)	13 (**12**)	26 (**24**)	52 (**48**)
	0.1	13 (**12**)	26 (**23**)	51 (**46**)	102 (**92**)
	0.25	32 (**28**)	64 (**56**)	128 (**112**)	256 (**224**)
	0.5	64 (**50**)	128 (**100**)	256 (**200**)	512 (**400**)
	1.0	128 (**81**)	256 (**161**)	512 (**322**)	1024 (**644**)

La partie supérieure du tableau nous intéresse plus particulièrement ; pour un rapport $\frac{L}{I} \leq 0.1$, le nombre de voies nécessaires et le nombre de voies effectivement occupées sont proches ou égaux, c'est à dire que la majorité ou totalité des requêtes sont satisfaites. Dans ce cas, le phénomène de conflits n'introduit que peu de limitation.

D'un point de vue théorique, il est donc possible de partager une voie de communication par plusieurs processeurs. Les valeurs données par ce tableau ne constituent que des bornes théoriques. En pratique, il est nécessaire de prendre en compte le temps d'établissement des communications. Ce paramètre peut s'avérer prépondérant pour des messages courts. L'un des objectifs du projet est de définir un protocole qui minimise ce temps d'établissement, afin d'exploiter au mieux la bande passante de chaque voie.

Chapitre 3

Présentation du projet ARP

2 février 1993

3.1 Introduction

Le projet ARP définit une architecture d'un réseau d'interconnexion pour un multicalculateur à mémoire distribuée et privée, où les processeurs communiquent par échange de messages sur un lien physique direct. Le réseau de communication est à reconfiguration dynamique et asynchrone, c'est à dire que l'établissement et la destruction des liaisons ne sont faits qu'à la demande explicite des processeurs concernés.

Le réseau de communication est du type crossbar CCM à V liens bidirectionnels, il permet d'interconnecter jusqu'à N processeurs. Nous venons de voir qu'il n'est pas nécessaire de disposer autant de liens que de processeurs, si les applications traitées sont du type compute-bound. Dans ce cas, il est nécessaire de partager l'ensemble des liens de communication par les processeurs.

La gestion du réseau comprenant des tâches comme l'établissement et la destruction des liaisons, la résolution des conflits entre les demandes de communication vers le même processeur, et l'attribution des liens de communication, est à la charge d'une unité spécialisée : le gestionnaire de communications. Il permet,

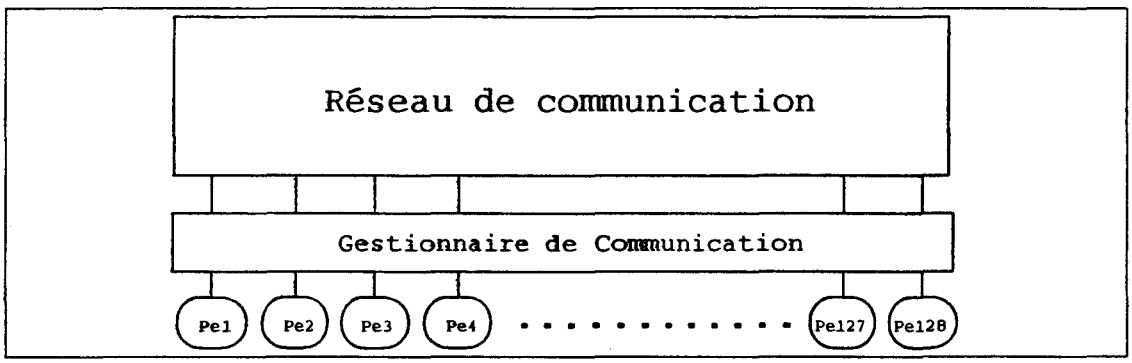

Figure 3.1 : Organisation d'ARP

d'une part, d'alléger la charge des processeurs et, d'autre part, de rendre la conception du réseau indépendant des processeurs.

Nous allons, dans ce qui suit, introduire les objectifs et les contraintes du projet ARP.

3.2 Objectifs et contraintes

- Le projet consiste dans un premier temps à définir une machine interconnectant quelques centaines de processeurs permettant d'atteindre une performance de l'ordre du Gigaflop/seconde. Ensuite, nous proposerons des solutions qui utilisent l'architecture de base pour obtenir un machine de taille supérieure.

- La création et la destruction d'une liaison bipoint entre deux processeurs se fait de façon dynamique asynchrone. La modification d'une liaison ne doit pas affecter les communications en cours entre les processeurs. Nous verrons, dans ce qui suit, que cet objectif sera atteint par un arbitrage distribué.

- Le réseau de type crossbar CCM $N \times M$ utilisé peut être réalisé par assemblage de crossbars CCM $K \times M$, avec $K < N$. Cette propriété autorise l'extensibilité du système, et pour faciliter cette extension la commande du réseau doit être distribuée.

- Le nombre de liens étant inférieur au nombre de processeurs, un lien doit être partagé par un sous-ensemble de processeurs, leur nombre dépend de la fréquence des communications, de leur durée et de leur temps d'établissement.

Il est nécessaire de définir un protocole de communication efficace, qui minimise le temps d'établissement des communications, afin d'augmenter la fréquence ou la durée des communications ou alors le nombre de processeurs.

- Une machine à mémoire distribuée, de par son organisation, supporte les modèles de programmation MIMD et à parallélisme de données; ce dernier suscitant actuellement beaucoup d'intérêts dans la communauté scientifique. Néanmoins, l'implémentation de ces modèles nécessite des primitives de communication globale et de synchronisation; le réseau doit donc fournir des mécanismes capables d'effectuer ces primitives.

- Comme nous venons de voir dans le §2.1.2, le réseau idéal doit permettre 2 types de reconfigurations : synchrone et asynchrone. La reconfiguration dynamique synchrone permet de réaliser le placement statique et le placement dynamique. La reconfiguration asynchrone permet de modifier partiellement la configuration du réseau; cette opération est parfois nécessaire puisque le placement dynamique est une opération très coûteuse en temps de traitement et de communication.

- Le réseau de communication ne doit pas introduire de hiérarchie. Tous les processeurs se trouvent au même niveau. Le réseau de nature reconfigurable utilisé fait que tous les processeurs sont virtuellement voisins et disposent de la même puissance de communication. Cette propriété permet de facilité le placement.

- Les progrès obtenus dans le domaine de l'intégration autorisent la conception des architectures comportant un grand nombre de processeurs. Cela induit une augmentation des performances, mais malheureusement aussi de la probabilité de panne, car elle est proportionnelle au nombre de circuits de la machine. Des éléments de calculs identiques sont présents en grand nombre dans les architectures actuelles, cette abondance permet à la machine de mieux tolérer des pannes éventuelles de ses éléments. La partie la plus sensible aux pannes d'une machine est son réseau de communication, la largeur de bissection du réseau définie dans §1.1.1 permet d'évaluer sa capacité de tolérance aux pannes. Il est donc primordial que le réseau de communication ait une largeur de bissection importante et soit capable de fonctionner en mode dégradé.

- Le réseau doit être conçu indépendamment des spécificités d'un processeur particulier, cela permet au système d'évoluer de façon indépendante en fonction des progrès techniques réalisés dans chacun des 2 domaines. Cela conduit à la définition d'un gestionnaire de communication prenant en charge

les primitives de création et de destruction des connexions. Par ailleurs, la résistance aux pannes nécessite de réaliser la commande du réseau de façon totalement distribuée.

- La contrainte de faisabilité impose l'intégration dans un même boîtier PGA de l'ensemble des liaisons bipoints ainsi que des gestionnaires de communication des processeurs.

3.3 Les communications globales dans ARP

Il est à remarquer que seul le modèle de communication un vers un est considéré ici. Or, d'autres modèles de communication fréquemment utilisés doivent être également pris en compte, comme les modèles de communication globale.

Des études sur les modèles de communication globale peuvent être trouvées dans [JH89] [Fra90]. Nous rencontrons principalement 4 modèles de communication globale :

- **la diffusion** (one-to-all broadcasting) : un processeur source envoie à tous les autres processeurs le même message.
- **l'échange total** (all-to-all broadcasting) : chaque processeur effectue une diffusion.
- **la distribution** (one-to-all personalized communication) ; un processeur source envoie à chaque processeur un message personnalisé. Notons que **la réduction** qui consiste à rassembler des données en un processeur est la forme inverse de la distribution.
- **la multi-distribution** (all-to-all personalized communication) : chaque processeur effectue une distribution.

Le coût de chacun de ces modèles de communication est borné dans les architectures à topologie statique, et les bornes sont données pour des topologies classiques, comme anneau ou hypercube, dans [JH89] [Fra90] [Fra92] [Try92]. Les fonctions de coût minimal sont au mieux dépendantes du diamètre D du réseau, puisque, par la technique du pipeline, le processeur émetteur commence par envoyer le message vers les processeurs les plus éloingnés, c'est à dire situés à la distance D de lui, puis D-1, ..., jusqu'à 1. Pendant qu'un message progresse vers son destinataire en franchissant successivement des noeuds intermédiaires, ces noeuds laissés par ce message pourront être aussi tôt utilisé par les messages destinés aux processeurs moins éloignés. Les messages sont pipelinés tout au

long du chemin et parviennent à leurs destinataires avec un temps borné par le diamètre D.

L'intérêt d'un réseau de crossbar CCM réside dans la possibilité d'avoir un émetteur et plusieur récepteurs simultanés; il est donc possible d'effectuer la diffusion en une seule étape.

3.4 Organisation générale de la machine ARP

Le crossbar CCM utilisé dans le projet ARP permet une implémentation modulaire, aussi bien du point de vue des liaisons physiques composées d'un ensemble de commutateurs et de lignes de transmission, que du point de vue commande du réseau.

La machine ARP est constituée par un assemblage de plusieurs modules de communication identiques, qui sont reliés par des lignes de communication et de contrôle. Chaque module dispose d'un ensemble de m points d'entrée et de sortie, qui permet de connecter des processeurs ou des interfaces d'E/S (disques).

La mémoire étant distribuée au niveau des processeurs, chaque processeur peut exécuter des tâches indépendamment des autres. La synchronisation et la coopération se font par passage de messages. Les processeurs qui désirent communiquer, doivent au préalable demander une liaison physique. L'établissement et la destruction de cette liaison nécessitent un protocole d'accords entre tous les modules. Une fois la liaison établie, les processeurs connectés disposent d'une liaison physique directe pour communiquer.

Chaque module appelé **MCB** (Module de Communication de Base) est conçu pour être capable de fonctionner soit de manière autonome soit en coopération avec d'autres MCB. Pour cela, toute la partie commande est réalisée de manière distribuée sur chaque MCB. L'absence d'une unité de supervision, d'une part, autorise une extensibilité assez simple du système -le système fonctionne de la même manière avec un ou K modules- et, d'autre part, permet une meilleure tolérance aux fautes.

Le MCB n'a pas été conçu en fonction des spécificités d'E/S d'un processeur particulier. Son interface d'E/S présente un protocole de communication simple, les données sont transmises ou reçues par des liaisons séries. Cette solution présente l'avantage d'une universalité pour l'utilisation des processeurs et par conséquent autorise une hétérogénéité du système, moyennant l'utilisation d'interfaces dédiées.

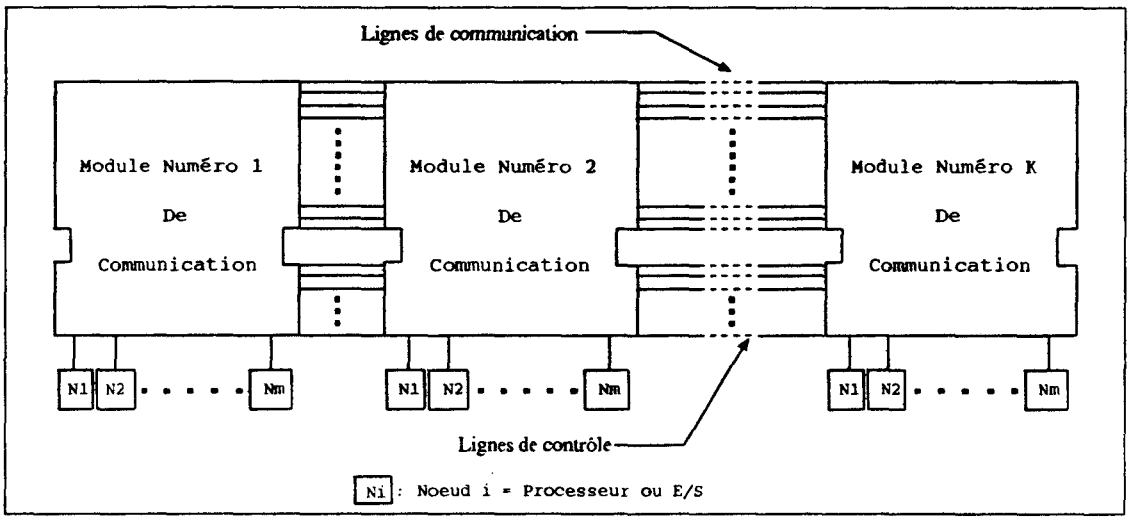

Figure 3.2 : Organisation générale d'ARP

3.5 Module de communication de base (MCB)

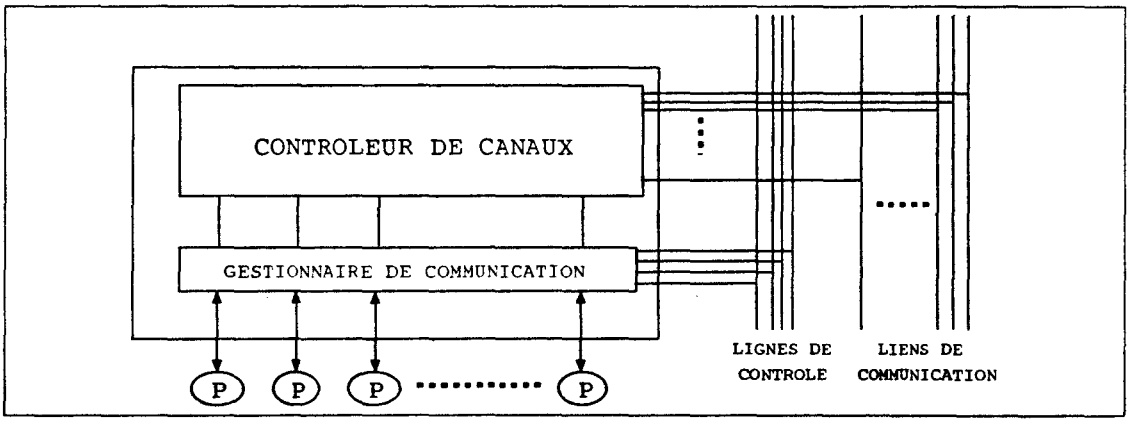

Figure 3.3 : Module de Communication de Base

Le MCB est constitué d'un gestionnaire de communication et d'un contrôleur de canaux. Tous les gestionnaires sont connectés à un bus de contrôle, les gestionnaires peuvent ainsi recevoir (resp. émettre) les informations des (resp. vers les) autres.

L'ensemble des MCB prend en charge la gestion des liens de communication. Ils reçoivent puis traitent les requêtes de communication des processeurs. Ils créent ou détruisent les liaisons en fonction des requêtes des processeurs. Nous avons vu dans le §2.3.2.2 qu'il existe 2 sources de conflits : les conflits d'accès au processeur

destinataire et les conflits d'acquisition des liens de communication disponibles ; par conséquent, les MCB doivent gérer le protocole d'accès à un processeur en 2 temps :

> 1 – Une requête effectuée par un processeur est déclarée **admissible**, si elle ne provoque pas de conflit: c'est à dire si le destinataire est libre.
> 2 – Quand un lien est libéré, une requête admissible est sélectionnée: les processeurs sélectionnés deviennent détenteurs du lien libéré, et peuvent communiquer à travers le lien.

Le rôle des MCB est de créer un réseau virtuel entre la couche physique du réseau et les processeurs, de telle sorte que le réseau est vu par les processeurs comme un réseau complètement connecté.

3.5.1 Le gestionnaire de communication

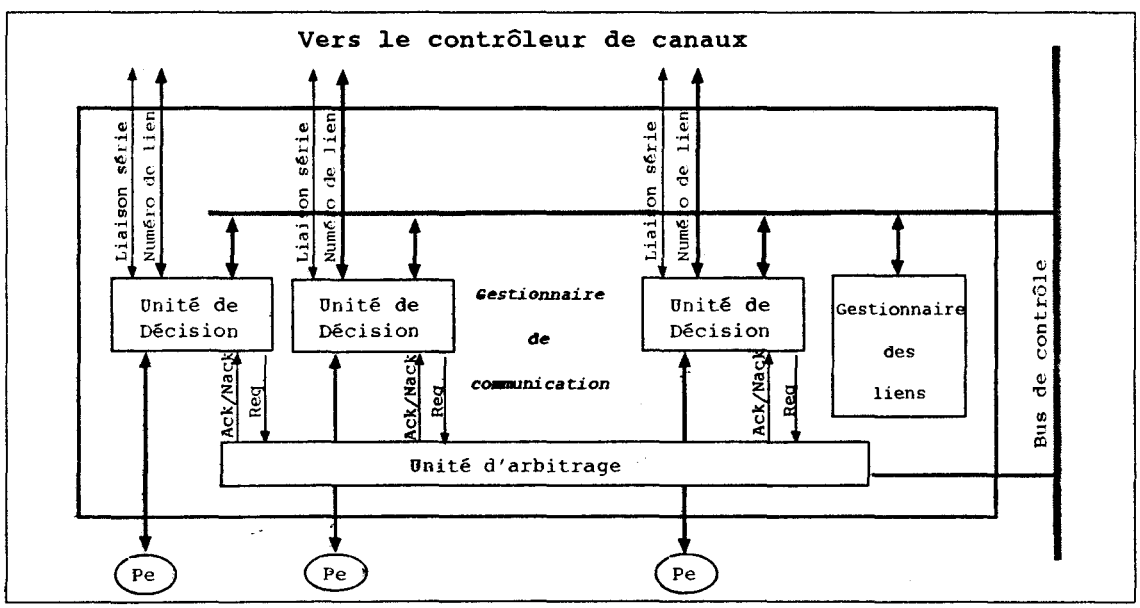

Figure 3.4 : Organisation du gestionnaire de communication

L'un des objectifs du projet est de réaliser la commande du réseau de façon totalement distribuée ; le gestionnaire de communication (*GC*) est constitué par un ensemble d'unités de décision (UD), chacune de ces UD est associée à un processeur. Une UD reçoit deux types de requêtes de son processeur : celles pour

l'acquisition d'un lien de communication et celles pour la libération d'un lien. Chaque UD est chargée de traiter les primitives de communication de son processeur, elle possède l'état du processeur vis à vis des communications (en cours de communication, libre, ..., etc). Chacune de ces informations sera nécessaire pour déterminer si une requête est admissible ou pas.

Dans le cas où un processeur Pe_i désire effectuer une communication vers Pe_j, Pe_i envoie une requête de communication vers UD_i; après la prise en compte de la requête, UD_i informe UD_j de la requête. UD_j refuse la communication, si le Pe_j est déjà en communication, en retournant une réponse négative (*Nack ou No acknowledge*). Elle accepte la communication si Pe_j est libre en retournant une réponse positive (*Ack ou Acknowledge*). Dans ce cas, les unités UD_i et UD_j reçoivent un numéro de lien libre du gestionnaire des liens pour l'envoyer à leur contrôleur respectif, et informent leurs processeurs. La communication commencera lorsque les contrôleurs auront établi la connexion demandée. Le gestionnaire des liens sert d'une part à mémoriser l'état d'occupation de tous les liens du réseau, d'autre part à choisir un lien à utiliser parmi un ensemble de liens libres.

Dans le cas d'une libération, les processeurs en communication envoient une requête de fin de communication vers leur UD. Après la prise en compte de la requête, les UD des demandeurs envoient le numéro du lien à libérer aux contrôleurs de canaux, libérant ainsi physiquement le lien. La libération logique, qui le rend le lien réutilisable, nécessite l'envoie du numéro de lien libéré au gestionnaire de liens. Et une fois le lien libéré logiquement, les processeurs demandeurs sont informés de la libération.

Dans les protocoles décrits ci-dessus, la communication entre les UD n'est pas décrite : le bus de contrôle permet aux UD de transmettre des requêtes et d'en recevoir les réponses. L'accès à l'écriture de ce bus doit donc se faire par exclusion mutuelle suivant un schéma d'arbitrage que nous décrirons par la suite.

3.5.2 Le contrôleur de canaux

Le contrôleur de canaux gère physiquement le réseau de communication. Il reçoit le numéro de lien et la commande (connexion ou déconnexion) à effectuer de l'UD. Ces données lui permettent de positionner les commutateurs pour établir ou détruire les connexions entre les différents processeurs.

Il existe 2 approches pour réaliser le contrôleur de canaux : soit utiliser un contrôleur par module, soit utiliser un contrôleur par processeur.

La première approche (centralisée) utilise un contrôleur unique, qui se charge

de positionner les commutateurs du module, pour cela ; il reçoit en plus des numéros de lien et de la commande, le numéro du processeur demandeur. La communication entre le contrôleur et les UD du module se fait par un bus interne unique, cela permet de réduire le nombre de lignes de commande par le nombre de processeurs dans le module M. Mais, l'utilisation du bus séquentialise l'exécution des commandes.

La deuxième approche (distribuée) utilise un contrôleur par UD ; chaque contrôleur est indépendant ; il reçoit le numéro du lien et la commande, uniquement de son UD, puis réalise la connexion ou la déconnexion. Cette approche nécessite autant de lignes de commande (numéros de lien et de la commande) que de processeurs du module. Mais elle autorise l'exécution en parallèle des commandes, qui est très utile dans le cas d'une diffusion. Cette solution est adoptée dans le projet ARP.

3.6 Le Bus de contrôle

Les unités de décision réalisent les primitives d'accès au réseau (acquisition et libération d'un lien) suivant un protocole défini. Par absence d'unité superviseur, la réalisation du protocole nécessite des échanges d'informations entre toutes les UD qui sont organisées suivant une structure linéaire et sans hiérarchie. Un bus global constitue le moyen de communication le plus simple et le plus efficace pour transmettre ces informations.

3.6.1 Performances des standards de bus

Les deux avantages essentiels de la structure de bus sont le faible coût et la souplesse. En définissant un seul schéma d'interconnexion, on peut ajouter facilement de nouveaux composants. L'inconvénient principal d'un bus est qu'il crée un goulot d'étranglement, limitant le débit maximal. Une des raisons pour lesquelles la conception d'un bus est si difficile est que la vitesse maximale du bus est limitée surtout par des facteurs physiques : la longueur du bus et le nombre de composants (et, en conséquence, la charge du bus). Ces limites physiques empêchent d'accélérer à volonté le bus [Gus84]. Parmi les standards de bus existant actuellement (Bus VME, Mutibus II, Nubus) [Lit92], la bande passante varie de 200 à 400 Moctets/s, sur une largeur de bus de 32-64 bits, avec une fréquence de fonctionnement de 50 Mhz, ce qui fait une bande passante de 50 Mb/s sur une ligne. Quant au standard de bus Futurebus [Bal84] [ES88], le successeur de Mutibus II, il devrait atteindre une bande passante de 8 Goctets/s sur 256 bits

à 250 Mhz. Le nombre de cartes maximal de ces standards est de l'ordre de 20.

L'intérêt du bus est qu'il réalise facilement la diffusion, mais il n'autorise pas d'émissions de requêtes simultanées. Tous les UD sont à l'écoute de ce bus (bus de contrôle), et un *droit unique d'accès* au bus, qui circule entre les UD suivant un schéma d'arbitrage donné, permet à une seule UD d'émettre sur ce bus de contrôle (BC).

3.6.2 Techniques d'Arbitrage des bus

Les unités attachées à un bus sont de 2 types: maîtres et esclaves. Une unité maître peut écrire ou lire des informations sur le bus, une unité esclave ne peut que les lire. Dans un système multiprocesseurs, plusieurs maîtres peuvent tenter de prendre la ressource bus simultanément, ce qui conduit à un état de contention. Si deux ou plusieurs de ces maîtres obtiennent l'usage du bus, on obtient un cas de collision qui conduit invariablement à une perte de l'information. Pour le bon fonctionnement du bus, un maître doit obtenir la permission d'écrire avant de l'utiliser. Si plusieurs maîtres sont en compétition pour l'accès du bus, une unité d'arbitrage doit permettre de choisir le maître du bus. La sélection du maître peut se faire de trois façons différentes :

- éviter collision et contention par une technique du type passage de jeton. Cette solution n'est pas utilisée en environnement multiprocesseur à bus commun du fait des possibilités de temps morts.
- accepter collision et contention mais éviter la perte de l'information par une technique d'abandon et réessai. Cette technique utilisée dans les réseaux locaux est impropre à l'environnement multiprocesseur du fait de l'écroulement du système au delà d'un certain niveau de charge.
- éviter la collision et résoudre la contention par une technique d'arbitrage des requêtes, solution uniformément adoptée dans les systèmes multiprocesseurs.

3.6.2.1 Arbitrage de base

Ce qui semble être une méthode simple d'arbitrage est d'utiliser des signaux particuliers sur le fond de panier, câblé en étoile :

Un réseau de signaux de requête du bus et un réseau de signaux d'allocation du bus permettent de relier chaque unité à l'arbitre par une connexion privée à deux

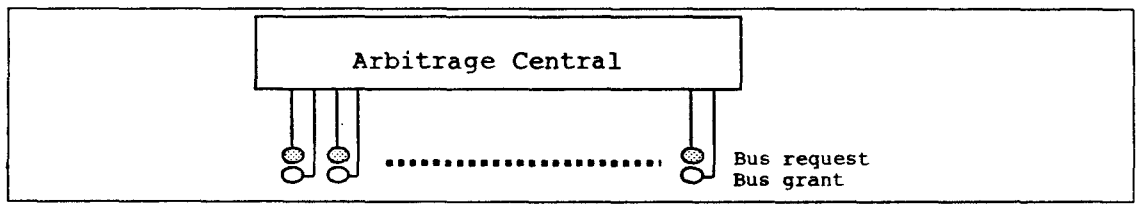

Figure 3.5 : Arbitrage de base

voies. L'arbitre peut alors décider, selon un algorithme quelconque, à qui il attribue le bus. Cette méthode est très souple et permet d'envisager d'implémenter à peu près n'importe quel algorithme, de façon rapide et efficace. Mais cette méthode a aussi de sérieux désavantages :

- solution centralisée.
- le câblage spécial sur le fond de panier est coûteux,
- les informations sur l'arbitrage en cours ne sont pas présentes sur le bus: la surveillance pour diagnostic est difficile,
- si l'arbitre doit être accessible par le bus pour être initialisé ou pour changer la stratégie d'arbitrage, des connexions supplémentaires doivent être prévues.

3.6.2.2 La Daisy chain

Une seconde méthode dite en marguerite ("Daisy chain") utilise un câblage moins onéreux :

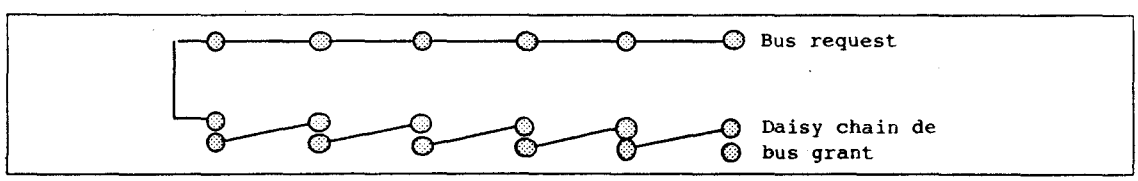

Figure 3.6 : Arbitrage daisy chain

La marguerite utilise une paire de broches de chaque connecteur, câblées de telle façon qu'un signal entre par une broche sur la carte de circuit imprimé et retourne sur le bus par l'autre. On obtient ainsi une connexion série de circuits logiques. Une autre série de broches fonctionnant en OU câblé et relié à une extrémité de la marguerite permet d'obtenir une forme d'arbitrage très commune. Quand une unité désire le bus, elle active le signal de requête sur le OU câblé et attend un

signal en retour sur sa broche d'entrée de marguerite. Chaque unité transmet le signal de la broche d'entrée vers la broche de sortie de la marguerite, tant qu'elle n'a pas elle-même une requête d'accès au bus. On voit que cette forme d'arbitrage définit une priorité fixe, l'unité étant la plus proche de la tête de la marguerite ayant la priorité la plus élevée. Une variante possible est de déplacer dynamiquement la tête du daisy chain permettant d'obtenir une forme de priorité tournante.

Pour empêcher une unité de forte priorité de prendre le bus à une unité de plus faible priorité en milieu de cycle, il faut ajouter d'autres règles de fonctionnement, par exemple synchroniser les requêtes avec d'autres activités du bus.

La daisy chain, très économique, a cependant aussi ses inconvénients :

- elle peut être lente, car les signaux doivent traverser des couches logiques sur les cartes de circuits imprimés,
- elle impose la présence d'une carte ou d'un "pont" à chaque emplacement,
- elle ne fournit pas d'information sur l'arbitrage.

3.6.2.3 Arbitrage Multibus

Un autre schéma d'arbitrage, inventé en 66 par Computing Devices au Canada, redécouvert par Taub d'IBM en 75 [Tau76], amélioré en 84 [Tau84] est utilisé dans divers bus (Fastbus, Futurebus, Multibus II, Nubus). L'idée de base est que chaque unité qui désire le bus tente de placer son numéro de priorité sur le bus d'arbitrage, puis, enlève ses bits les moins significatifs si elle voit un numéro de priorité présent sur le bus plus élevé que le sien. Au bout d'un certain temps seul demeure le nombre de plus forte priorité:

Sur l'exemple, chaque module possède un numéro unique d'arbitrage an6..an0. Le bus possède autant de lignes d'arbitrage et les modules présentent leur numéro via une logique à collecteur ouvert: les lignes véhiculent le OU table des numéros et le niveau électrique zéro représente l'état logique 1. Si un module présente sur une ligne donnée un état logique 0 et détecte que la ligne est à l'état logique 1 ceci signifie qu'un autre module possède un numéro plus grand que le sien et il cesse la présentation des bits de rang inférieur.

Par exemple, si an6=1 et AN6=1 alors on continue l'exploration. an5=0 et AN5=1 signifie qu'un autre module a placé an5=1. L'exploration s'arrête: le OU logique bloque les requêtes suivantes et la sortie "gagnant" est à l'état faux.

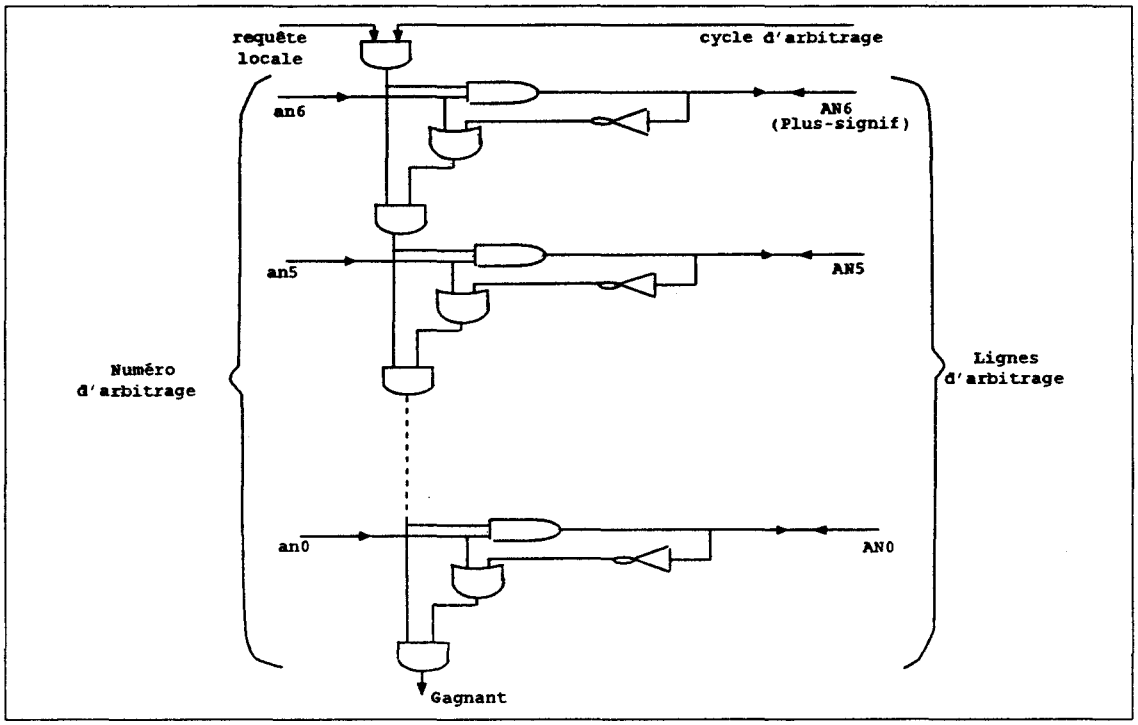

Figure 3.7 : Arbitrage multibus

En ajoutant une règle d'équité du type "pas de nouvelle requête tant que toutes celles en cours ne sont pas satisfaites", on empêche une unité de forte priorité d'accaparer le bus. Dans le Futurebus, le fort poids de l'adresse crée deux classes de requêtes: celles à priorité absolue et celles à priorité équitable respectant la règle ci-dessus.

Le protocole de l'arbitrage Multibus a toujours été présenté comme un protocole équitable [Tau76] [Gus84], mais de récentes études en modélisation et en simulation démontrent que, sous des hypothèses systèmes importantes, ce protocole est inéquitable [VM88].

3.6.2.4 Structure d'Arbitrage ARP

Du faite du caractère modulable du réseau ARP, notre réseau d'arbitrage doit présenter cette caractéristique. Cela conduit à adopter une structure d'arbitrage linéaire du type Daisy chain. Mais cet arbitrage présente l'inconvénient d'être lent. Elle devient vite inutilisable, dès que le nombre de processeurs à arbitrer devient important.

Néanmoins nous conservons la structure linéaire, parce qu'elle est économique et

modulable. Mais avant de présenter notre schéma d'arbitrage, nous allons nous intéresser aux problémes liés au traitement des requêtes de communication sur un bus global.

3.7 Le traitement des requêtes de communication

L'une des principales difficultés du projet ARP est la conception du mécanisme de traitement des requêtes. Ce mécanisme de traitement doit être suffisamment efficace pour exploiter au mieux le réseau de crossbar CCM. Le traitement des requêtes doit se faire de façon totalement distribuée. Il est effectué en établissant un protocole de communication entre les UD. Un bus de contrôle (**BC**) est dédié pour supporter la charge en communication introduite par l'établissement du protocole.

Un protocole intuitif, qui sera affiné par la suite, est décrit à la page 73. Nous remarquons d'après ce protocole que la nature des informations qui circulent sur le bus est très différente : le numéro du processeur destinataire, le numéro du lien, la nature de la demande (acquisition, libération), et la réponse du destinataire.

Quelle que soit la nature de la requête, une acquisition ou une libération nécessite un échange d'au moins 3 informations (requête, réponse et numéro de lien) sur le bus et un temps d'acquisition du droit d'accès. Le traitement des requêtes est effectué sur un bus, ce qui entraîne un traitement sériel des requêtes, il est donc primordial d'optimiser le traitement, optimisation qui se fera en 2 parties : premièrement, réduire le nombre de messages et deuxièmement, minimiser le temps d'acquisition du droit d'accès. Nous désignerons par la suite **le temps de contrôle** comme étant la durée nécessaire au traitement d'une requête.

La totalité des messages nécessaires aux UD pour accomplir leur requête constitue la bande passante nécessaire du bus de contrôle. On parlera de **saturation en contrôle** lorsque la bande passante des messages est supérieure à la bande passante du bus de contrôle.

3.7.1 Réduction du temps de contrôle

La condition nécessaire pour que le réseau ne sature pas, s'exprime par la relation suivante:

$$\boxed{N \times \frac{P}{I} \times (N_m + T \times D_{bus}) \leq D_{bus}} \tag{3.1}$$

L'UD possède le droit d'accéder au BC
- Si requête=libération ou ∃ au moins un lien libre alors - Envoyer sur le BC le numéro de destinataire et la nature de la requête (acquisition ou libération) : **message 1** - Attendre la réponse sur le BC - Si Ack alors Envoyer le numéro de lien sur BC : **message 3** Si requête = acquisition alors connexion sinon déconnexion - Fsi Fsi - Transmettre le droit d'accès du bus

L'UD ne possède pas le droit d'accéder au BC
- Scruter le BC - Si le numéro sur le BC = numéro du processeur alors - Si le processeur est libre alors Envoyer Ack sur le BC : **message 2** Lire le numéro de lien sur BC Connexion. - sinon Si requête=libération alors envoyer Ack sur le BC : **message 2** lire le numéro de lien sur BC Déconnection - sinon envoyer Nack - Fsi

Avec:

- **N** : le nombre de processeurs du réseau.
- $\mathbf{N_m}$: le nombre de messages à échanger pour une acquisition et une libération. Nous avons pris en compte le nombre de messages plutôt que sa taille, car le bus de contrôle peut être défini avec une largeur égale au MAX(Log_2N, Log_2V). Dans ce cas, le temps de transmission d'une information quelconque correspond au temps de transmission d'un bit sur une ligne de données.
- **T** : la durée moyenne séparant 2 accès au BC. Cette durée correspond au temps de contrôle, qui comprend : le temps d'arbitrage du droit d'accès et les temps de connexion et déconnexion au BC. Le produit $T \times D_{bus}$ convertit cette durée en nombre de messages à faire

transiter par le BC.
- **P** : la puissance du processeur en instructions par seconde.
- **I** : le nombre moyen d'instructions exécutées entre 2 requêtes.
- $\mathbf{D_{bus}}$: Le débit du bus de contrôle.

Cette inéquation exprime le débit D_{bus} que doit posséder le bus de contrôle pour supporter la bande passante produite par le traitement des requêtes de communication. D_{bus} s'exprime comme étant le produit du nombre de processeurs N, la fréquence des requêtes de communication $\frac{P}{I}$ et le temps de contrôle d'une requête $(N_m + T \times D_{bus})$.

Les valeurs des paramètres (P, D_{bus}) sont fixées, elles dépendent uniquement du matériel utilisé. Par contre, le nombre de processeurs N et le nombre d'instructions exécutées I sont conditionnés par N_m et T, c'est à dire qu'ils dépendent du temps de contrôle. Les valeurs N et I sont des paramètres qui définissent la performance d'une machine parallèle, N définit sa puissance de calcul et I le domaine d'applications pouvant être traités.

Donc, à valeurs de P et D_{bus} données, l'augmentation de la performance nécessite l'optimisation de N_m et T. Pour cela, il faut, d'une part, réduire le nombre de messages N_m en optimisant le protocole de communication, et, d'autre part, diminuer la durée moyenne séparant 2 accès en utilisant un mécanisme d'arbitrage rapide.

L'occupation du bus de contrôle en fonction du temps est représentée par la suite $T_1 T_{t_1} T_2 \ldots T_{t_{n-1}} T_n T_{t_n}$. Où T_{t_i} est le temps de traitement effectif d'une requête de communication, et T_i est le temps qui sépare la fin du traitement de la requête i et le début du traitement de la requête $i+1$. Le temps T_i dépend directement du temps d'arbitrage. Le pourcentage d'utilisation effective O_{bc} du bus de contrôle s'exprime alors :

$$\boxed{O_{bc} = \frac{\sum_{i=1}^{n} T_{t_i}}{\sum_{i=1}^{n} (T_{t_i} + T_i)}} \qquad (3.2)$$

Nous voyons l'influence du temps d'arbitrage, qui détermine la valeur de T_i : plus T_i est grand devant T_{t_i} moins le bus est utilisé, moins il y aura de requêtes traitées. Le premièr objectif est donc de réduire ce temps d'arbitrage pour approcher une utilisation effective du bus de 100%.

3.7.2 Schéma d'arbitrage d'ARP

Dans le cadre du projet ARP, nous avons adopté un schéma d'arbitrage issu de la technique à passage de jeton. L'idée de base est simple : un jeton circule en permanence d'unité en unité, une unité devient maître du bus lorsqu'elle possède le jeton. Quand une unité désire le bus, elle arrête le jeton dès que celui-ci lui arrive, et le libère en fin d'utilisation du bus (figure 3.8.a).

Cette méthode utilise un protocole équitable du type à priorité tournante (*round robin*); il a été démontré que ce type de protocole donne la meilleure performance parmi un ensemble de protocoles usuels (priorité égale, inégale, aléatoire, protocole par file d'attente) [Gui89].

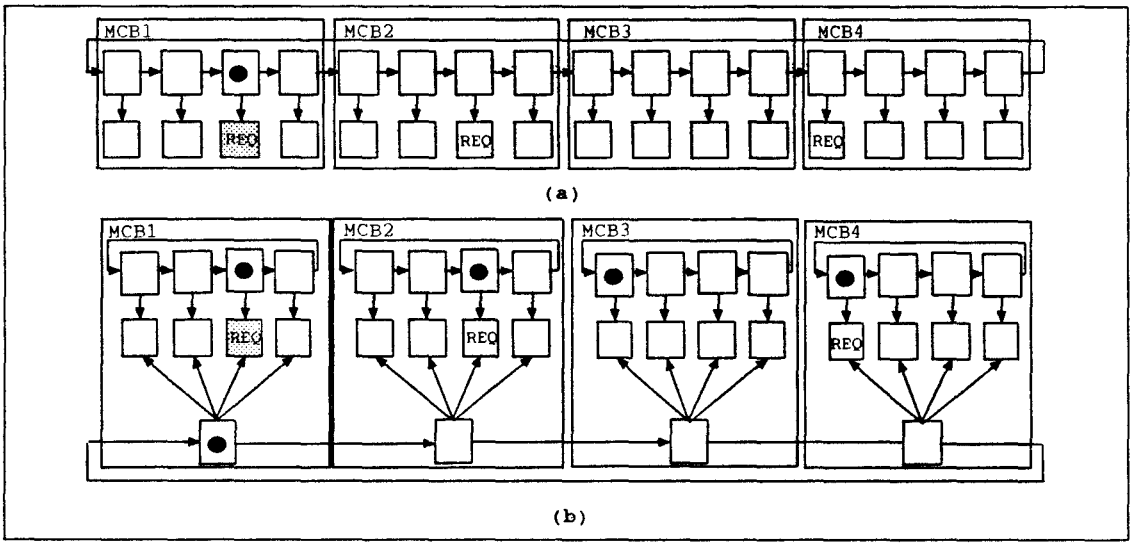

Figure 3.8 : Schéma d'arbitrage à jeton simple (a), sur 2 niveaux (b)

La technique à passage de jetons présente l'intérêt d'être simple à réaliser, nécessite peu de lignes, et est largement et facilement extensible. Mais, à cause de sa structure linéaire, cette technique est pénalisante à cause des possibilités de temps mort. Nous voyons sur la figure 3.9.a que le temps séparant 2 traitements de requête de communication dépend du nombre de places à jeton, qui les sépare. Dans l'exemple 4 places séparent la première requête de la deuxième et 6 places séparent la deuxième de la troisième. Le temps mis par le jeton pour franchir ces places constitue un temps mort, pendant laquelle le BC n'est pas utilisé. Nous proposons ici une amélioration de la technique en minimisant les temps de latence de ce schéma d'arbitrage en apportant plusieurs modifications.

Dans un premier temps, les unités sont séparées en plusieurs groupes. Dans chaque groupe circule un jeton, qu'on appellera **jeton interne**, qui permet de

présélectionner une requête. Un autre **jeton externe** circulant de groupe en groupe permet de valider une des requêtes présélectionnées (figure 3.8.b). L'unité qui désire accéder au bus de contrôle doit dans un premier temps acquérir le jeton interne de son module, puis acquérir le jeton externe. Elle n'accédera au bus que lorsqu'elle possédera les 2 jetons; ces jetons ne seront libérés qu'à la fin du traitement (figure 3.9.b). Cette solution divise par k le temps moyen de latence, avec k le nombre d'unités contenues dans un groupe. Elle s'adapte bien à la structure même de la machine ARP, qui est constituée par un assemblage de modules de communication identiques.

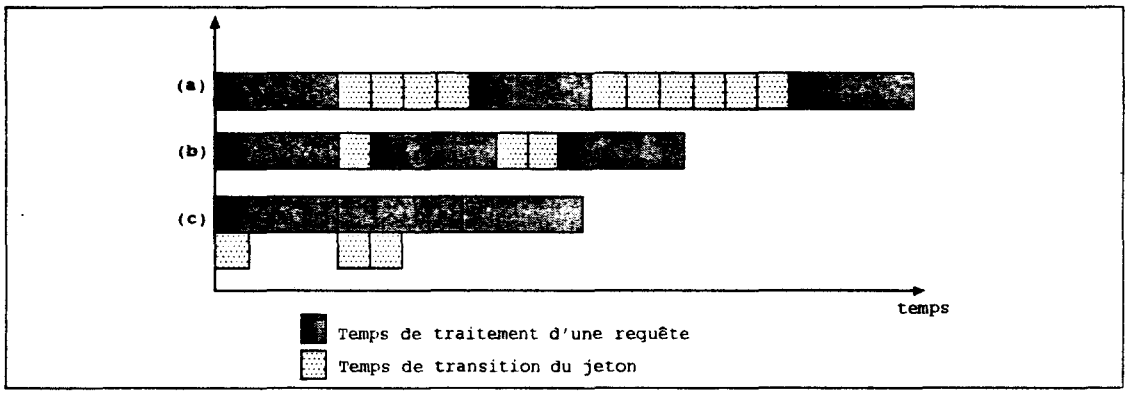

Figure 3.9 : Exemples des temps de latence des différents schémas d'arbitrage à jeton basés sur la figure précédente

Avec l'approche par hiérarchie de jetons, nous limitons le nombre nécessaire d'UD à traverser par le jeton entre chaque traitement de requête en le ramenant au nombre de MCB, mais il subsiste toujours un ou plusieurs temps de circulation dûs au jeton externe. La deuxième modification a pour but de diminuer ce temps en anticipant le passage du jeton externe. Dans le protocole proposé à la page 73, le traitement d'une requête nécessite un temps largement supérieur au temps de circulation du jeton : ce temps de circulation est de l'ordre de la durée de franchissement d'une bascule, alors que le traitement nécessite un échange de messages entre les unités concernées.

La solution adoptée consiste à ajouter sur le bus de contrôle une ligne supplémentaire appelée **bus en cours d'utilisation (BCU)**, dont l'état est vu par toutes les unités connectées au bus. Cette ligne informe les unités sur l'état d'occupation du bus de contrôle, ce qui permet au maître du bus de ne plus conserver les jetons pendant le traitement, et notamment le jeton externe. Cette solution diminue le temps de transition du jeton externe : le jeton externe se déplace à la recherche de la requête suivante à traiter, pendant qu'une requête est en cours de traitement. L'unité possesseur d'un jeton interne dont la requête est sélectionnée par le

jeton externe ne deviendra maître du bus que si l'état de la ligne BCU est à zéro. L'unité devenue maître du bus doit mettre à un la ligne BCU avant d'effectuer son traitement sur le bus, et la remettre à zéro à la fin de son traitement. Sur la (figure 3.9.c), la transition du jeton externe s'effectue en même temps que le traitement, nous obtenons ainsi une utilisation efficace de la bande passante du bus de contrôle.

Le schéma d'arbitrage proposé ici présente les avantages suivants :

- Il ne nécessite que deux lignes d'arbitrage, une pour l'entrée et une pour la sortie du jeton. L'organisation en anneau rend le réseau facilement extensible, et le nombre d'unités connectables n'est pas limité par le réseau d'arbitrage.

- Il permet une utilisation efficace du bus de contrôle, les temps morts sont réduits par la présélection des requêtes par les jetons internes et l'anticipation du passage du jeton externe.

- L'arbitrage est sans famine : au niveau module, le réseau du jeton en anneau réalise un arbitrage équitable, donc sans famine sur les unités, et de même le réseau du jeton externe réalise un arbitrage équitable au niveau des modules. La combinaison des deux réseaux produit un arbitrage sans famine, et "quasi équitable" dans le cas d'une distribution uniforme des requêtes.

3.7.3 Protocoles de communication

Dans la section précédente, nous nous sommes attachés à réduire le temps d'arbitrage, nous allons voir maintenant de quelle manière il est possible de réduire le temps de traitement des requêtes de communication.

Dans le protocole de la page 73, le nombre de messages nécessaires à un traitement de requête est de 3, quelle que soit la nature de la requête : le numéro du processeur destinataire, sa réponse et le numéro de canal. Une solution simple consiste à faire envoyer le numéro de canal non plus par l'expéditeur, mais par le destinataire. Le destinataire retourne ce numéro en même temps que sa réponse à la requête, ce qui réduit le nombre de messages nécessaires à un traitement de requête à 2 et pour une communication, qui comprend une acquisition et une libération, à 4 au lieu de 6.

Dans les protocoles proposés précédemment, les acquisitions et les libérations sont effectuées sur le même bus, pour optimiser le traitement de ces requêtes,

nous avons dédoublé le bus de contrôle et donc l'unité d'arbitrage associée. Une requête de libération et une requête d'acquisition peuvent alors être traitées simultanément.

Les UD disposent de 2 unités d'arbitrage, qui leur permettent d'accéder soit au bus de contrôle d'acquisition (**BCA**), soit au bus de contrôle de libération (**BCL**). Cette solution, qui nécessite 2 réseaux à jeton externe, est un peu plus coûteuse en nombre de lignes. Il faut 4 lignes par réseau à jeton, une ligne pour l'entrée du jeton dans le module, une ligne pour la sortie et 2 lignes de contrôle, ce qui fait en tout 8 lignes pour les 2 réseaux. Mais, elle autorise un traitement en parallèle d'une acquisition et d'une libération, ce qui réduit de moitié la charge du bus de contrôle.

3.7.3.1 Protocole d'acquisition

Le traitement d'une requête d'acquisition nécessite au moins deux messages : requête de l'expéditeur et réponse du destinataire. Le troisième message, correspondant au numéro du lien de communication à prendre, peut être mis sur le bus en même temps que la réponse afin d'économiser un message.

L'approche que nous avons adoptée utilise un gestionnaire des liens présent dans chaque gestionnaire de communication (cf §3.5.1). Le gestionnaire des liens est une unité qui réalise plusieurs fonctions :

- Il mémorise l'état (occupé ou non) de tous les liens de communication dans un vecteur d'état des liens.
- Il fournit en sortie un indicateur d'état (**liens disponibles**) qui signale s'il existe encore des liens libres.
- Si l'état de *liens disponibles* est vrai, alors le registre **numéro de lien** contient le numéro d'un lien choisi parmi l'ensemble des liens disponibles.
- Une ligne **lien pris** permet le passage de l'état libre à l'état occupé du lien dont le numéro est donné par le registre *numéro de lien*.
- Une ligne **lien libéré** permet le passage de l'état occupé à l'état libre du lien dont le numéro est présent sur le bus de contrôle de libération (BCL).

Nous avons dupliqué le gestionnaire de liens dans chaque MCB. A chaque traitement de requête, la même opération de mise à jour est réalisée de façon synchrone dans tous ces gestionnaires; nous disposons donc dans chaque module du même

numéro de lien libre. Les unités n'ont plus besoin de mettre sur le BCA le numéro de lien, elles l'obtiennent par leur bus interne. La commande de mise à jour de tous les vecteurs d'état des liens est produit par le destinataire, lorsqu'il accepte la requête en communication.

Le protocole d'acquisition est décrit ci-dessous : le protocole de communication s'effectue sur le BCA. Les UD, qui désirent /'emettre une requête d'acquisition, doivent avant d'accéder au BCA obtenir le **droit d'accès** au BCA. Une UD obtient le droit d'accès si les 2 conditions suivantes sont remplies :

> 1– si elle possède à la fois le jeton interne et le jeton externe de requête,
> 2– si la ligne d'état BCU associée à l'acquisition, qu'on nommera **BAU** (Bus d'Acquisition Utilisé) par la suite, est à zéro.

Une fois le droit d'accès obtenu, l'UD conserve ce droit en mettant la ligne BAU à un, puis libère les jetons. Elle dépose alors sur le BCA le numéro du processeur destinataire, et lit le numéro de lien libre fourni par le gestionnaire du module. Puis, elle se met en attente de la réponse du destinataire. Si elle obtient une réponse positive (Ack), alors elle restitue le droit d'accès en mettant BAU à zéro, puis se connecte au réseau, sinon elle restitue le droit de la même manière et se remet en attente du droit d'accès. Le nombre de messages nécessaires est de deux dans ce cas. Tantque ∃ aucun lien libre faire attendre

Acquisition	
L'UD obtient le droit d'accéder au BCA	L'UD ne possède pas le droit
– Mettre à un BAU et libérer les jetons – Envoyer sur le BCA le num de pe destinataire (message 1) – Attendre la réponse sur le BCA – Si Ack alors Tantque ∃ aucun lien libre faire attendre Lire le numéro de lien sur bus interne Transmettre le droit d'accés au BCA connexion – sinon transmettre le droit d'accés au BCA	– Scruter le BCA – Si num sur le BCA = num du pe alors – Si le processeur est libre alors Envoyer Ack sur le BCA (message 2) Tantque ∃ aucun lien libre faire attendre Lire le numéro de lien sur bus interne Et mise à jour des vecteurs d'état des liens connexion – sinon envoyer Nack sur BCA – sinon retourner au début

3.7.3.2 Protocole de libération

Le traitement d'une requête de libération nécessite au moins le message contenant le numéro de lien à libérer, afin de mettre à jour tous les vecteurs d'état des

liens. Au cours d'une communication entre deux processeurs, les UD associées ont connaissance du numéro de lien utilisé. Lorsque les processeurs désirent rompre la communication, ils demandent alors tous les deux le droit d'accès au BCL, qui représente :

1- la possession des jetons interne et externe de libération,
2- l'état de la ligne BCU associé à la libération, qu'on nommera **BLU** (Bus de Libération Utilisé) par la suite, à zéro.

Il suffit au premier des deux UD associées qui obtient ce droit, de déposer sur le BCL le numéro de lien à libérer. L'autre UD, étant continuellement à l'écoute du BCL, sera informée de la libération lorsque le numéro de lien présent sur le BCL correspondra à celui qu'elle possède. Lors d'une requête de libération, il n'est donc pas nécessaire de déposer sur le bus le numéro du destinataire. Le nombre de message est réduit à un dans ce protocole de libération.

Libération	
L'UD obtient le droit d'accéder au BCL	L'UD ne possède pas le droit
- Envoyer sur le BCL le num de lien **(message 1)** - Et mise à jour des vecteurs d'état des liens - déconnexion - Transmettre le droit d'accès au BCL	- Scruter le BCL - Si num lien sur BCL = num lien possédé - alors déconnexion

3.8 Faisabilité en nombre de broches

L'un des objectifs du projet ARP est d'intégrer dans un même boîtier un module de communication de base. Chaque MCB est connecté à un réseau de communication, à deux bus de contrôle BCA et BCL, à deux réseaux à jeton, et à un ensemble de processeurs.

Le nombre de broches d'un boîtier est limité, il est intéressant de comptabiliser le nombre de broches nécessaires pour la partie contrôle du MCB, afin d'en déduire le nombre de liens du réseau possible et le nombre de processeurs interconnectables dans un MCB. Les lignes de contrôle nécessaire sont :

- Bus de contrôle d'acquisition : $Log_2 N$ lignes.
- Ligne d'état BAU : 3 lignes (une ligne d'état et 2 lignes de mise à

jour).
- Bus de contrôle de libération : Log_2V lignes.
- Ligne d'état BLU : 3 lignes (une ligne d'état et 2 lignes de mise à jour).
- Ligne Ack/Nack : 1 ligne.
- Ligne d'état SRA : 3 lignes (une ligne d'état et 2 lignes de mise à jour).
- Réseau de jeton de requête externe : 4 lignes.
- Réseau de jeton de voie externe : 4 lignes.

soit au total $Log_2N + Log_2V + 18$ lignes. Pour un boîtier PGA de 144 broches, nous réservons 64 broches pour la connexion au réseau, si le nombre maximal de processeur est 256, alors nous disposons de 48 broches pour la connexion des processeurs au MCB, les lignes d'initialisation et d'alimentation.

3.9 Conclusion

Nous venons de présenter les grandes lignes du projet ARP, qui définit une architecture à mémoire distribuée et privée, où les processeurs communiquent par un réseau de crossbar CCM. Le réseau de communication est reconfigurable de manière dynamique et asynchrone. L'acquisition et la libération des connexions sont à la charge des unités de décision qui sont regroupées en sous-ensembles dans des modules de communication de base. Les MCB sont conçus de manière à pouvoir étendre facilement le taille du système. Les traitements des 2 types de requêtes (l'acquisition et la libération) nécessitent des échanges de messages entre les unités de décision. Ces messages sont échangés sur 2 bus communs différents selon la nature de la requête en cours de traitement. Une unité d'arbitrage distribuée, qui utilise une optimisation de la technique à passage de jeton pour sélectionner le maître de chaque bus, est implantée au niveau de chaque module.

Après la description fonctionnelle du projet ARP, il est intéressant d'évaluer la performance (essentiellement de son réseau de communication) d'une telle architecture.

Chapitre 4

Evaluation des performances de l'ARP

2 février 1993

La figure 4.1 illustre le spectre des techniques qui sont utilisées pour modéliser les performances des systèmes informatiques [All80]. La technique la plus à gauche, l'utilisation des règles de Thumb, est la plus simple à appliquer, mais produit des résultats peu précis puisqu'elle est basée sur l'observation des lois empiriques. La technique la plus à droite obtient de très bons résultats, mais nécessite des efforts de développement considérables. Entre ces 2 extrêmes, l'évaluation des performances par modélisation mathématique réalise un bon compromis entre la précision des résultats obtenus et le coût de développement. La modélisation devient un outil de plus en plus utilisé pour configurer ou analyser des systèmes informatiques. L'objectif d'un modèle est de permettre l'évaluation quantitative et/ou qualitative d'un système, de nous éclairer sur leur comportement et de vérifier leur bon fonctionnement.

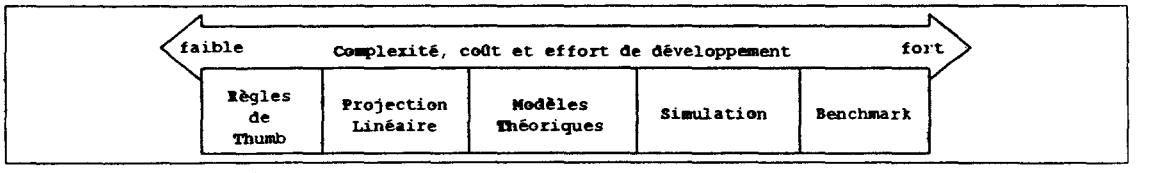

Figure 4.1 : Le spectre des techniques de modélisation des systèmes informatiques

Plusieurs méthodes de modélisation sont aujourd'hui utilisées, chacune d'elles étant plus ou moins bien adaptée à des aspects spécifiques d'analyse des performances.

4.1 Les techniques de modélisation théoriques

Les réseaux de Pétri [Pet66], [Bra83] sont principalement utilisés pour représenter le comportement logique d'un système sans faire intervenir des considérations temporelles. Ils fournissent un modèle graphique de description des opérations d'une machine informatique, et sont particulièrement adaptés pour décrire les phénomènes de concurrence, de conflit et de synchronisation. On obtient à l'aide de ce type de modèle une évaluation qualitative d'un système. Des travaux [Nat80] [Flo87] ont été menés pour inclure le facteur temps dans les réseaux de Pétri et pour permettre ainsi d'effectuer des évaluations de performance en utilisant ces réseaux de Pétri stochastiques ou temporisés.

Le principal inconvénient de ces méthodes provient d'un accroissement considérable du réseau lorsque le système à représenter est important.

Une approche différente d'analyse est donnée par l'utilisation de chaîne de Markov [Dyn62]. Une chaîne de Markov décrit l'évolution d'un système à partir de ses états. Cette méthode analyse le comportement d'un système d'après les spécifications suivantes qui régissent le modèle :

- la quantité d'informations utilisées pour décrire les états de la chaîne,
- le calcul des taux de transition entre états.

Le choix de la définition des états de la chaîne est très important car il intervient dans la complexité de résolution du modèle. Comme pour les réseaux de Pétri, le nombre d'états de la chaîne restreint l'utilisation de cette technique, dans la plupart des cas, à de petits systèmes.

Une autre alternative de modélisation est fournie par les réseaux de files d'attente [FP89]. Cette méthode est certainement la plus utilisée pour estimer les performances d'un système informatique. De ce fait, les recherches dans le domaine de l'évaluation des performances se sont focalisées sur la théorie des files d'attente. De plus, ces méthodes représentent les techniques d'évaluation les plus économiques, car elles débouchent sur des calculs analytiques. Cependant, la classe des réseaux de files d'attente, que l'on sait résoudre exactement, est très réduite et ne permet pas de prendre en considération certains comportements très répandus dans les systèmes informatiques (architectures multiprocesseurs, systèmes répartis, calcul parallèle,...). On peut citer, par exemple, la possession simultanée de plusieurs ressources, le blocage d'un client, la concurrence pour l'accès à des ressources partagées (gestion de sémaphores) ou encore des comportements de dépendance entre les différents éléments ou entre les files du

réseau (synchronisation, génération ou regroupement de clients). Ces comportements engendrent des interactions entre les différents éléments d'un système qui sont difficiles à représenter dans un modèle. Ignorer ces caractéristiques restreint les domaines d'application des réseaux de files d'attente ou conduit à des erreurs d'évaluation des performances.

De nombreux outils de résolution exacte ou approchée ont été publiés dans la littérature. De plus, l'utilisateur dispose de langages spécialisés (QNAP2 [VP84], BEST1 [TBA85]) permettant la description et l'étude quantitative de modèle ayant la structure générale de réseaux de files d'attente. L'intérêt essentiel de ces outils est qu'ils permettent aux non spécialistes d'évaluer les performances de leurs machines sans pour autant s'investir dans l'étude approfondie de la théorie des files d'attente. Un exemple de modélisation du multiprocesseur SM90 à l'aide du logiciel QNAP2 a été présenté dans [Fdi84].

Nous avons donc porté notre choix sur le logiciel QNAP2 pour modéliser et évaluer les performances du réseau de communication ARP. QNAP2 est le résultat du travail d'équipes de recherche de l'INRIA (Institut National de Recherche en Informatique et Automatique) et de Bull qui ont uni leurs efforts, dans les domaines théorique et pratique, pour créer un produit opérationnel.

4.2 Le langage QNAP

QNAP (Queueing Network Analysis Package) [Sim88] est un langage de description et d'analyse de systèmes de files d'attente. Son principal intérêt est d'offrir un moyen descriptif pour caractériser le système à étudier en même temps que des méthodes d'analyse pouvant donner des résultats numériques caractérisant les performances du système.

4.2.1 Description du langage

QNAP traite des objets simples comme des nombres réels ou entiers, des caractères, mais aussi, ce qui fait sa spécificité, des files d'attente.

La modélisation d'un système se fera par sa description sous la forme d'une structure de réseau de files d'attente en décrivant un ensemble de **stations**. Une station se compose des éléments essentiels suivants:

- une file d'attente
- un ou plusieurs serveurs attachés à cette file

- la description algorithmique du service *fourni*
- un algorithme de routage qui permet de définir la circulation des **clients** dans le réseau

Les clients du réseau circulent entre les stations où ils reçoivent un service. Il est possible de caractériser des clients en leur attribuant des classes distinctes.

Lorsque la description du réseau est terminée, QNAP fournit un certain nombre de méthodes de résolution qui sont :

- un simulateur à événements discrets,
- des méthodes analytiques exactes (algorithmes de convolution, Analyse des Valeurs Moyennes MVA, résolution Markovienne),
- des méthodes analytiques approchées (méthodes itératives, approximation par diffusion, approches heuristiques).

Quelle que soit l'une de ces méthodes de résolution, si elle est applicable, elle est utilisable à partir d'une description unique du réseau. Les résultats extraits d'un modèle, après l'exécution de QNAP, sont des données standards du type temps de réponse, taux d'occupation, longueurs de files etc...

Par ailleurs, QNAP fournit trois outils pour rendre compte des mécanismes de synchronisation: les sémaphores, les ressources et les drapeaux, ainsi que les primitives permettant le blocage et le déblocage des files d'attente.

4.2.2 un exemple de modélisation

Nous présentons ici un exemple de modélisation d'un ordinateur : un ordinateur est constitué d'une unité centrale (UC) qui traite les instructions des programmes. Un programme est représenté par un *client*. De temps en temps, le programme effectue une entrée sortie vers un organe périphérique. Les processeurs d'entrées-sorties peuvent être de différents types : un disque, un lecteur de disquette, une bande magnétique, etc...

Dans ce modèle, nous supposerons qu'il y a deux processeurs d'entrées-sorties. Lorsqu'un client sort de l'unité centrale, soit le programme correspondant est terminé et le client sort vers l'extérieur (OUT), soit le programme doit subir une entrée-sortie et il se dirige vers l'un ou l'autre des processeurs d'entrée-sortie (DISCA, DISCB).

On suppose que la source S génère des clients suivant une distribution exponentielle et que les trois files d'attentes du modèle ont également des temps de service exponentiellement distribués avec des disciplines de service FIFO (First In First Out). Les différentes probabilités de routage P_{co}, P_{ca}, P_{cb}, définissent les probabilités, en sortant de l'unité centrale, de se diriger respectivement vers l'extérieur et vers les disques d'entrées-sorties. Le modèle est donné par la figure 4.2.

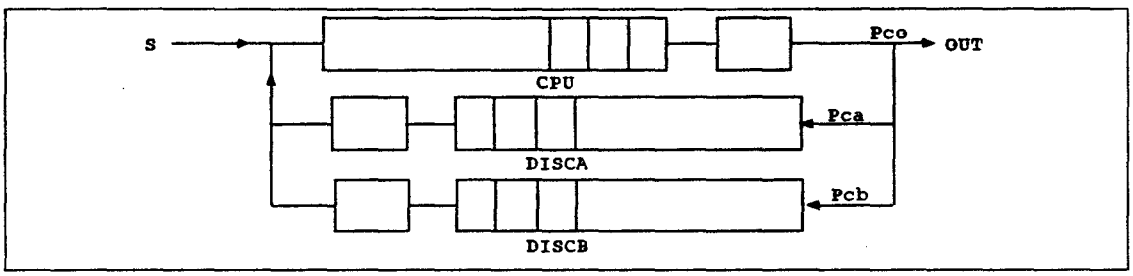

Figure 4.2 : Modèle d'un calculateur

et le programme QNAP correspondant est le suivant :

```
1   /DECLARE/
2   QUEUE S, CPU, DISCA, DISCB;
3   & description de la station source
4   /STATION/
5   NAME = S;
6   TYPE = SOURCE;
7   SERVICE = EXP(100);
8   TRANSIT = CPU;
9   & description de la station unite centrale
10  /STATION/
11  NAME = CPU;
12  SERVICE = EXP(15);
13  TRANSIT = OUT,1,DISCA,4,DISCB,5;
14  & description du serveur disque A
15  /STATION/
16  NAME = DISCA;
17  SERVICE = EXP(15);
18  TRANSIT = CPU;
19  & description du serveur disque B
20  /STATION/
21  NAME = DISCB;
22  SERVICE = EXP(12);
23  TRANSIT = CPU;
24  & RESOLUTION
25  /EXEC/
26  SOLVE("CONVOL");
```

La résolution utilisée ici est la méthode de convolution. Le résultat obtenu avec QNAP est, de façon standard, un tableau contenant une ligne par file présente dans le tableau. Sur chaque ligne figurent les résultats suivants :

NAME	nom de la file d'attente considérée.
SERVICE	La durée moyenne d'un service (résultat d'une mesure dans le cas de la simulation, la valeur fournie dans la description lors d'une résolution analytique).
BUSY PCT	Le taux d'occupation de la file (résultat d'une mesure dans le cas de la simulation, d'un calcul lors d'une résolution analytique).
CUST NB	Le nombre moyen de clients présents dans la file (même remarque que ci-dessus).
RESPONSE	Le temps de réponse moyen (même remarque que ci-dessus).
THRUPUT	Le débit moyen de sortie de la file (résolution analytique).
SERV NB	Le nombre de clients servis par la station (en simulation).

Tableau 4.1 : Résultats de la résolution par la méthode de convolution

NAME	SERVICE	BUSY PCT	CUST NB	RESPONSE	THRUPUT
S	100.0	1.000	1.0	100.0	0.01
CPU	5.0	0.5	1.0	10.0	0.1
DISCA	15.0	0.6	1.5	37.5	0.04
DISCB	12.0	0.6	1.5	30.0	0.05

La difficulté réside essentiellement dans la description du modèle réel sous la forme d'une structure de réseau de files d'attente. Une fois le modèle décrit, la transcription en QNAP ne présente pas de difficultés majeures, vue la simplicité de la syntaxe de ce langage.

C'est la raison pour laquelle nous n'entrerons pas dans le détail de la syntaxe du langage QNAP. Nous proposons ci-dessous un ensemble de symboles graphiques, permettant de décrire les différents objets de QNAP. Cet ensemble de symboles aidera à décrire plus facilement le réseau de communication du projet ARP.

4.2.3 Les symboles graphiques

4.2.3.1 Les objets de base

QNAP comporte également, en plus des objets simples comme des entiers ou caractères, l'objet *station*, l'objet *client* et l'objet *drapeau.*

Un objet **station** peut être défini de plusieurs façons, comme :

- une source, fournissant des clients suivant des lois probabilistes ou non.
- un sémaphore, ou une ressource. Ils permettent entre autre de réaliser des synchronisations.
- une structure comportant une file d'attente avec sa discipline de service (FIFO, FILO, etc...), un ou plusieurs serveurs attachés à cette file, la description algorithmique du service fourni et l'algorithme de routage.

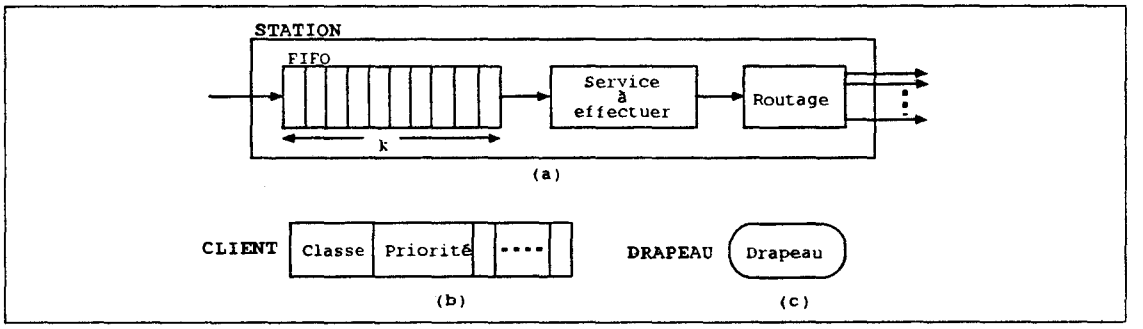

Figure 4.3 : Les objets simples de QNAP

Un objet **client** est l'entité qui circule entre les stations, où il reçoit des services. C'est une structure comportant deux champs prédéfinis : **la classe** et **la priorité**. Le champ classe sert à caractériser les objets clients, afin de pouvoir réaliser des traitements spécifiques selon leur classe. Le champ priorité de type entier est utilisé dans le cas où la discipline de service d'une file d'attente est du type les plus prioritaires d'abord. Des champs supplémentaires peuvent être définis par l'utilisateur.

Un objet **drapeau** sert à réaliser des synchronisations. Il possède deux états : ouvert (Set) ou fermé (Reset). Chaque client a la possibilité de tester l'état d'un drapeau ; si ce drapeau est fermé, le client reste bloqué jusqu'à ce qu'il s'ouvre; si le drapeau est ouvert, aucun blocage n'intervient. De plus, chaque client a la possibilité d'ouvrir ou de fermer un drapeau.

4.2.3.2 Le routage des clients

QNAP fournit des primitives permettant de réaliser un très grand nombre de routages différents : conditionnel, probabiliste, mécanisme de barrière de synchronisation, etc.

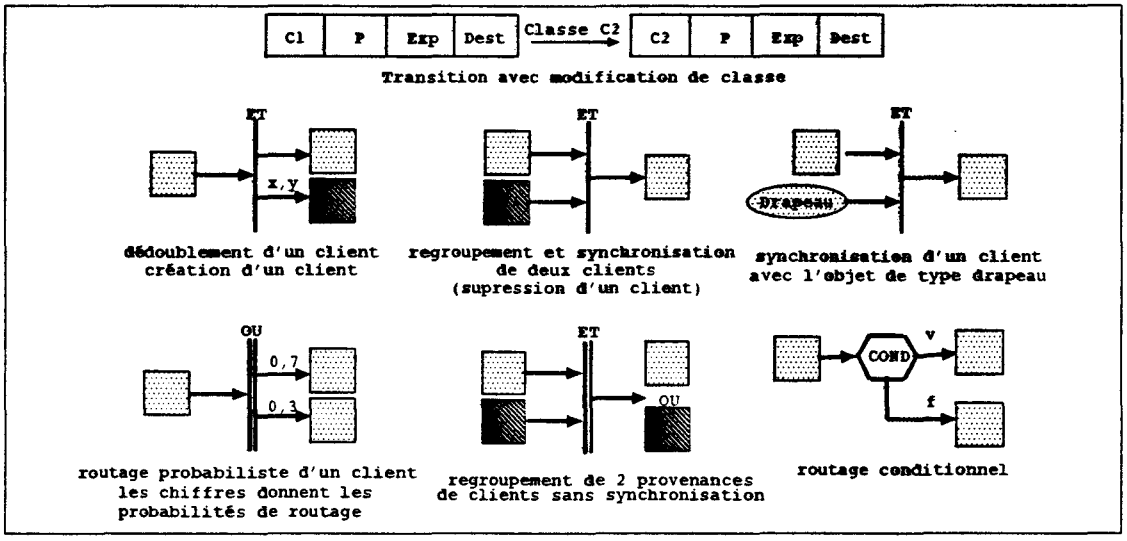

Figure 4.4 : Les routages principaux sur QNAP

4.3 Modélisation du réseau d'ARP

La modélisation du système par Qnap consiste à décrire les différentes unités : processeur, unité de décision, unité de communication, jeton et lien de communication par un ensemble de stations.

4.3.1 Le processeur

Le processeur est vu comme une *source* émettant des requêtes de communication à des instants donnés. Une requête est représentée par un objet client de classe *demande* et deux champs supplémentaires qui indiquent les numéros des processeurs expéditeur et destinataire.

Les instants d'émission des requêtes (ou clients) sont déterminés par une loi aléatoire exponentielle, et l'intervalle moyen entre 2 instants correspond au grain du traitement, c'est à dire au rapport entre le temps de communication et le temps de traitement.

Le choix du processeur destinataire est déterminé par un algorithme, qui sera modifié pour étudier le comportement du système en fonction de la répartition de la charge en communication : communication uniforme, concentration, etc. Le choix se fait dans un premier temps de façon équitable et probabiliste.

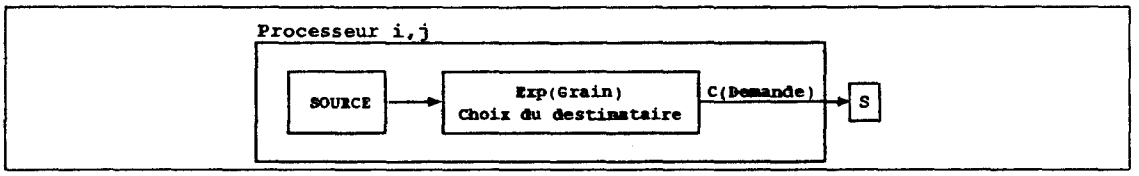

Figure 4.5 : Modélisation du processeur

4.3.2 Le réseau à jeton

Le réseau à jeton est constitué par un ensemble de stations. Les stations forment un anneau dans lequel circule un jeton unique représenté par un client de classe jeton. Une station représente une place du jeton, à laquelle est associé un objet drapeau.

Un client (jeton) entrant dans une station met à un (par la primitive Set) le drapeau associé, reste dans la station pendant un temps égal au temps de transfert du jeton vers la place suivante, puis quitte la station en remettant à zéro (par la primitive Reset) le drapeau. Le jeton peut être bloqué dans une station par une unité de décision, dans ce cas, il ne pourra repartir que si la même unité débloque la station.

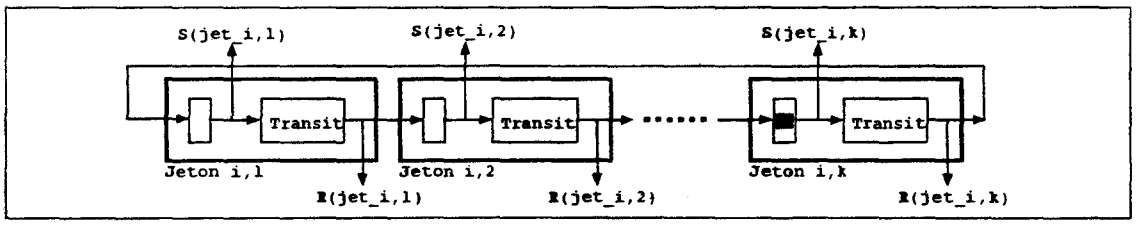

Figure 4.6 : Modélisation du réseau à jeton

4.3.3 L'unité de décision

Une requête (un client) sortant de la station processeur $P_{i,j}$ (i le numéro du module, j le numéro du processeur dans le module) entre dans la station unité de décision $U_{i,j}$ associée. Si une requête est déjà en cours de traitement, la nouvelle

requête sera mise dans la file d'attente qui possède théoriquement une longueur infinie. La gestion de cette file est effectuée suivant la politique du premier arrivé, premier servi (PAPS).

Les requêtes en attente de service sont mémorisées dans l'unité de décision, alors qu'en pratique ces requêtes restent bloquées au niveau du processeur. Ce choix permet, lors de la simulation, d'évaluer facilement les différents paramètres du système : le nombre de requêtes en attente, le temps de service d'une requête, le nombre d'échecs,...etc. La valeur de ces paramètres est donnée par des primitives Qnap.

L'attente du jeton par l'unité de décision est décrite en utilisant une barrière de synchronisation de type ET entre une requête et un objet de type drapeau. Le drapeau à l'état un symbolise la présence du jeton. La requête ne peut franchir la barrière ET que si le drapeau $Jet\_I_{i,j}$ est à l'état un. En franchissant la barrière ET la requête bloque (fonction B) le jeton interne à la place (i,j).

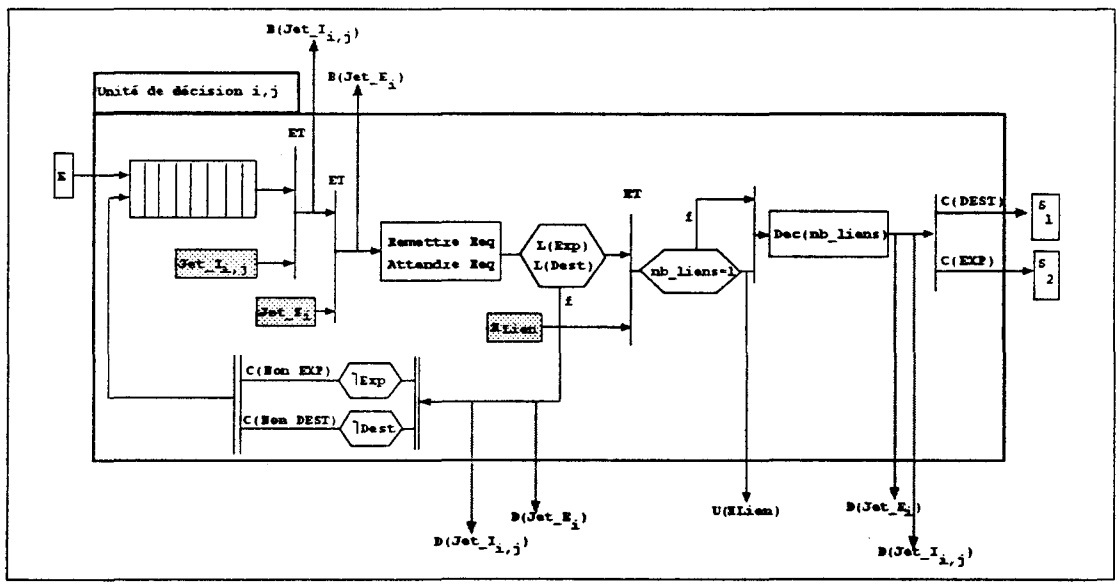

Figure 4.7 : Modélisation de l'unité de décision $U_{i,j}$

L'unité de décision doit posséder les jetons interne et externe au module pour accéder au bus de contrôle. L'accès à ce bus se traduit par l'occupation d'une ressource "bus de contrôle" pendant un temps égal à la durée nécessaire pour effectuer le protocole de communication (décrit précédemment). Il permet à l'UD expéditeur d'informer l'UD destinataire de sa requête et de recevoir sa réponse.

Une requête n'est admissible que si le destinataire n'est pas en cours de communication et si le processeur expéditeur n'a pas été pris en destinataire, entre

temps, par un autre. Si l'une de ces 2 conditions n'est pas remplie la requête libère les jetons en débloquant les 2 stations jetons par la primitive D (débloque) et retourne dans la file d'attente avec une classe différente (Non_exp ou Non_dest) et une priorité supérieure: elle sera prise en compte par l'UD avant celles venant du processeur. Cette approche respecte le modèle réel, dans lequel lorsqu'une requête est refusée, l'UD libère les jetons et réeffectue les différentes opérations qui lui permettront d'accéder de nouveau au bus de contrôle et de réemettre la demande... La requête sera traitée tant qu'elle n'aura pas été satisfaite.

Si la requête est admissible, alors elle passe à l'état en attente de lien (synchronisation ET avec le drapeau 'existe lien'). Elle conserve les jetons tant qu'il n'y a pas de liens disponibles, car il est inutile de passer les jetons, s'il n'y a aucun lien. Si ce drapeau est à un alors la requête peut franchir la barrière , libérer les jetons (fonction D) et passer à l'état en cours de connexion. Si le lien pris est le dernier, le drapeau sera mis à zéro par la primitive U (unset). Pour décrire une communication entre 2 processeurs, l'UD crée un nouveau client de classe 'DEST', qu'elle transmet vers l'unité de communication destinataire et le client courant est transmis vers l'unité de communication expéditeur avec la classe 'EXP'.

4.3.4 Description d'une ressource

L'ensemble du réseau, c'est à dire l'ensemble des voies de communication, constitue une ressource critique que les processeurs partagent. La description d'une ressource par la méthode des files d'attente a été présentée dans [FPM86]. Un objet ressource est défini par une file d'attente de longueur finie K. Un client de la file F ne peut entrer en section critique que si la file ressource est non vide. Un franchissement de la première barrière consomme un client dans chacune des 2 files, et un franchissement de la deuxième barrière ajoute un client dans la file ressource.

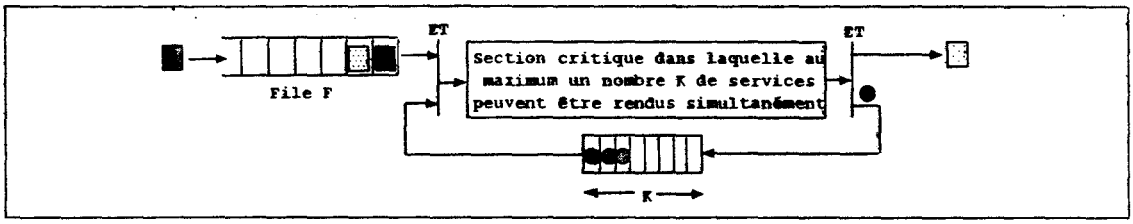

Figure 4.8 : Modélisation d'une ressource

4.3.5 L'unité de communication

L'unité de communication dispose de 2 entrées. L'entrée E1 reçoit les marques représentant les liens de communication disponibles. L'entrée E2 reçoit les requêtes de classe Exp venant de son unité de décision ou les requêtes de classe Dest venant des autres unités. Lorsqu'une demande accède à l'unité de communication, elle positionne le drapeau Com à un, rendant le processeur associé indisponible à d'autres requêtes.

La requête de classe 'Exp' n'accède à la section critique (le réseau de communication) que s'il reste au moins une marque dans la file ressource. Une marque est consommée à chaque accès de la section et sera restituée à chaque sortie par une requête. En fin d'émission une requête positionne le drapeau 'existe lien' à un et le drapeau Com à zéro pour rendre le processeur disponible.

La requête de classe 'Dest' reste dans la station tant que l'expéditeur est en cours de transmission. Elle ne la quittera que si l'expéditeur a terminé de transmettre. Cette contrainte est représentée par une synchronisation avec le drapeau Com de l'expéditeur.

Une requête sortant de la station unité de communication entre dans la file prédéfinie 'OUT', dans laquelle la requête est détruite marquant ainsi la fin du traitement.

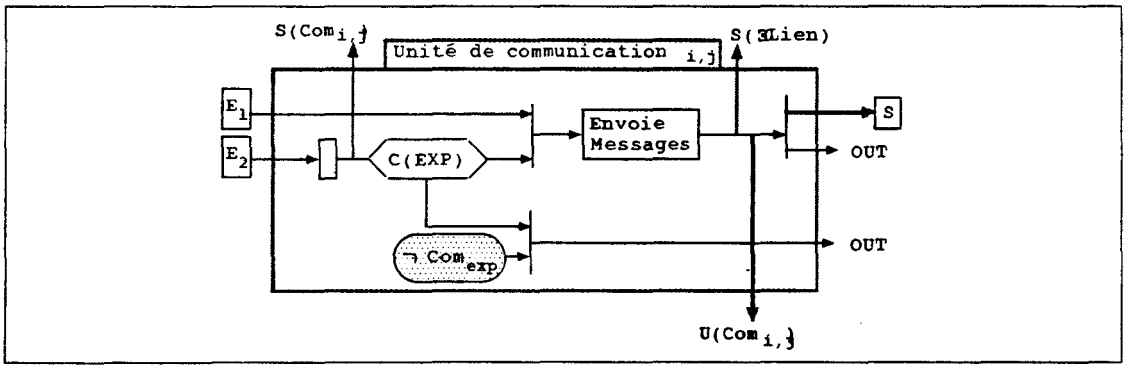

Figure 4.9 : Modélisation d'une unité de communication

4.3.6 Le réseau ARP

Le réseau ARP est constitué par un assemblage de modules de communication de base. Chaque module est lui-même constitué par l'ensemble des stations définies précédemment : les stations processeurs, unités de décision et de communication, et jetons. Pour ne pas surcharger le schéma du MCB, nous n'avons représenté

qu'un seul réseau à jeton. Pour la simulation tous les réseaux de jeton interne et externe pour l'acquisition et pour la libéraion sont utilisés.

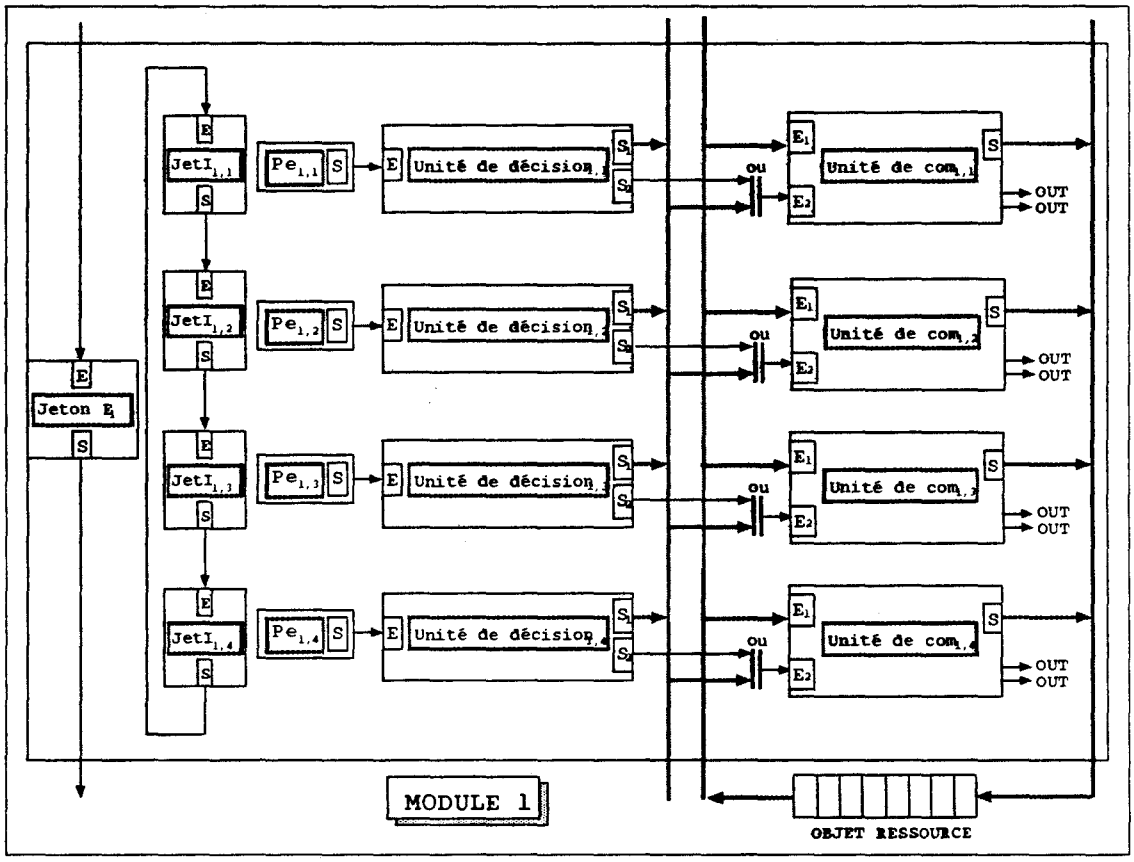

Figure 4.10 : Modélisation du Module de Communication de Base

4.4 Simulation du réseau ARP sous Qnap2

Nous venons de décrire la modélisation du réseau ARP à l'aide d'un réseau de files d'attente. Cette solution nous permet d'étudier et de déterminer le comportement du réseau ARP en fonction des paramètres physiques qui comprend : N (le nombre de processeurs), V (le nombre de voies), D (le débit de la liaison série), P (la puissance du processeur) et K (le nombre de processeurs par MCB) ; et les paramètres logiciels qui sont I (le nombre d'instructions) et L (la longueur du message échangée).

Nous ne disposons que d'une version de Qnap pour les stations sun 3/80. Pour des raisons de vitesse de calcul et de taille mémoire, nous nous limitons dans le cadre

de ce travail à la simulation d'un réseau à 128 processeurs. Les processeurs sont répartis régulièrement dans 16 MCB. Le choix de mettre 8 processeurs par MCB a été dicté par les contraintes d'intégration, nous verrons dans le §5.8 qu'il est difficile d'intégrer plus de 8 processeurs dans un même MCB avec la technologie que nous disposons.

L'essentiel de cette partie du travail consiste en plusieurs points :

- déterminer le domaine de validité d'ARP, c'est à dire, pour des paramètres physiques fixes, quelles sont les valeurs limites de I et de L que le système peut supporter. Nous compareront ces résultats avec les résultats obtenus par calcul.

- déterminer les temps de réponse des différentes unités qui composent le MCB au cours d'une simulation d'exécution. Ces temps prennent en comptes des paramètres liés au fonctionnement réel comme le conflit d'accès au processeur destinataire, ou au lien de communication. Ils constituent donc des informations plus significatives que les temps relevés en simulations électriques sous Solo1400, qui sont néanmoins nécessaires, puisqu'ils servent de données d'entrées pour ces simulations.

- comparer 2 architectures, qui possèdent les mêmes paramètres physiques, sauf le nombre de liens, dans différentes situations de fonctionnement, c'est à dire pour différents rapport de $\frac{L}{I}$.

4.4.1 Quelles valeurs choisir pour I et L ?

Avant toute simulation il convient de déterminer judicieusement un jeu de valeurs pour I et L, afin d'éviter des simulations inutiles. Nous disposons pour cela des inéquations 2.4, 3.1 sur la charge du réseau et la saturation du bus de contrôle. Nous disposons également des résultats issus du test d'un MCB sous solo1400 au §5.8.3 qui nous donne les temps de circulation des jetons, et de traitement des requêtes d'acquisition et de libération.

L'inéquation 3.1 va nous permettre d'établir la borne inférieure de I en fonction du nombre de processeurs. Pour simplifier l'inéquation nous supposons que le temps de contrôle T est négligeable. Nous posons $\boxed{T_R = \frac{I}{P}}$ et $\boxed{T_T = \frac{N_m}{D_{bus}}}$. T_R est la durée séparant 2 requêtes de communication, et T_T est la durée nécessaire au traitement d'une requête par une UD sans tenir compte du temps d'arbitrage. En remplaçant les variables I, P, N_m et D_{bus} par T_R et T_T dans cette inéquation,

nous obtenons alors l'inéquation suivante :

$$\boxed{N \times T_T \leq T_R} \tag{4.1}$$

La valeur de T_T est connue pour les traitements des 2 types de requêtes. Elle est de l'ordre de 80 ns pour le traitement une requête d'acquisition et de 50 ns pour une libération. Par ailleurs, la vitesse de circulation du jeton est de l'ordre de 5 ns, ce qui constitue la durée minimale. Dans tout ce qui suit, nous définissons comme unité de temps noté **ut** la durée correspondant à 5 ns. Donc, pour le nombre de processeurs N égal à 128, nous obtenons une borne inférieure pour T_R égale à 2048 ut.

Par ailleurs en posant $\boxed{\mathbf{T_L = \frac{L}{D_{res}}}}$, où T_L est la durée de transmission d'un message. Et en remplaçant les variables I, P, L et D_{res} par T_R et T_L dans l'inéquation 2.4, nous obtenons alors l'inéquation suivante :

$$\boxed{T_L \leq \frac{V}{N} \times T_R} \tag{4.2}$$

ainsi, cette inéquation nous fournie une borne supérieure pour la valeur de T_L pour des valeurs de N, de V et de T_R données.

Les variables T_R et T_L constituent des données d'entrées du simulateur. En combinant les variables P avec I et L avec D_{bus}, cela évite de figer la simulation pour un type de processeur et un type de support de communication donnés. De plus, ces variables, qui représentent des rapports, fournissent comme solution non pas un couple de données, mais un ensemble de couples de données ayant le même rapport. Par exemple pour $T_R = 100$ ut, I est égale à 100 si le processeur est un T9000, il est égal à 50 pour un i860XP.

L'approche, que nous adoptons, consiste à fixer la valeur de T_R, et ensuite à simuler le réseau avec différentes valeurs de T_L. La valeur de N étant fixe (=128), les seules valeurs de V intéressantes à simuler sont 64 et 32, puisque, dans notre cas, nous avons vu dans le §2.3.2.2 qu'à cause des problèmes de conflits d'accès au réseau de communication et au processeur destinataire, le nombre de processeurs simultanément en communication ne dépasse pas 63%.

Nous présentons dans ce qui suit des résultats issus des simulations pour des valeurs de T_R égales à 4000 et 2000 ut. Les simulations utilisant la première valeur de T_R donnent les caractéristiques du système en fonctionnement en mode équilibré et en mode saturation du réseau de communication, et celles utilisant la deuxième valeur, qui est inférieure à la valeur limite de T_R, donnent les caractéristiques en fonctionnement en mode saturation du réseau de contrôle. Les

résultats sont donnés sous formes graphiques, car ils permettent de montrer si le système est dans un état stable ou non.

4.4.2 Simulations avec $T_R = 4000$ ut

Pour chaque valeur de V (32 et 64), nous avons effectué des simulations avec 3 valeurs de T_L (400, 1000, 2000) qui correspondent à 3 rapports de $\frac{T_L}{T_R}$: $\frac{1}{10}$, $\frac{1}{4}$ et $\frac{1}{2}$. L'inéquation 4.2 permet de connaître à l'avance les rapports de $\frac{T_L}{T_R}$ qui aboutissent à une saturation, puisque le rapport $\frac{V}{N}$ est connu et il est égal soit à $\frac{1}{2}$ pour V=64, soit $\frac{1}{4}$ pour V=32. Nous obtenons 8 graphiques représentant les paramètres principaux du système en fonction du temps de simulation (en abscisse), dont l'unité de temps ut est égale à 5 ns. Ces graphiques sont :

G_1 - le nombre moyen de processus en communication.
G_2 - le nombre moyen de processus en attente de traitement de requête de communication dans toutes les UD. Ce nombre comprend des requêtes venant du processeur et des requêtes qui ont été déjà traitée et qui ont abouti à un échec (destinataire est occupé, le processeur est pris en destinataire ou plus de lien libre).
G_3 - la charge du bus de contrôle d'acquisition.
G_4 - la charge du bus de contrôle de libération.
G_5 - le temps moyen de réponse d'un service, qui comprend le temps d'attente dans la file, et le temps passé pour le traitement de la requête.
G_6 - le temps moyen de réponse d'un client (processus). Il indique le temps entre le moment où une requête est transmise à l'UD et le moment où l'UD informe le processeur que la liaison est établie.
G_7 - le nombre total de communications effectuées.
G_8 - le temps moyen de réponse du jeton de requête externe.

Nous donnons dans ce qui suit les différents graphiques que nous venons de mentionner. Les courbes sont souvent confondues, nous avons ajoutés à la fin de la légende de chaque courbe une valeur entre parenthèses, qui correspond à la valeur finale du paramètre. Ces valeurs sont rappelées dans le tableau 4.2.

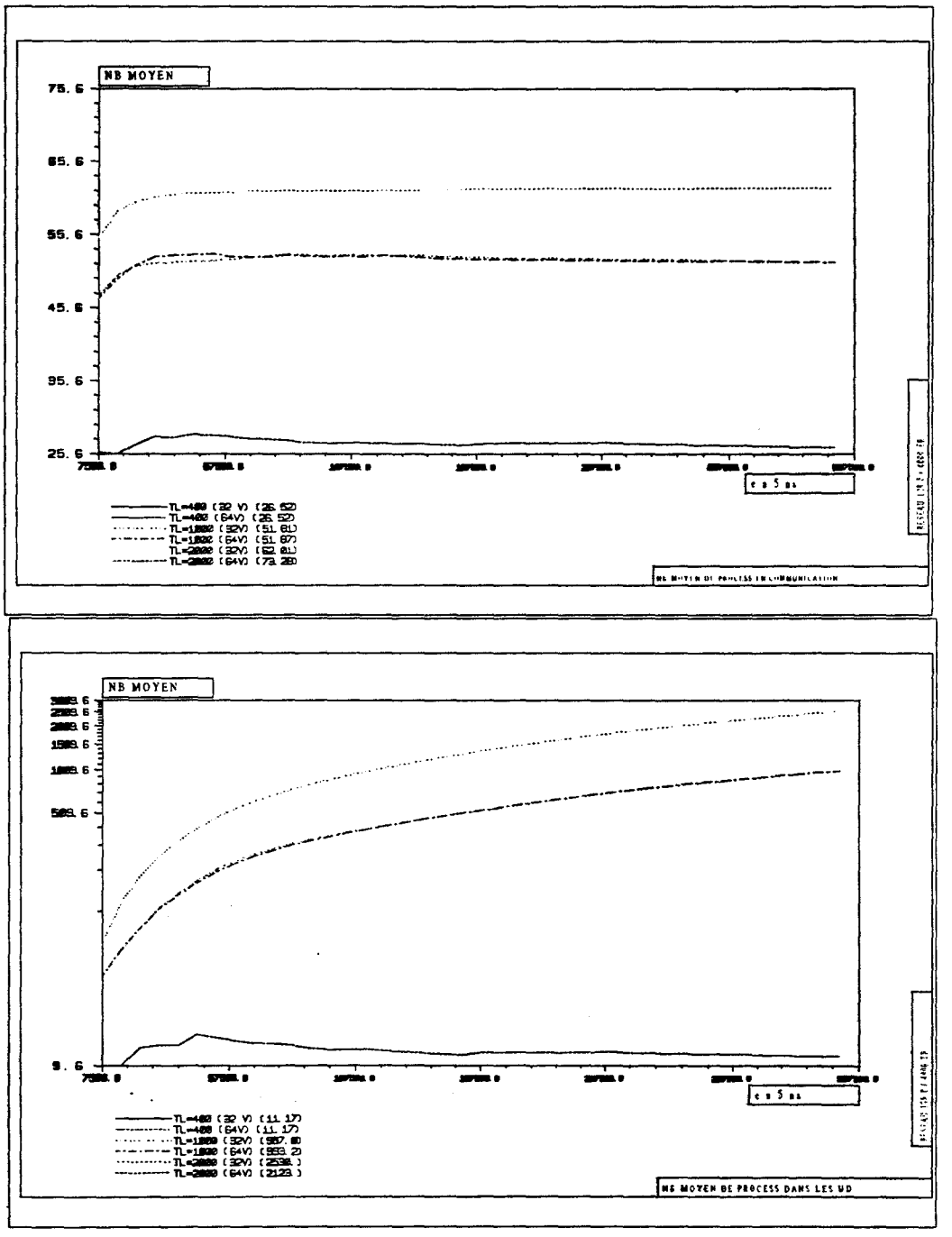

Figure 4.11 : Les résultats des simulations avec $T_R = 4000$ ut : G_1 le nombre moyen de processus en communication (à gauche), G_2 le nombre moyen de processus dans tous les UD (à droite)

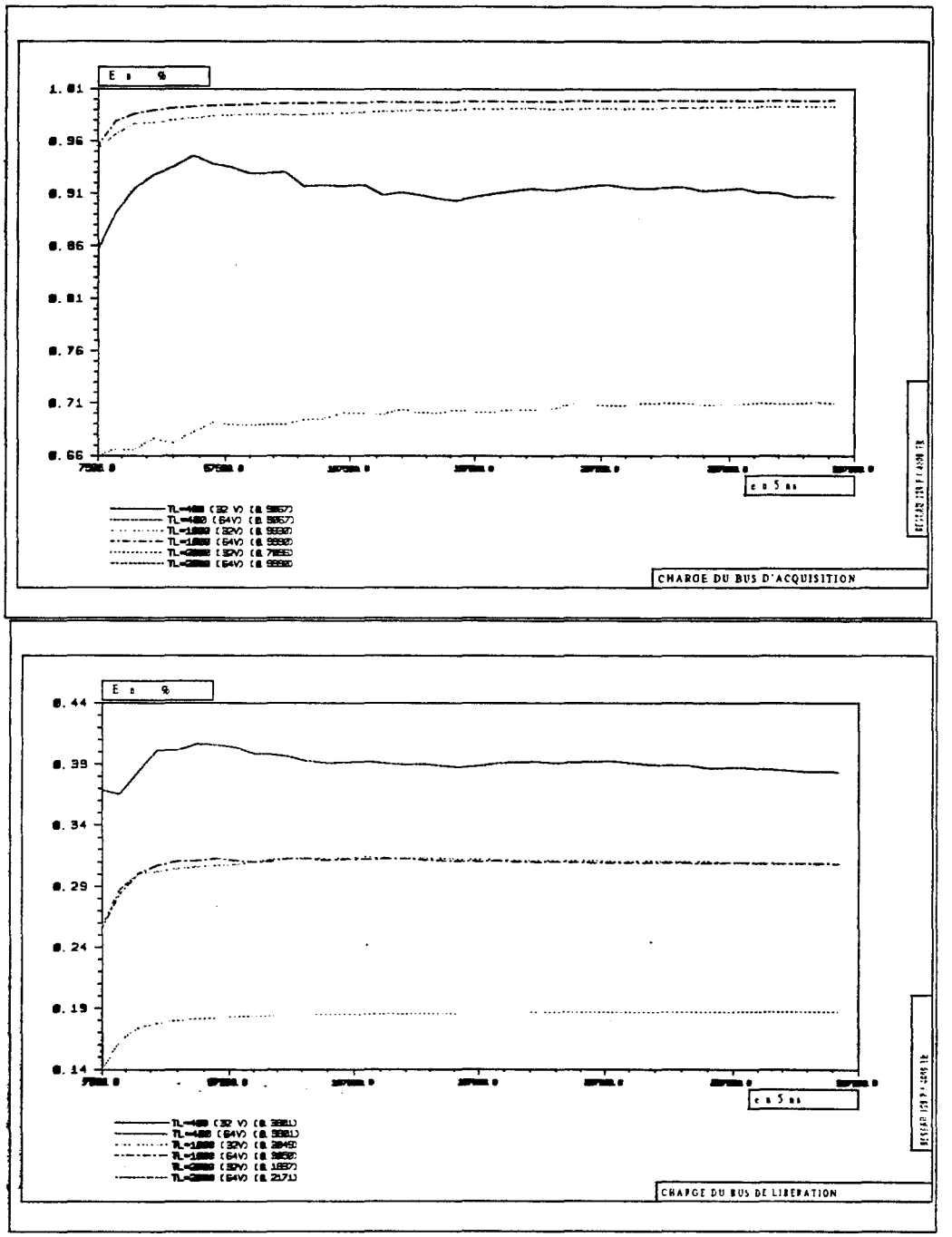

Figure 4.12 : Les résultats des simulations avec T_R = 4000 ut : G_3 la charge du bus d'acquisition (à gauche), G_4 la charge du bus de libération (à droite)

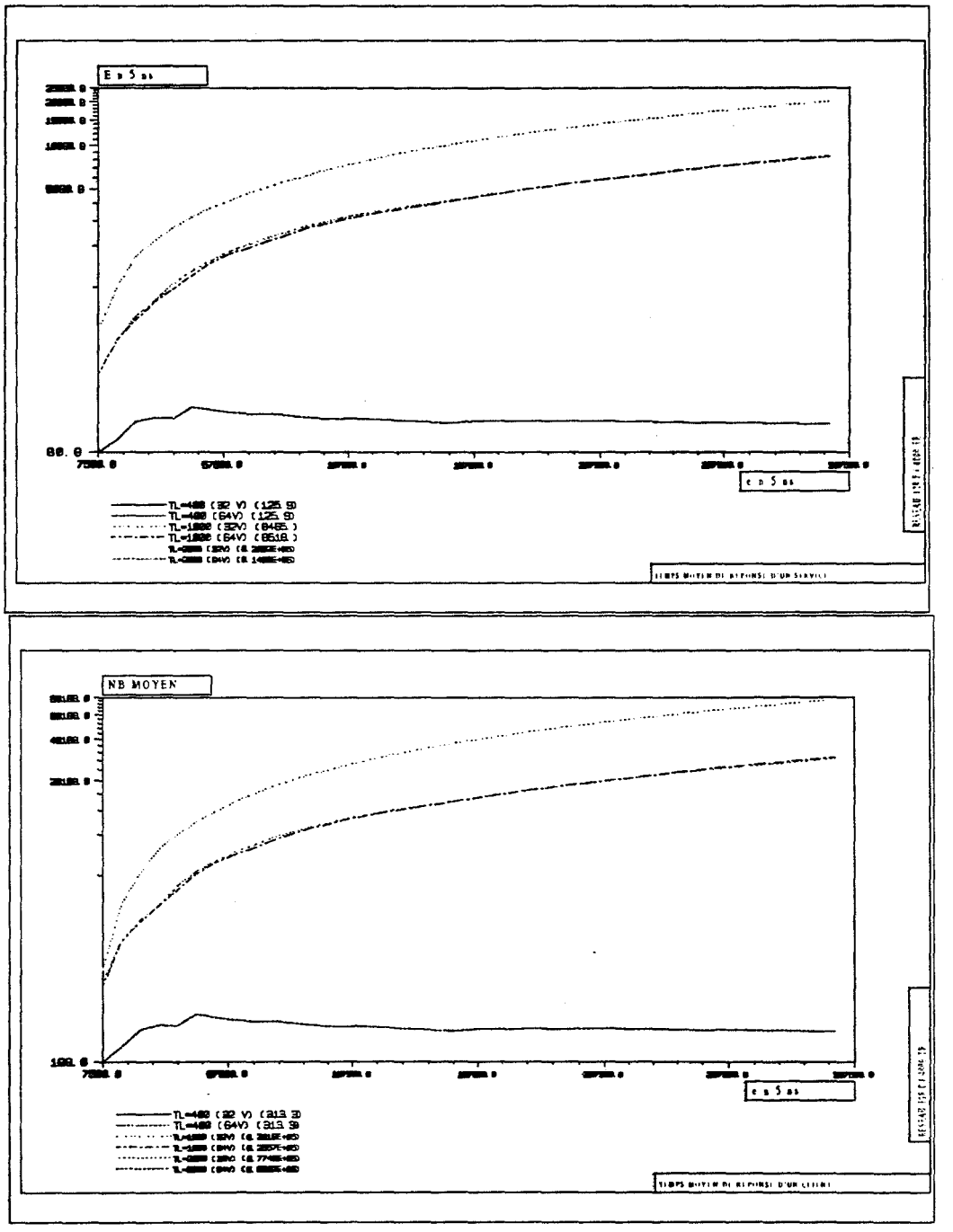

Figure 4.13 : Les résultats des simulations avec T_R = 4000 ut : G_5 le temps moyen de réponse d'un service (à gauche), le temps moyen de réponse d'un client (à droite)

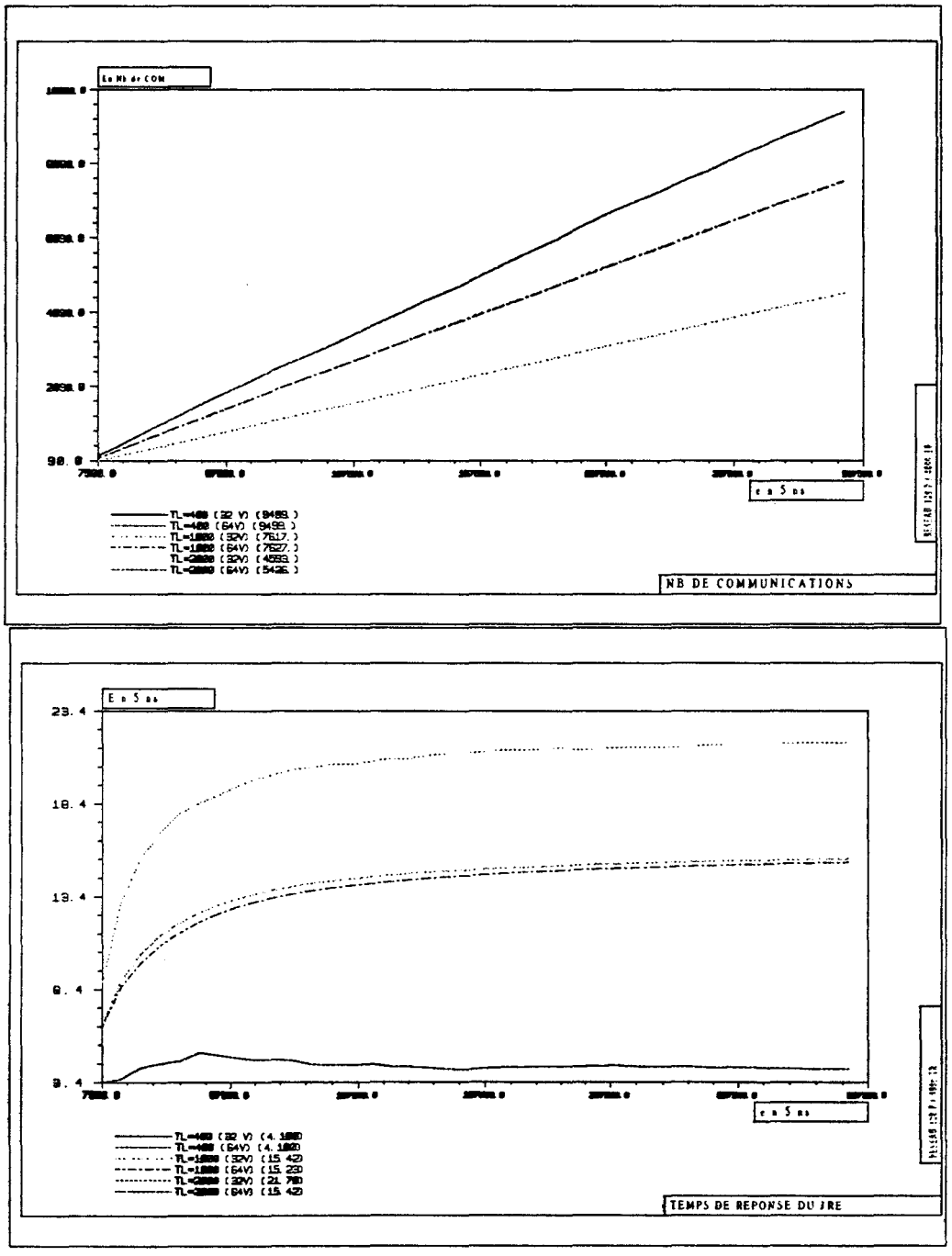

Figure 4.14 : Les résultats des simulations avec T_R = 4000 ut : G_7 le nombre total de communicatione (à gauche), G_8 le temps moyen de réponse du jeton de requête externe (à droite)

Tableau 4.2 : Les valeurs des paramètres du système obtenue à la fin des simulations pour T_R=4000

	T_R=4000, T_L=400		T_R=4000, T_L=1000		T_R=4000, T_L=2000	
	V=32	V=64	V=32	V=64	V=32	V=64
G_1 : nombre moyen de processus en communication	26.52	26.52	51.81	51.87	62.01	73.28
G_2 : nombre moyen de processus en attente dans les UD	11.17	11.17	987	993.2	2530	2123
G_3 : la charge du bus d'acquisition (en%)	90.67	90.67	99.3	99.9	70.96	99.9
G_4 : la charge du bus de libération (en%)	38.01	38.01	30.49	30.5	18.37	2171
G_5 : le temps moyen de réponse d'un service (en 5 ns)	125.9	125.9	8465	8518	20320	14890
G_6 : le temps moyen de réponse d'un client (en 5 ns)	313.3	313.3	29160	29570	77490	65020
G_7 : le nombre total de communications effectuées	9499	9499	7617	7627	4593	5426
G_8 : le temps moyen de réponse du JRE (en 5 ns)	4.1	4.1	15.42	15.23	21.70	15.42

4.4.2.1 Le fonctionnement en mode équilibré ($T_L \leq 400$)

Le graphique G_2 (figure 4.11) montre l'évolution du nombre de processus en attente de traitement dans les UD. Nous voyons que pour les valeurs de T_L égales à 400, ce nombre reste constant et est égal à 11. Le système est équilibré, c'est à dire que le flux des requêtes provenant des processeurs est égal au flux des requêtes satisfaites par les MCB. Le temps de traitement des requêtes est constant (G_6 figure 4.13) et égal à 313 ut. Ce dernier est supérieur au temps moyen de réponse d'un service (G_5 figure 4.13), car il comprend à la fois les temps de traitement d'une acquisition et d'une libération, et le temps de traitement supplémentaire introduit par des possibilités de conflits divers : de destinataire, d'expéditeur ou de lien de communication.

Le nombre moyen de processus en communication pour les T_L inférieur ou égal 400 se stabilise au voisinage de 26 (G_1 figure 4.11). Nous obtenons les mêmes résultats pour le nombre de voies de communication égal à 32 et 64. Dans ce cas de figure, un réseau de communication à 14 voies est suffisante. Le nombre de voies effectivement occupé, que nous obtenons par simulation, correspond aux résultats déterminés par calcul dans le tableau 2.4.

Le graphe G_3 (figure 4.12) montre que le bus de requête est chargé à plus de 90%, et comme nous venons de voir que le système est équilibré. Nous pouvons en déduire que la fréquence d'arrivée des requêtes, sans introduire de saturation, est ici proche de l'optimum. Le bus d'acquisition constitue l'élément clef de ARP, puisque le nombre de requêtes, que le système peut satisfaire, est limité. Ce nombre est liée directement à l'utilisation efficace et à la performance de ce bus.

Le bus de libération subit moins de contraintes : primo, le nombre de requêtes de libération ne peut pas augmenter par rapport aux requêtes satisfaites par les MCB, puisqu'une requête de libération n'aboutit théoriquement jamais à un échec, secundo, le bus d'acquisition a une capacité limite de traitement, au delà de laquelle il se sature, et tertio, le temps de traitement d'une acquisition est supérieur au temps de libération, ces raisons font que le bus de libération ne pourra jamais être saturé. Son taux d'occupation est de l'ordre de 38% pour T_L = 400 (G_4 figure 4.12).

4.4.2.2 Le fonctionnement en mode saturé ($T_L \geq 1000$)

Nous allons maintenant nous intéresser aux situations qui aboutissent à une saturation du réseau. Pour des valeurs de T_L supérieures ou égales à 1000, le graphe G_2 montre que le nombre de processus en attente dans les UD augmente linéairement. Comme le flux d'arrivé des requêtes de communication est le même que précédemment, la cause de la saturation provient de la durée des communications T_L. Les processus communiquent plus longtemps, ce qui augmente les probabilités de conflits d'accès. Les requêtes qui aboutissent à des échecs, tenteront de réeffectuer la demande. Ils s'ajoutent alors au flux des requêtes qui proviennent des processus. Nous avons vu que T_L=400 constitue avec T_R un flux des requêtes limite sans saturation. Une augmentation du flux dû aux conflits, augmente le nombre des requêtes à traiter. Et comme pour T_L=400 le bus d'acquisition possède déjà une charge de 90%, les requêtes, qui ne peuvent pas être satisfaites, sont mises en file d'attente des UD. Et la longueur de ces files ne cesse de croître avec le temps (G_2), ce qui augmente le temps moyen de service (G_5) et le temps de réponse moyen d'un client (G_6).

La réponse du jeton de requête externe (G_8 figure 4.14) a une valeur proche du

temps nécessaire au traitement de la requête de communication. En effet, la réponse du JRE obtenu en divisant le temps de simulation total par le nombre de places que JRE a parcourues pendant la simulation. Plus le nombre de requêtes est élevé, moins le JRE franchit de places pour se positionner sur la requête suivante, et reste bloquer jusquà la fin du traitement de la requête courante. Comme le nombre de requêtes en attente est très élevé, le JRE s'arrête pratiquement à chaque place, c'est pourquoi son temps de réponse est proche du temps de traitement.

Pour T_L=1000 et V=32,64 et pour T_L=2000 et V=64, la charge du bus d'acquisition atteint ici son optimum (G_3), elle dépasse 99%. Nous pouvons dire que le système est en saturation de contrôle, puisque le bus d'acquisition est à 99%, alors que la charge du réseau de communicaion n'atteint pas son optimum (G_1).

La charge du bus est plus faible (70%) pour le cas où T_L=2000 et V=32. La raison est que le nombre moyen de processus en communication (G_1) est proche de la limite admissible, il est égale à 62 pour une limite de 64. Lorsque tout le réseau est occupé, la nouvelle requête admissible (§3.5) possesseur du bus d'acquisition va le conserver jusqu'à ce qu'un lien se libère, pendant ce temps aucune autre requête n'est traitée, ce qui réduit la charge de ce bus. Nous appelerons ce phénomène **la saturation en communication**.

L'utilisation de 64 liens de communication supprime ce phénomène, elle autorise plus de communications simultanées. Le gain commence à être significatif pour T_L=2000, le nombre total de communication (G_7 figure 4.14) est de 5426 contre 4593 pour V=32, soit un gain de 18%. Le temps de réponse est également amélioré ainsi que le nombre de processus en attente. Mais pour cette valeur de T_L, le système reste néanmoins en saturation. Et nous voyons que pour la valeur de T_L égale à 1000, le gain obtenu par l'apport des 32 liens supplémentaires est minime. Le nombre de processus en communication est voisin de 51 (pour une valeur maximale de 64). Pour des valeurs de T_L inférieures à 1000, l'utilisation de 64 liens de communication n'est pas justifiée, d'autant que le coût en matériel est doublé par rapport à V=32.

Pour $T_L \geq 1000$, le nombre moyen de processus en communication est plus grand que pour $T_L \leq 400$, il atteint une moyenne de 73 processus (soit 37 liens utilisés) pour T_L=2000 et V=64. Nous voyons que le nombre obtenu est très inférieur à la valeur obtenue par calcul (50 liens). Ceci est liée principalement au temps de contrôle, qui n'a pas été pas pris en comptes dans les formules.

4.4.3 Simulations avec T_R = 2000 ut

Nous présentons ici les résultats issus des simulations effectuées avec T_R=2000. Nous avons procédé de la même façon que pour T_R égal à 4000. C'est à dire pour chaque valeur de V (32 et 64), nous avons effectué des simulations avec 3 valeurs de T_L (200, 500, 1000) qui correspondent à 3 rapports de $\frac{T_L}{T_R}$: : $\frac{1}{10}$, $\frac{1}{4}$ et $\frac{1}{2}$. Nous présentons ici uniquement les graphiques G_1, G_2, G_3, G_4, G_5 et G_8 (figures 4.15, 4.16). Puisque le graphique G_6 (temps de réponse d'un client) a la même allure que G_5 (temps de réponse d'un service), et que le nombre de communications effectué (G_7) n'a pas d'importance ici. Les valeurs finales des paramètres sont regroupés dans le tableau 4.3.

Tableau 4.3 : Les valeurs des paramètres du système obtenue à la fin des simulations pour T_R=2000

	T_R=2000, T_L=200		T_R=2000, T_L=500		T_R=2000, T_L=1000	
	V=32	V=64	V=32	V=64	V=32	V=64
G_1 : nombre moyen de processus en communication	16.84	16.84	31.72	31.72	49.77	49.37
G_2 : nombre moyen de processus en attente dans les UD	3514	3514	4830	4830	5855	5855
G_3 : la charge du bus d'acquisition (en%)	99.98	99.98	99.98	99.98	99.68	99.98
G_4 : la charge du bus de libération (en%)	46.13	46.13	36.66	36.66	29.27	29.04
G_5 : le temps moyen de réponse d'un service (en 5 ns)	19430	19430	24130	24130	26170	26200
G_6 : le temps moyen de réponse d'un client (en 5 ns)	51990	51990	73370	73370	91880	92540
G_7 : le nombre total de communications effectuées	11530	11530	9165	9165	7319	7261
G_8 : le temps moyen de réponse du JRE (en 5 ns)	15.88	15.88	15.87	15.87	15.90	15.84

Les résultats obtenus confirme ceux issus de l'inéquation 4.1. En effet, les résultats

des simulations montrent que, quels ques soient le rapport $\frac{T_L}{T_R}$ et la valeur de V, les courbes donnant le nombre de processus en attente (G_2), le temps moyen de réponse d'un service (G_5) croissent linéairement en fonction du temps et le JRE a un temps de réponse (G_8) qui tend vers la valeur de T_T (Temps de traitement d'une requête). Ces paramètres indiquent que le flux des requêtes à traiter est plus fort que celui que les UD peuvent traiter, puisque la charge du bus de contrôle d'acquisition (G_3) atteint pratiquement le maximum (plus de 99%), le bus ne dispose donc pas du débit nécessaire dans ce cas. Le nombre moyen de processus en communication (G_1) ne dépasse pas 50 aux meilleurs des cas, ce qui fait 25 voies de communication occupées, le réseau de communication n'est donc que moyennement chargé. La saturation du système provient donc de son réseau de contrôle.

Et pourtant certaines valeurs de $\frac{T_L}{T_R}$, par exemple pour T_L=200 et pour V=32,64, vérifient l'inégalité 4.2. C'est à dire que le réseau dispose d'un débit supérieur aux besoins en communication des processeurs.

Il est donc nécessaire que les valeurs de N, V, T_L et T_R vérifient à la fois les inéquations 4.2 et 4.1. Mais cette condition est nécessaire, mais pas suffisante, puisque dans les simulations précédentes avec T_R=4000, le rapport de $\frac{T_L}{T_R}$ égal à $\frac{1}{4}$ pour V=64, vérifie les 2 inéquations, mais malgré cela, les résultats de la simulation montrent que le système aboutit à une saturation. Nous allons dans ce qui suit tenter d'expliquer ce phénomène.

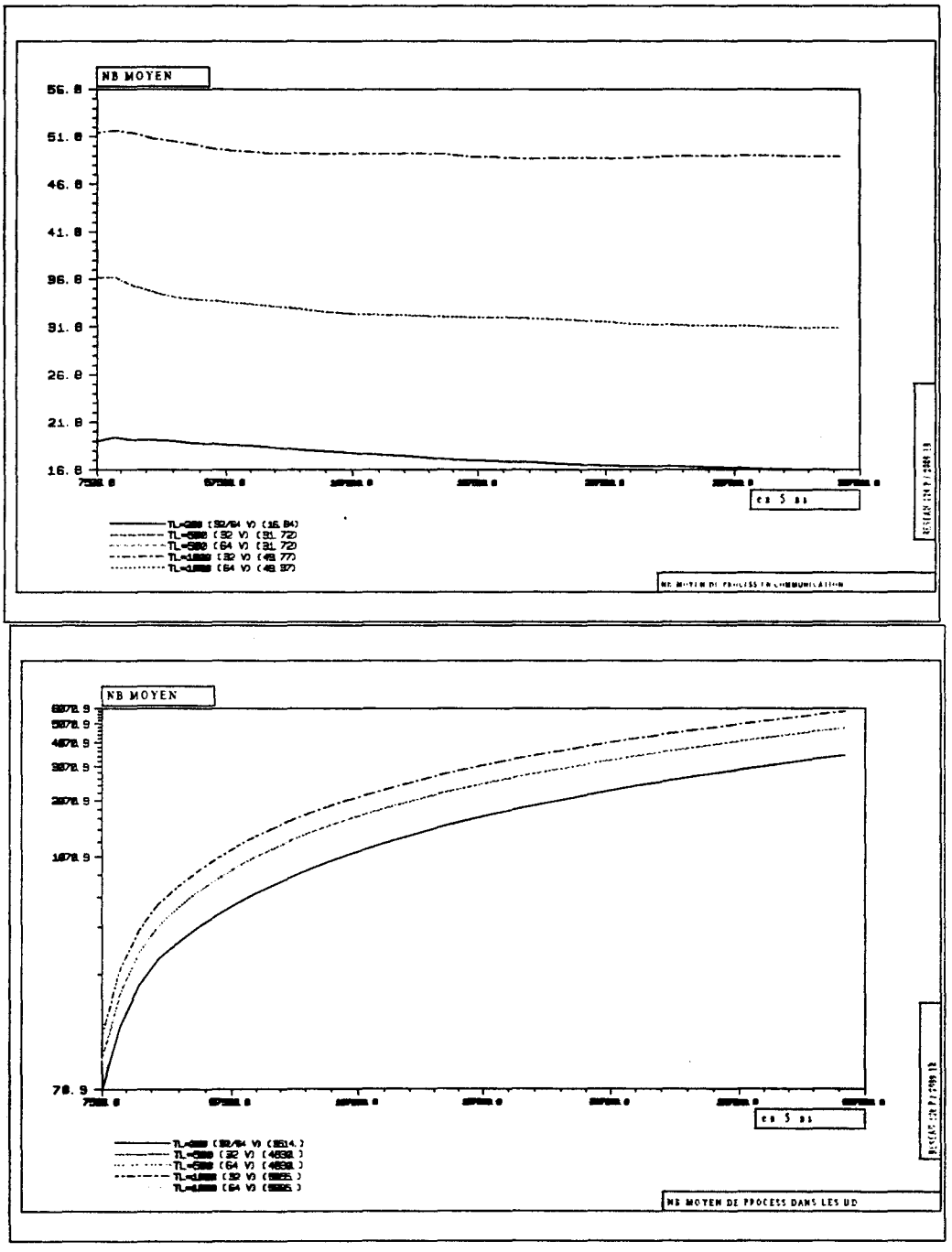

Figure 4.15 : Les résultats des simulations avec T_R = 2000 ut : G_1 le nombre moyen de processus en communication (à gauche), G_2 le nombre moyen de processus dans tous les UD (à droite)

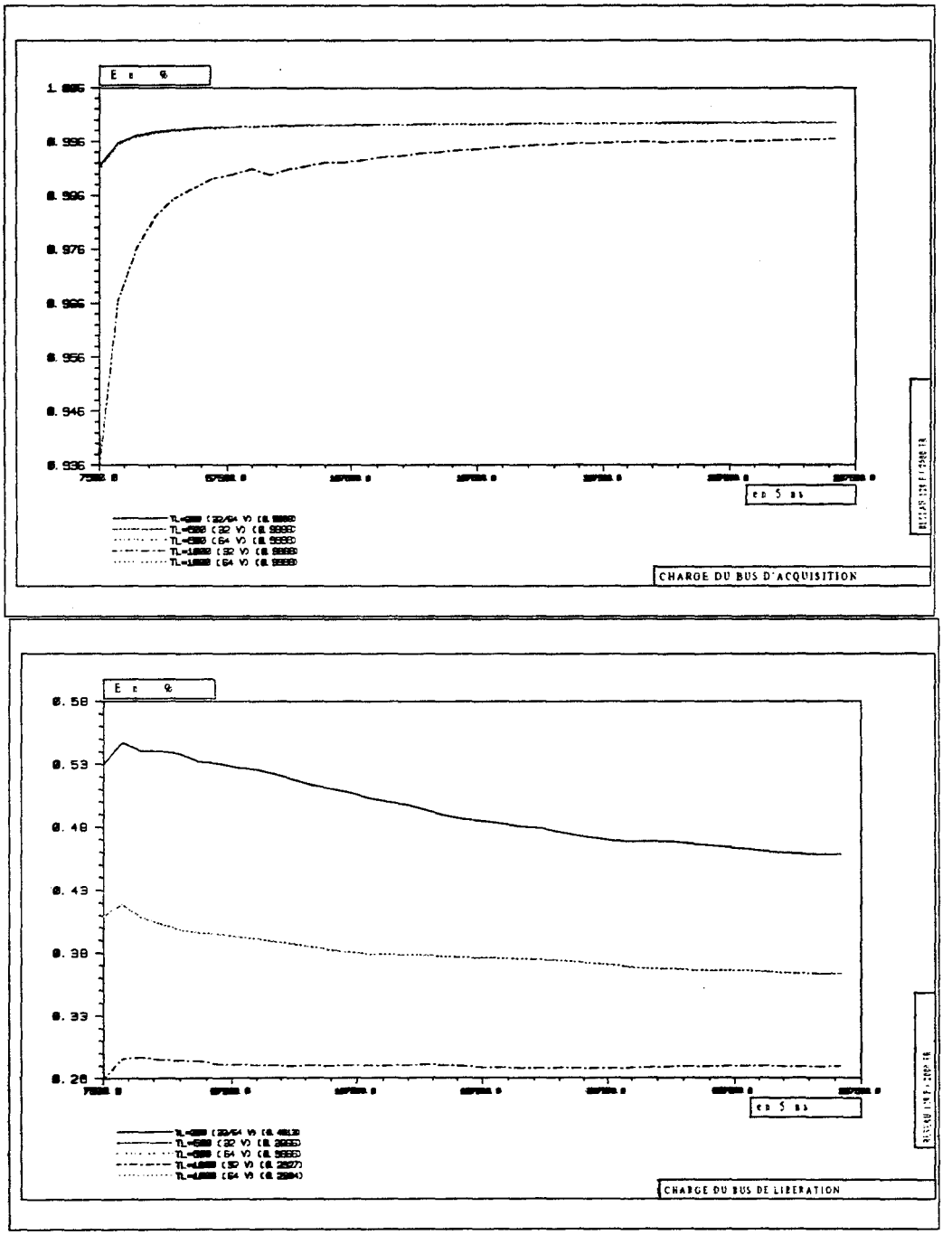

Figure 4.16 : Les résultats des simulations avec T_R = 2000 ut : G_3 la charge du bus d'acquisition (à gauche), G_4 la charge du bus de libération (à droite)

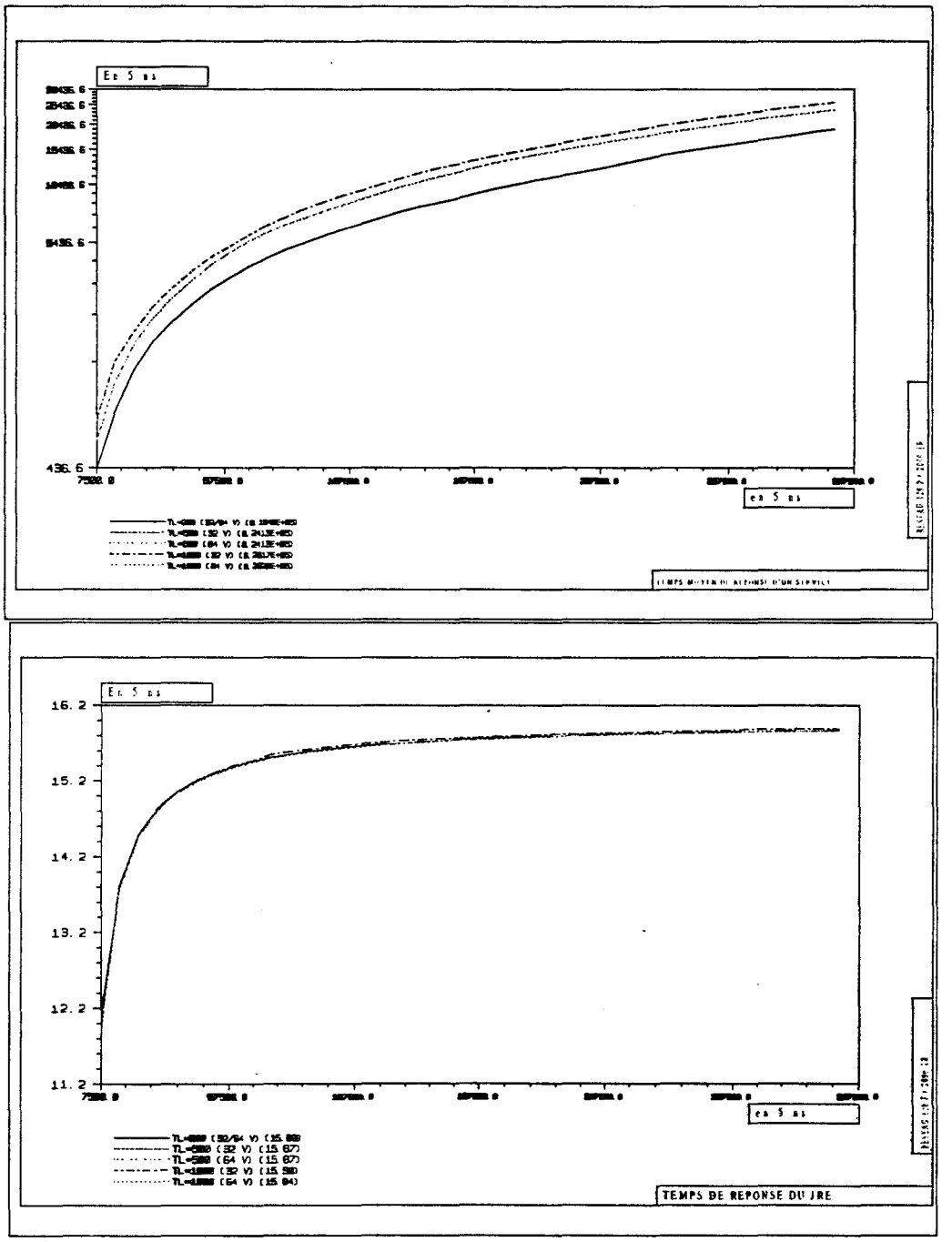

Figure 4.17 : Les résultats des simulations avec T_R = 2000 ut : G_5 le temps moyen de réponse d'un service (à gauche), G_8 le temps moyen de réponse du jeton de requête externe (à droite)

4.4.4 Le conflits d'accès au processeur

Dans le simulateur que nous avons écrit, nous échantillonnons des résultats donnant l'état du système à des intervalles de temps fixes, ces résultats sont utilisés pour générer les courbes que nous venons de présenter. A la fin de la simulation des résultats plus précis sont récupérés. Nous donnons dans le tableau 4.4, les résultats qui concernent les différentes classes de clients qui ont été servis par l'UD au cours de la simulation.

Nous rappelons que les requêtes provenant du processeur, qui entrent dans l'UD, sont des clients de *classe Demande*. Lorsque le traitement de la requêtes n'aboutit pas à une communication, le requête retourne dans la file d'attente de l'UD après avoir changer de classe : en *classe Non expéditeur* si l'expéditeur est en cours de communication, en *classe Non destinataire* si le destinataire est en cours de communication. Et enfin, lorsque la communication se termine le client retourne dans la file d'UD pour effectuer la libération de voie.

Tableau 4.4 : Les clients des différentes classes servis par l'UD au cours de la simulation

	T_R=4000, T_L=400		T_R=4000, T_L=1000		T_R=4000, T_L=2000	
	V=32	V=64	V=32	V=64	V=32	V=64
Demande d'acquisition	74.34	74.34	60.30	60.38	36.77	43.30
Non expéditeur	3.79	3.79	17.74	17.55	11.46	16.44
Non destinataire	54.72	54.72	67.42	68.40	55.74	86.59
Demande de libération	74.21	74.21	59.51	59.57	35.88	42.39
Total	207.05	207.05	204.97	205.90	139.85	188.72

La charge du bus de contrôle d'acquisition dépend non seulement des requêtes de classe *demande* (**RCD**), mais aussi des requêtes de classe non destinataire (**RCND**). Les résultats du tableau montre que le nombre de clients (requêtes) de classe *non expéditeur* et *non destinataire* traités par l'UD au cours de la simulation est non négligeable devant le nombre de requêtes de classe *demande*. Nous constatons d'après ce tableau que plus le message est long, plus le nombre de RCND est grand. Ce nombre n'a pas du tout été prise en comptes dans l'inéquation 4.1.

Ce qui explique que pour le cas T_L=1000, T_R=4000 et V=64, bien que les paramètres vérifient ces inéquations, le système se sature en contrôle puisqu'il reste des liens non occupés. Si maintenant nous prenons en comptes le nombre des RCND, il faudrait alors ajouter ces requêtes aux RCD. La somme des 2 classes de requêtes constituent la charge réelle du bus d'acquisition. Le nombre de RCND à ajouter peut s'exprimer par le rapport k entre le nombre moyen de

RCND et le nombre moyen de RCD servis par l'UD au cours l'exécution. En tenant comptes de ce rapport k, l'inéquation 4.1 devient :

$$\boxed{(1+k) \times N \times T_T \leq T_R} \tag{4.3}$$

En appliquant cette nouvelle inégalité aux jeux de données que nous avons choisis initialement, et en déterminant la valeur de k par les résultats fournis par les simulations (le rapport des RCND sur les RCND). Nous obtenons le tableau 4.5, qui exprime le nombre minimal de T_R nécessaire pour éviter la saturation en contrôle en tenant comptes des RCND. Les valeurs de T_R minimal sont données en fonction des paramètres de simulation T_R, T_L et V.

Tableau 4.5 : Le nombre T_R minimal pour éviter la saturation en contrôle

	T_R=4000, T_L=400		T_R=4000, T_L=1000		T_R=4000, T_L=2000	
	V=32	V=64	V=32	V=64	V=32	V=64
	K=0.73	K=0.73	K=1.11	K=1.13	K=1.51	K=1.99
T_R minimal	3543	3543	4321	4362	5140	6123

Le tableau montre que le T_R minimal obtenu par calcul est inférieur uniquement pour les cas où T_L est égal à 400. Pour les autres valeurs de T_L la valeur de T_R minimal obtenue est supérieur au T_R utilisé pendant la simulation. Ceci explique les cas de saturation en contrôle que nous avons obtenus.

4.5 Conclusion

Nous avons présenté dans ce chapitre, une méthode d'évaluation des performances d'un système informatique en utilisant comme outil de base un logiciel d'aide à la modélisation par files d'attente. Un ensemble de symboles graphiques décrivant les différentes primitives de Qnap a été proposés et utilisés pour modéliser l'ARP.

Les résultats issus des simulations par Qnap, nous a non seulement permis de vérifier les résultats déterminés dans les chapitres précédent, comme l'occupation maximale du réseau de communication et la saturation du bus de contrôle. Mais il nous a aussi permis d'affiner ces résultats.

Un des intérêts essentiels de cette méthode est qu'elle permet de connaître le comportement de toutes les unités du système en fonction des paramètres N, V, T_R et T_L. L'information la plus importante que nous tirons de ces simulations est la forte influence des RCND. Ce paramètre n'ayant pas été pris en

compte précédemment constitue un élément limitateur de performances. Nous proposerons dans le chapitre 6, des solutions permettant de réduire le nombre de RCND à traiter au cours d'une simulation.

Chapitre 5

Réalisation du MCB sous SOLO1400

2 février 1993

Un module de communication de base (MCB) a été réalisé à l'aide du logiciel Solo1400 de chez ES2. Solo1400 permet de décrire et de simuler le comportement du circuit aussi bien logiquement qu'électriquement. Par ailleurs, il génère une description du circuit pour sa fabrication en fonderie. Solo1400 dispose de 2 types de technologie d'intégration : 1.2 et 1.5 micromètre, dont les surfaces maximales d'intégration sont respectivement de 2 $mm^2 et 1\ mm^2$. Pour des considérations d'ordre purement économique, toute l'étude est basée sur l'utilisation d'un boîtier PGA de 144 broches.

La description de la réalisation des différentes fonctions du MCB, qui ont été présentées dans le chapitre 3, est exposée dans ce chapitre. Et en plus des difficultés de réalisation de ces fonctions, l'utilisation du logiciel Solo1400 ajoute des contraintes supplémentaires imposées par le constructeur ES2, notamment sur le nombre de broches et la surface intégrable. Un MCB est constitué par un vecteur des liens et des sous-ensembles identiques qui se composent d'une unité de décision, d'une unité de communication et d'une partie de l'unité d'arbitrage. Le nombre de ces sous-ensembles est limité par la surface intégrable; il est donc envisageable d'augmenter ce nombre dans un contexte plus favorable, de même en ce qui concerne le nombre de liens de communication qui est limité ici au maximum à 64, puisque le nombre de broches est de 144. Rappelons, à titre indicatif, que le processeur Alpha 21064 est intégré dans un boîtier PGA de 431 broches.

5.1 Vue générale d'un MCB

La figure 5.1 montre les différentes unités qui composent un MCB. Chaque MCB est composé :

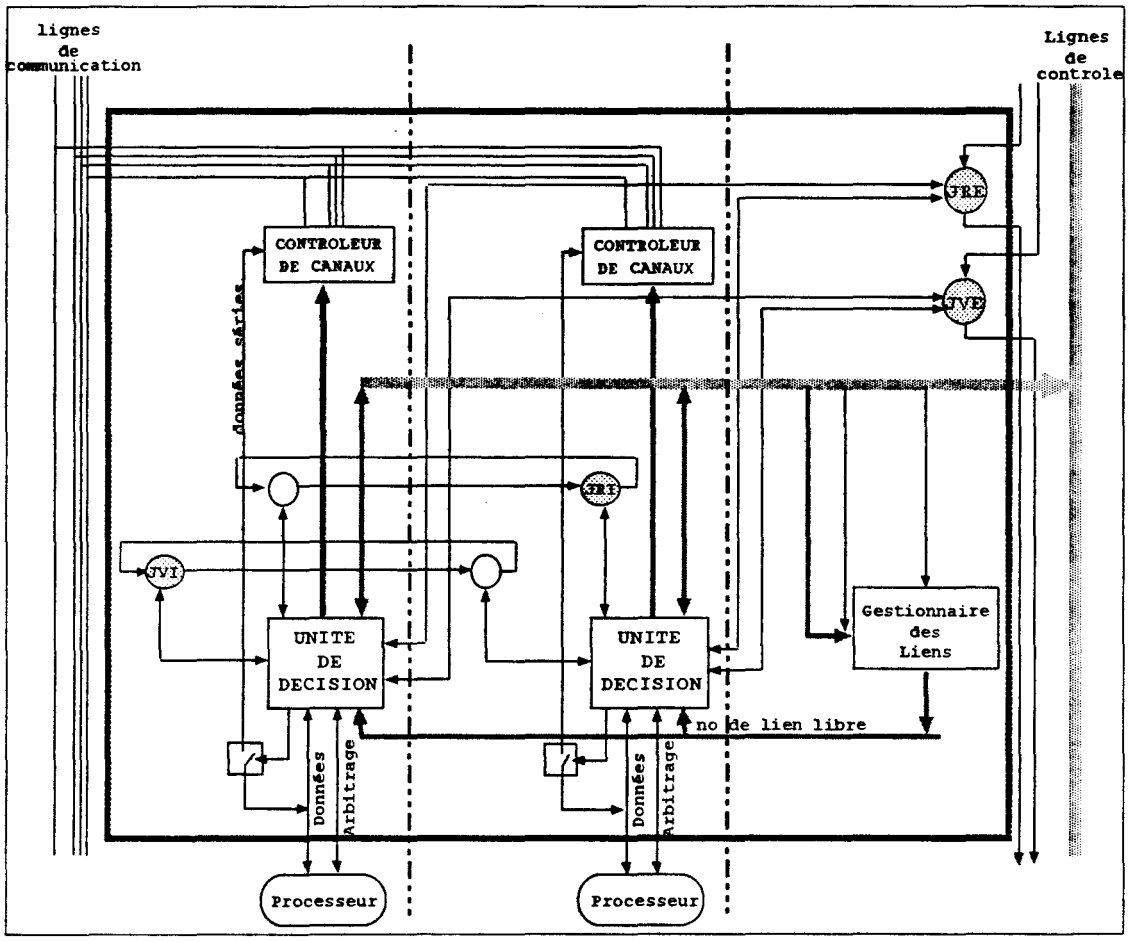

Figure 5.1 : Schéma partiel d'un MCB

- d'un ensemble d'unités de décision,
- d'une unité d'arbitrage, qui est constitué de 2 réseaux à jeton interne et une partie des réseaux à jeton externe. Les jetons **JRI** (Jeton de Requête Interne) et **JVI** (Jeton de Voie Interne) circulent dans les réseaux à jeton interne, et les jetons **JRE** (Jeton de Requête Externe) et **JVE** (Jeton de Voie Externe) circulent dans les réseaux externes,
- d'un ensemble de contrôleurs de canaux,
- d'un gestionnaire des liens.

La représentation du MCB montre bien l'approche modulaire adoptée. Mis à part le gestionnaire des liens, le MCB est construit par assemblage de plusieurs unités

identiques. Nous allons, dans ce qui suit, présenter d'un point de vue réalisation les différentes unités du MCB.

5.2 Le contrôleur de canaux

Le contrôleur de canaux est l'unité permettant au processeur d'accéder au réseau de communication. Chaque processeur peut matériellement se connecter sur l'un des 64 liens du MCB. Deux approches sont possibles pour réaliser le contrôleur : l'approche centralisée ou distribuée.

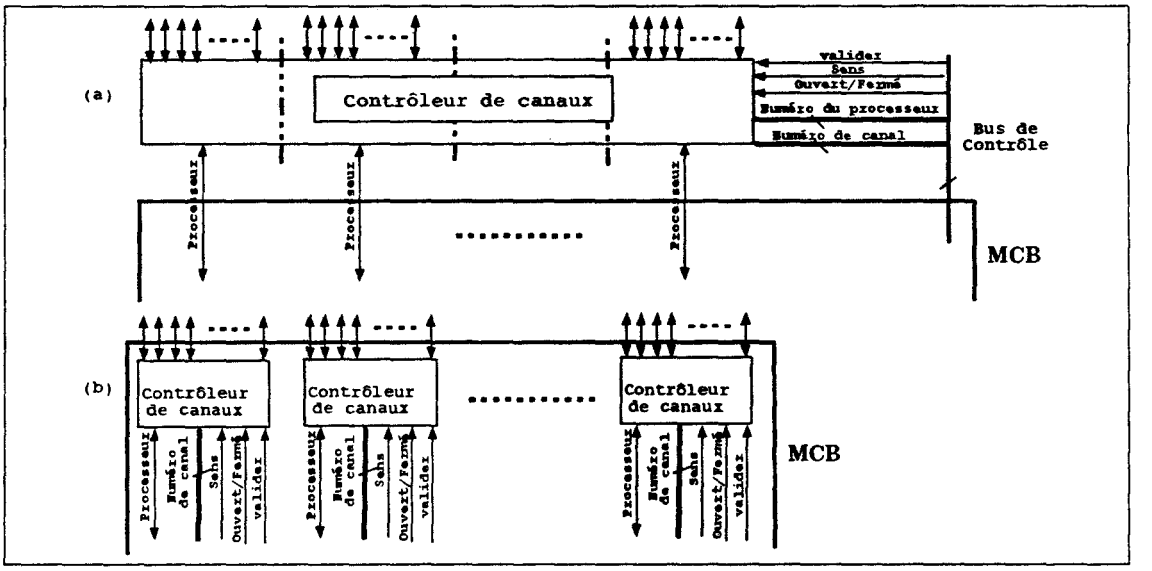

Figure 5.2 : Le contrôleur de canaux centralisé (a) et distribué (b)

5.2.1 L'approche centralisée

L'approche centralisée (figure 5.2.a) réalise une économie importante en termes de lignes de contrôle, mais ne permet pas d'effectuer des opérations simultanément. De plus, les lignes du bus de contrôle global, qui doivent interconnecter toutes les UD du module au contrôleur de canaux, traversent tout le circuit. La longueur de ces lignes est importante dans notre cas et devient pénalisante dans le cadre d'une intégration VLSI.

Néanmoins, cette approche présente un intérêt : si le réseau de communication et les unités de décision sont intégrés dans des boîtiers différents, alors le gain en nombre de lignes permet de réduire le nombre de broches nécessaires. En effet, le

nombre de broches nécessaires est égal à la somme de $M + Log_2V + Log_2M + 3$, avec M les lignes de données venant des processeurs du module, Log_2V les lignes de sélection de la voie, Log_2M les lignes de sélection du processeur et les lignes *sens*, *Ouvert/Fermé* et *valider*. Pour $M = 8$ et $V = 64$, le nombre de lignes nécessaires est de 20 au lieu de 64, puisque dans l'approche distribuée le MCB doit être connectée aux 64 liens du réseau.

Comme le contrôleur de canaux est séparé du MCB, il est possible de connecter des contrôleurs de canaux de tailles différentes au MCB, tant que leur taille reste inférieure à V. Dans une approche à architecture extensible, cette propriété permet d'avoir un réseau de communication extensible en nombre de liens. De ce fait, la taille du réseau peut être modifiée en fonction du nombre de processeurs. L'architecture obtenue est alors parfaitement extensible linéairement.

Cette approche n'a pas été adoptée puisqu'elle nécessite, d'une part, une couche supplémentaire pour gérer les accès aux lignes de contrôle, ce qui ralentit le temps d'accès au réseau. Et, d'autre part, bien qu'il soit possible d'obtenir un réseau parfaitement extensible, on peut s'interroger sur l'intérêt d'une telle approche, du fait du phénomène de saturation en contrôle qui fait que le grain de traitement dépend du nombre de processeurs. Enfin, la longueur des liaisons introduit des difficultés de réalisation ayant pour conséquence directe la diminution de la bande passante.

5.2.2 L'approche distribuée

L'approche distribuée (figure 5.2.b) a été adoptée pour réaliser la commande du réseau de communication. Cette solution est plus rapide, puisqu'elle n'ajoute pas de couche supplémentaire entre l'UD et le contrôleur. Chaque UD dispose d'un contrôleur de communication privé, ainsi, chaque UD peut effectuer des opérations de connexion ou de déconnexion au réseau de communication indépendamment des autres UD. Ces opérations peuvent se faire en parallèle, ce qui est très utile dans le cas des schémas de communication du type diffusion partielle ou totale.

Du point de vue intégration en VLSI, le contrôleur de communication est dans ce cas fonctionnellement indépendant des autres contrôleurs, il dépend uniquement du réseau et de son unité de décision. Cette propriété permet, à la fois, de faciliter son placement et d'éviter des connexions trop longues entre les unités.

5.2.3 Réalisation

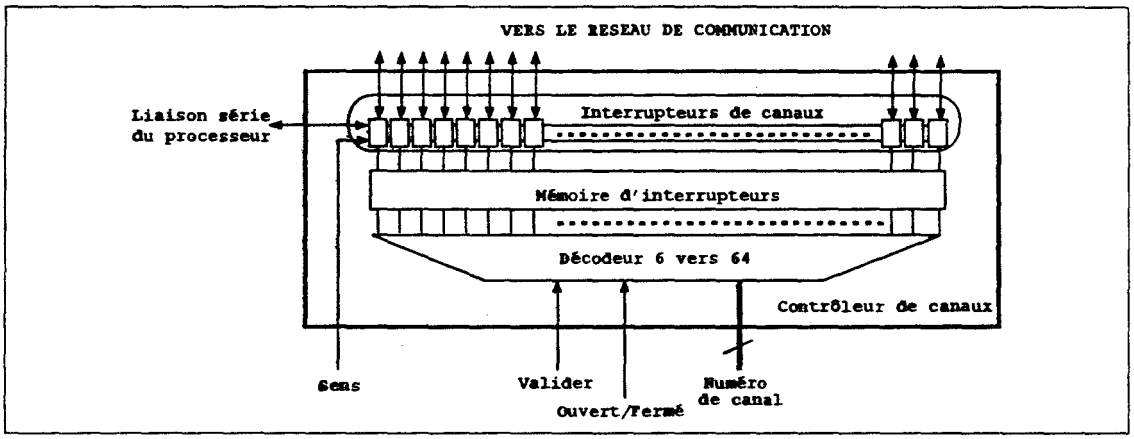

Figure 5.3 : Synoptique du contrôleur de canaux

Le contrôleur de canaux (figure 5.3) est composé d'un décodeur 6 vers 64, d'une mémoire d'interrupteurs et de 64 interrupteurs. La ligne *Valider* sert à valider le décodeur 6 vers 64. Le numéro de canal et la commande *Ouvert/Fermé* servent à mettre à jour une des composantes de la mémoire d'interrupteurs. Cette mémoire constitue la commande des interrupteurs qui sont du type bidirectionnel, et le sens du transfert est commandé par la ligne *sens*.

5.3 Communication entre Pe et UD

La communication entre Pe et UD doit se faire en utilisant un minimum de lignes physiques. On rappelle, ici, qu'un des objectifs du projet est d'intégrer un MCB dans un boîtier PGA de 144 broches. 64 broches sont utilisées pour le réseau de communication, ce qui laisse 80 broches pour les lignes de contrôle (bus de contrôle, réseaux à jeton), les lignes de communication entre Pe et UD et les lignes diverses (Vcc, Gnd, ...etc).

Une solution simple consiste à utiliser 3 lignes par couple (Pe, UD): la ligne Pe , la ligne UD et la ligne données série. Cette solution, utilisée dans la plupart des systèmes, est trop coûteuse dans notre cas, car l'interconnexion de 16 Pe dans le même MCB nécessite 48 broches! Nous utiliserons ici une solution développée dans [DS87a] qui n'utilise que 2 lignes pour la communication entre Pe et UD

5.3.1 Protocole de communication

Le processeur et l'UD communiquent par une ligne série bidirectionnelle. Un jeton sert à contrôler l'accès à cette ligne. Le protocole de passage de ce jeton utilise une seule ligne d'arbitrage. A un moment donné, un des 2 partenaires, par exemple le processeur, possède le jeton et devient donc le maître de la ligne de données. Pour demander la ligne de données, l'unité de contrôle de l'UD met à un la ligne d'arbitrage, le processeur accepte en la mettant à zéro, le jeton est alors envoyé vers le 'UD et les deux partenaires changent de rôle. L'UD restera en attente, tant que le processeur ne remettra pas à zéro la ligne.

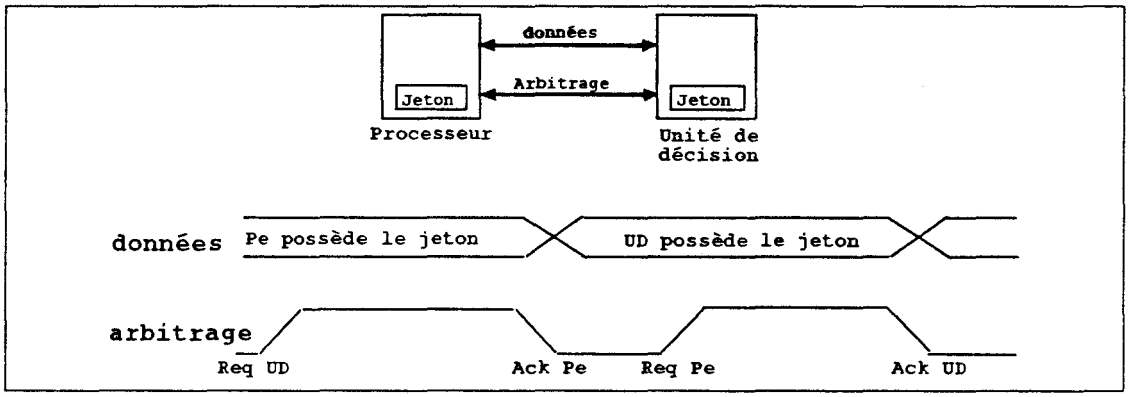

Figure 5.4 : Protocole de communication entre le processeur et son unité de décision

5.3.2 Communication entre les Pe

Comme nous venons de le voir, chaque Pe est connecté au MCB par 2 lignes (données et arbitrage). La ligne de données sert pour communiquer, à la fois, avec l'UD associée et le processeur destinataire. Le protocole de communication entre Pe et UD adopté impose que l'UD doit rester constamment à l'écoute, même lorsque la liaison entre les 2 Pe est établie. Car, une fois la liaison établie, le Pe (maître de la ligne de données) ne peut plus envoyer de message (notamment la requête de fin de connexion) vers l'UD, si celle-ci n'est pas à l'écoute.

Le commutateur est commandé par l'UD (figure 5.5). Une fois la requête satisfaite, l'UD ferme le commutateur de façon à laisser passer les informations vers le processeur destinataire. La fin de la communication est marquée par un message particulier (message de fin). Dès que l'UD voit passer ce message, elle arrête la communication en ouvrant le commutateur.

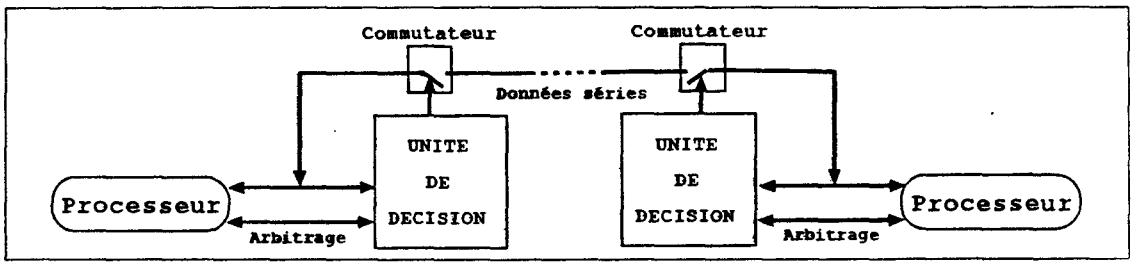

Figure 5.5 : Schéma de communication entre les processeur

5.3.3 Format du message

La réception du message nécessite une conversion série/parallèle. Le message se compose de 3 champs :

Entête du message	Données	Fin du message

5.3.3.1 L'entête du message

Le protocole de communication entre Pe n'étant pas encore parfaitement défini. Dans le format actuel du message, l'entête ne contient que la nature du message qui est de 2 types :

Les messages du processeur vers l'unité de décision sont :

- Requête de communication vers un processeur.
- Requête de fin de communication.

Les messages de l'unité de décision vers le processeur sont :

- Liaison établie en réception.
- Liaison établie en expédition.
- Destinataire occupé.

5.3.3.2 Données

Le format du champ "données" dépend de la nature du message. Si le message est une requête de communication vers un processeur, le champ données contient

alors l'adresse de ce processeur. Si le message est une information venant de l'unité de décision, alors le premier champ du message suffit (le champ données est vide).

5.3.3.3 Format pour la diffusion

Dans le modèle original, seul le mode de communication un vers un était prévu. Il nous semble judicieux de pouvoir effectuer des communications mettant en jeu non plus un seul destinataire mais un sous-ensemble de k processeurs avec $K \leq N$.

Une solution triviale consiste à mettre dans le champ données la liste de tous les processeurs que l'on désire atteindre. Cette méthode permet d'accéder à n'importe lequel des processeurs mais présente plusieurs inconvénients :

- elle ralentit considérablement la communication entre le Pe et l'UD, à cause de la liaison série entre ces derniers.
- l'UD doit disposer d'une zone importante de mémoire pour sauvegarder tous les numéros des processeurs destinataires.
- un traitement de requête de communication globale introduit une charge plus importante pour le bus de contrôle, puisqu'il nécessite non plus 2 messages, mais autant de messages que de processeurs à adresser.

La solution adoptée consiste à "élargir" le champ *Entête du message* qui est de 2 bits pour coder les différents messages de l'UD vers le Pe. En plus des messages de Pe vers l'UD existants, qui ne concernent que des communications de type un vers un, nous ajoutons des fonctions supplémentaires :

1. mode d'adressage par masque,
2. mode d'adressage par pas.

5.3.3.3.1 Le mode d'adressage par masque permet de sélectionner un sous-ensemble de processeurs en fonction de la valeur contenue dans le champ *données*. Ce dernier ne contient pas le numéro du processeur destinataire, mais un ensemble de masques. Un masque, associé à chaque bit du numéro du processeur, comporte trois états : 0, 1 et M. Le masque à 0 (resp. 1) permet de mettre le

bit associé à 0 (resp. 1) ; un masque à M indique que le bit associé peut prendre les valeurs 0 et 1. Le résultat du traitement de ce masque donne l'ensemble des numéros des processeurs adressés. Soit un masque de valeur 001M0M11, son traitement produit l'ensemble des numéros {00100011, 00100111, 00110011, 00110111}. L'adressage par masque nécessite 2 données pour coder le masque. Donc, un message supplémentaire à transmettre permettra en contre partie :

- d'effectuer la diffusion en mettant tous les bits du masque à M,
- d'adresser les voisins du processeur demandeur,
- d'adresser des processeurs d'un même module,
- d'adresser plusieurs groupes de processeurs.

5.3.3.3.2 Le mode d'adressage par pas permet d'adresser des processeurs dont le numéro appartient à l'ensemble $\{a/a = n + kp, \forall k = 1, \ldots, \lceil \frac{N}{p} \rceil\}$ avec n le numéro du processeur demandeur et p la valeur du pas.

5.4 Le Gestionnaire des liens

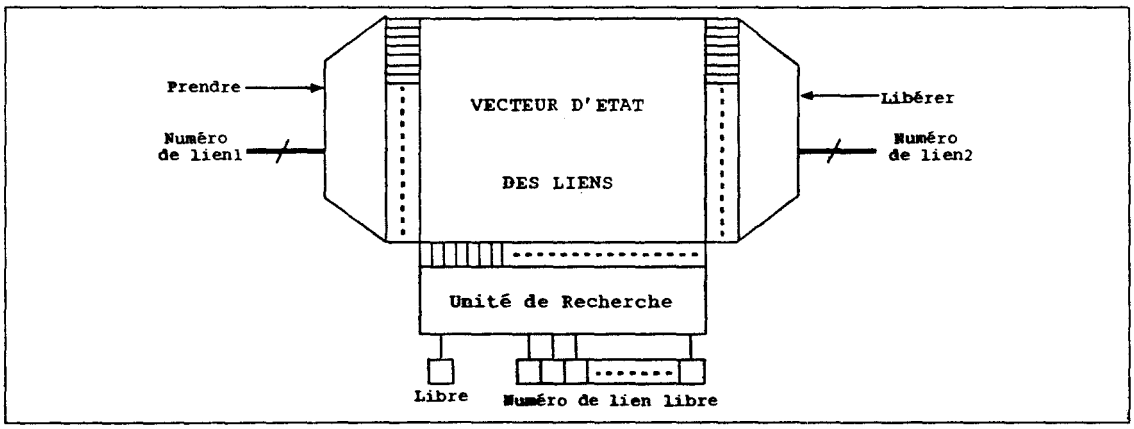

Figure 5.6 : Vue générale d'un gestionnaire des liens

Le gestionnaire des liens mémorise l'état (libre ou occupé) de tous les liens de communication du réseau dans son vecteur d'état. Chaque MCB dispose d'un gestionnaire des liens dont le contenu du vecteur d'état est identique.

Le bus *Numéro de lien1* et la ligne *Prendre* permettent de mettre à l'état occupé la composante du vecteur adressée par le bus d'acquisition. De la même manière,

le bus *Numéro de lien2* et la ligne *Libérer* permettent de libérer la composante du vecteur adressée par le bus de libération. Ces 2 entrées autorisent une mise à jour en parallèle du vecteur. Les valeurs présentes sur les 2 bus sont toujours différentes, puisqu'un même lien ne peut être pris et libéré en même temps. Cela évite le problème d'un conflit d'accès à la même composante du vecteur.

Une ligne d'état *libre* indique s'il reste encore un lien disponible. Le gestionnaire présente en sortie un registre contenant le numéro d'un lien libre déterminé par l'unité de recherche. L'algorithme, implémenté dans cette unité, recherche le plus petit numéro de lien libre parmi l'ensemble des liens disponibles.

En cas de panne d'un lien de communication, la composante indiquant l'état du lien sera mise à l'état occupé par un processeur du réseau. Ainsi, le lien sera rendu indisponible pour tous les processeurs. La méthode utilisée de tolérance aux pannes présente l'avantage d'être très simple à implémenter, mais une panne d'un commutateur rend systématiquement le lien inutilisable. Des algorithmes supportant un certain nombre de pannes de commutateurs sont proposés dans [VC92]. En cas de panne d'un commutateur, le lien est réutilisé sur sa portion valide. Cette approche exploite au mieux le réseau de communication et conserve la propriété d'absence de blocage du crossbar CCM pour un nombre de pannes inférieur à un seuil, mais nécessite la mémorisation d'un nombre important d'informations de contrôle.

5.5 Le réseau à jeton

Un réseau à jeton est utilisé pour arbitrer les requêtes d'accès au bus de contrôle. Un jeton unique, symbolisant l'autorisation d'accès au bus, circule dans ce réseau. Pour réduire le temps d'attente du jeton, une solution cablée est nécessaire, et 2 solutions sont possibles pour la réalisation du réseau à jeton : les solutions synchrone et asynchrone.

La solution synchrone, plus simple, (figure 5.7.a) consiste à faire circuler un jeton synchrone sur un anneau de bascules. Le jeton est transmis à chaque impulsion de l'horloge d'une bascule à l'autre. L'UD_i demandeur de jeton met la bascule associée R_{jet} à un. Ainsi, lorsque le jeton arrivera à la $bascule_i$, il sera conservé par l'UD_i le temps d'effectuer son traitement. Cependant, la contrainte de rapidité nécessite l'utilisation d'une horloge supplémentaire plus rapide, et l'utilisation d'une horloge globale pose des problèmes de synchronisation pour le réseau à jeton externe.

Nous avons adopté une solution asynchrone, (figure 5.7.b), qui consiste à faire circuler une impulsion sur un anneau de cellules logiques. Lorsque l'impulsion

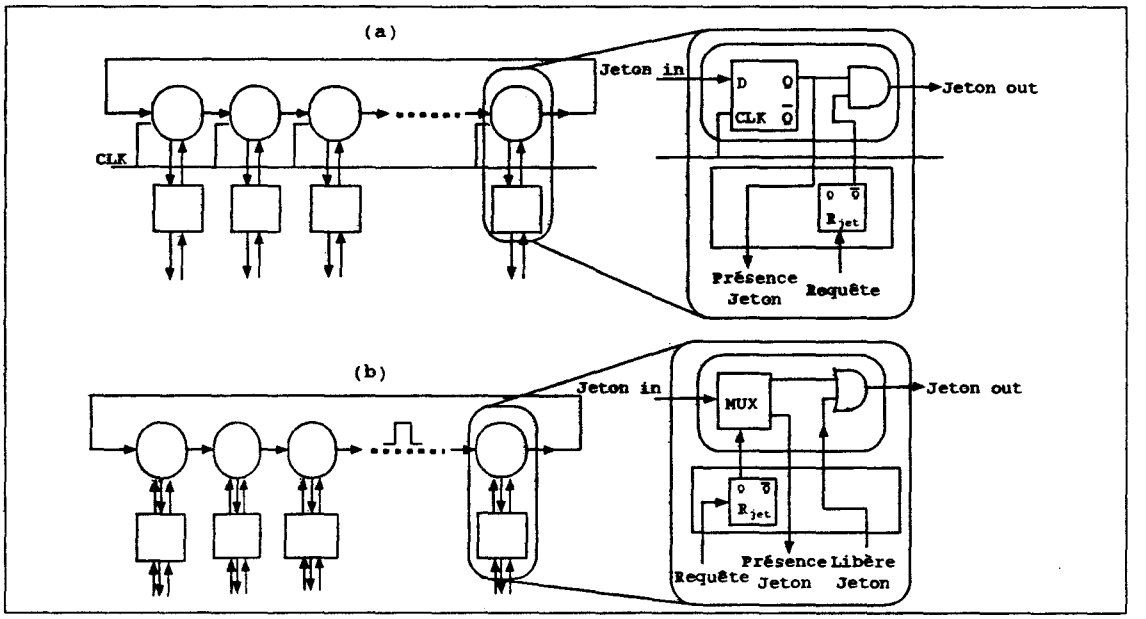

Figure 5.7 : Synoptiques du réseau à jeton : solution synchrone (a), solution asynchrone (b)

arrive sur une cellule demandeur, elle est "consommée" par la cellule et l'UD est alors informée de la présence du jeton ; la libération du jeton nécessite la génération d'une nouvelle impulsion.

Par cette solution, nous obtenons un temps de traversée de la cellule meilleur que dans la solution précédente, puisqu'il correspond au temps de traversée de 2 portes logiques de base. Par ailleurs, l'absence d'horloge globale permet une circulation du jeton (l'impulsion) avec une vitesse maximale.

En simulation, sous Solo1400 en technologie 1.5 microns, le temps de traversée d'une cellule est de 5 ns et le temps de startup du jeton, c'est à dire le temps nécessaire pour lancer l'impulsion, est de 14.75ns (voir annexe).

5.6 L'unité de décision

L'unité de décision constitue l'élément principal du module de communication de base. Une UD, associée à un processeur, prend en charge toutes les requêtes de communication (début, fin) émises par celui-ci. Un protocole de communication entre les UD a été défini. Il permet aux UD d'échanger des informations nécessaires pour le traitement de leurs requêtes d'acquisition ou de libération de liens de communication, pour la résolution des conflits d'accès à un même processeur

et aux liens disponibles.

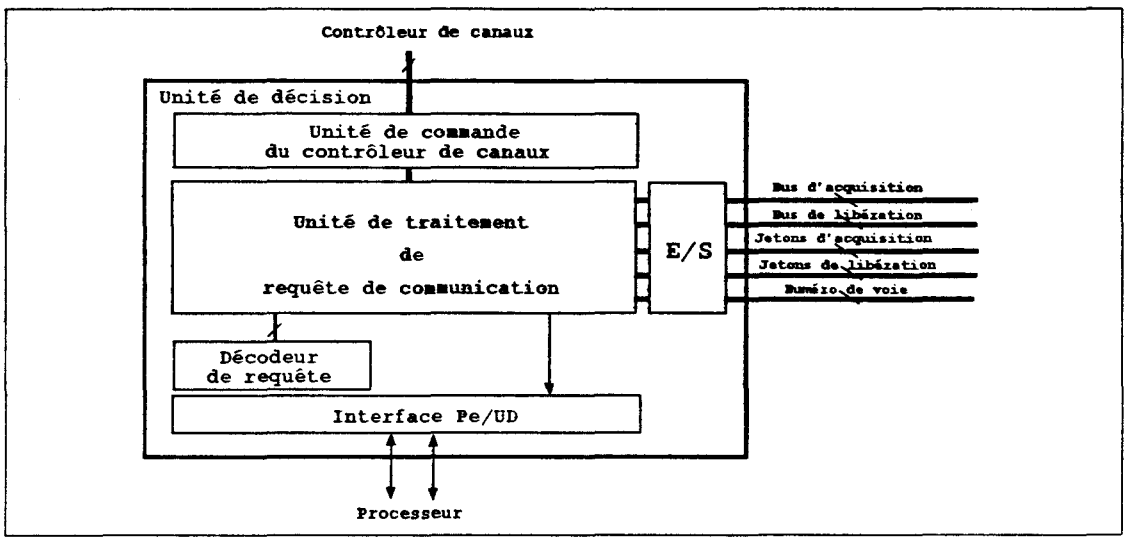

Figure 5.8 : Synoptique de l'unité de décision

L'unité de décision est composée de plusieurs parties :

- Une interface Pe/UD,
- Un décodeur de requête,
- Une unité de traitement de la requête de communication,
- Une unité de commande du contrôleur de canaux,
- Un organe d'E/S.

5.6.1 Interface Pe/UD

Afin de ne pas se limiter sur le choix du processeur, l'unité de décision n'a pas été conçue pour un type de processeur donné, le rôle de l'interface Pe/UD est d'adapter l'UD à un processeur donné. Ce choix ajoute une couche supplémentaire, mais permet d'interconnecter des éléments de nature différente sur le réseau. En plus des processeurs, il est envisageable, par une telle approche, d'associer également des organes d'E/S et des mémoires.

Dans le §5.3, nous avons défini le protocole de communication entre le Pe et l'UD et nous avons vu que la liaison entre le Pe et l'UD est du type série bidirectionnelle. L'interface Pe/UD est constituée d'une interface série parallèle qui convertit la commande venant du processeur sous forme série en une commande parallèle, et d'une interface parallèle série qui permet à l'UD de retourner des messages au

processeur. Comme le choix du processeur n'a pas été défini dans le cadre de ce travail, la taille du message de commande n'est donc que provisoire.

5.6.2 Organe d'E/S

L'organe d'E/S sert d'intermédiaire entre les UD pour recevoir et envoyer des informations. Une UD est connectée sur un bus global et sur un bus local.

Le bus global se compose d'un bus de contrôle d'acquisition (**BCA**) de 8 bits (pour adresser les 256 processeurs) sur lequel transitent toutes les requêtes de communication, d'un bus de contrôle de libération (**BCL**) de 6 bits (pour les 64 liens de communication) sur lequel transitent toutes les requêtes de libération et des 4 lignes de contrôle suivantes :

- **BAU** : une ligne d'état Bus en cours d'Utilisation qui informe les UD que le BCA est occupé par une UD. Cette ligne est utilisée pour permettre d'anticiper le passage du jeton de requête externe (§3.7.2).
- **BLU** : une ligne d'état Bus en cours d'Utilisation qui informe les UD que le BCL est occupé par une UD. Cette ligne permet d'anticiper le passage du jeton de voie externe.
- **ACK/NACK** : cette ligne est utilisée par l'UD destinataire de la requête de communication présent sur le BCA, pour déposer sa réponse (Acknowledge ou No Acknowledge).
- **SRA** : un Signal Réponse Ack/nack qui informe les UD que la ligne *Ack/Nack* est occupée par une UD.

Le bus local se compose d'un Bus Voie Libre **BVL** de 6 bits, sur lequel transite le numéro de voie libre issu du gestionnaire des liens du module, d'une ligne d'état **Libre** qui indique s'il existe au moins un lien libre et des lignes de gestion des jetons internes et externes.

L'organe d'E/S dispose également d'un comparateur de liens et un décodeur du processeur destinataire. Le comparateur de liens compare le numéro du lien présent sur le BCL avec le numéro du lien que dispose l'UD et envoie le résultat de la comparaison vers l'UD. Actuellement, le décodeur de processeur destinataire n'accepte que des requêtes de communication un vers un. Il est envisagé d'apporter des modifications pour accepter des requêtes de communication globale définies dans le §5.3.3.3.

5.6.3 Unité de traitement de requêtes de communication

L'Unité de Traitement (**UT**) sert d'intermédiaire pour le traitement des requêtes de communication de son processeur. Pour cela, elle réalise le protocole de communication décrit dans §3.7.3. Les données nécessaires au protocole sont reçues par l'intermédiaire de l'interface Pe/UD et de l'organe d'E/S. Ces données permettent à l'UT de traiter les requêtes de communication. Elle constitue l'élément de prise de décision et de commande de l'UD. Nous allons, dans ce qui suit, détailler l'unité de traitement.

5.7 L'unité de traitement

Le fonctionnement de l'UT est décrit par un automate à 6 états. La description de chaque état est la suivante :

5.7.1 Automate d'états de l'unité de traitement

Etat 0 : L'UD est libre : dans cet état, elle est en attente de requêtes. Deux types de requêtes peuvent lui parvenir : une demande de communication provenant de son processeur ou une requête venant de l'organe d'E/S qui a reconnu sur le BCA le numéro du processeur, c'est à dire qu'elle est demandée en destinataire.

Dans le premier cas, l'UD mémorise la requête dans un registre d'état interne **DC** (Demande de Communication) ainsi que le numéro du processeur destinataire, puis passe dans l'état 2 pour effectuer la demande de communication de son Pe. Elle passe dans l'état 1, dans l'autre cas, pour accepter la demande de communication d'un autre Pe.

Etat 2 : L'UD, ayant reçu une demande de communication (Registre DC à 1) de son processeur, cherche à accéder au BCA. Pour cela, elle demande dans un premier temps le jeton de requête interne (**JRI**) en positionnant la bascule R_{jri} à 1, puis, une fois le JRI obtenu, elle demande le jeton de requête externe (**JRE**) en positionnant le bascule R_{jre} à 1.

Lorsque l'UD a obtenu les 2 jetons, elle doit attendre que le BCA soit libre (ligne d'état BAU à zéro) avant de passer dans l'état 3.

Avant que l'UD n'obtienne les 2 jetons, elle peut être, à tout moment, demandée en destinataire. Dans ce cas, elle passe dans l'état 1.

Etat 3 : L'UD met à un la ligne d'état BAU pour devenir le maître de BCA et libère les deux jetons de requête. Ensuite, elle dépose sur le BCA le numéro du processeur destinataire et se met en attente de la réponse du destinataire.

En attendant, elle lit sur BVL le numéro de lien libre dès qu'il existe un lien libre (la ligne d'état *libre* à un). Quelle que soit la réponse retournée par le destinataire, une fois la réponse reçue, l'UD met la ligne SRA à zéro pour indiquer au destinataire que la réponse a été reçue.

Si la réponse est un Ack, l'UD libère le BCA en remettant à zéro la ligne d'état BAU, informe le processeur qu'il est pris en destinataire, envoie le numéro de voie libre à l'unité de commande du contrôleur de canaux, puis passe dans l'état 4.

Si la réponse est un Nack, alors l'UD libère le BCA en remettant à zéro la ligne d'état BAU et retourne dans l'état 2.

Etat 4 : Dans cet état, le processeur est en communication. L'UD espionne la communication de son processeur en attente du message de fin de communication. Une fois le message de fin reconnu, elle passe dans l'état 5.

Pendant tout le temps de la communication, si l'UD reçoit une requête de communication lui demandant d'être le destinataire (c'est à dire si elle voit son propre numéro sur le BCA), elle envoie Nack sur la ligne de réponse, et positionne la ligne SRA à 1 pour valider la réponse.

Etat 5 : C'est un état dual de l'état 2. L'UD doit dans un premier temps acquérir le jeton de voie interne (**JVI**) et le jeton de voie externe (**JVE**), puis attendre que la ligne BLU soit à zéro pour accéder au BCL.

Si la ligne BLU est à zéro, alors l'UD accède au BCL en mettant à un BLU, puis elle dépose sur le BCL le numéro de lien qu'elle a occupé, se déconnecte, libère les jetons de voie et retourne à l'état 0.
Avant qu'elle n'obtienne les 2 jetons de voie, elle peut être demandée en destinataire ; dans ce cas, elle envoie Nack sur la ligne de réponse, et positionne la ligne SRA à 1 pour valider la réponse.
Ou alors, son organe d'E/S peut lui signaler que le numéro de lien présent sur le BCL est le même que celui qu'elle possède, cela signifie que le message est déposé par son partenaire de communication. Dans ce cas, elle remet à zéro la ligne BLU, se déconnecte, rend le JVI (si elle le possède) et retourne dans l'état 0.

Etat 1 : L'UD libère le JRI qu'elle possède éventuellement. Elle vérifie qu'il existe un lien libre (ligne d'état libre à un). Dans ce cas, elle lit le numéro du lien libre sur le BVL pour l'envoyer à l'unité de commande du contrôleur de canaux et envoie un Ack sur la ligne réponse tout en mettant la ligne d'état SRA à un pour valider la réponse. Puis, elle informe le processeur qu'elle est prise en destinataire et passe, enfin, à l'état 5. Par contre, s'il n'existe pas de lien libre, l'UD reste en attente jusqu'à ce qu'un lien se libère, puis effectue le traitement décrit au début de cet état.

L'automate de l'UT est représenté sur la figure 5.9. Les A_i sont des commandes dont les descriptions sont données dans le §5.7.2.1.

Deux approches sont possibles pour la réalisation de l'automate, micro-programmée ou cablée. Nous avons préféré la solution cablée, car elle offre la possibilité d'exécuter plusieurs actions simultanément, et a un temps de réponse meilleur que la solution micro-programmée.

L'ensemble fonctionne de manière asynchrone, c'est à dire que le traitement et l'exécution des actions ne sont pas cadencés par une horloge globale. En effet, l'UT effectue des traitements de nature totalement différente suivant son état, comme, par exemple, une communication sur le bus ou une demande de jeton. La durée de ces traitements est très différente, elle varie de quelques dizaines de nanosecondes pour une communication sur le bus à quelques nanosecondes pour une demande de jeton. Les UT ne se trouvant pas forcément dans le même état, l'utilisation d'une horloge globale imposerait une cadence régulière des traitements qui alignerait la durée de tous les traitements sur la durée du traitement le plus long.

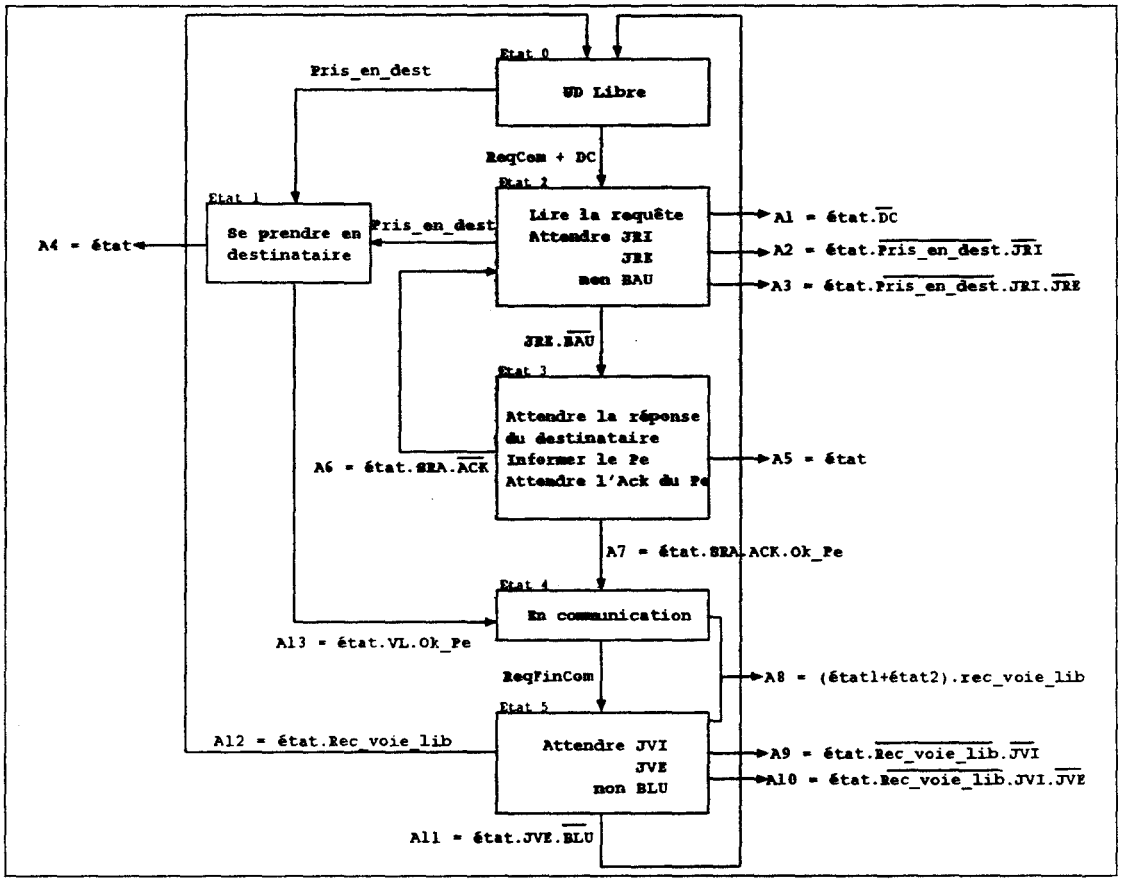

Figure 5.9 : L'automate décrivant le protocole de communication de l'unité de décision

5.7.2 Réalisation de l'automate d'états

L'automate d'états de l'unité de traitement comporte en tout 6 états. 3 bascules sont donc nécessaires pour les coder. La figure 5.10 représente le schéma logique avec ses 3 bascules et les données d'entrées qui constituent les conditions de changement d'états.

L'automate fonctionne de manière asynchrone afin d'obtenir un meilleur temps de réaction. La partie gauche du schéma représente le circuit de déclenchement des bascules d'états, dont le tableau de contrôle est donnée par le tableau 5.1. Au centre, nous trouvons les 3 bascules RS servant à coder les états de l'automate. Et, enfin, la partie droite représente un décodeur des états qui permet de générer les signaux de e_0 à e_5 servant à déclencher les actions A_i.

Le fonctionnement global de l'automate a été décrit dans la section précédente.

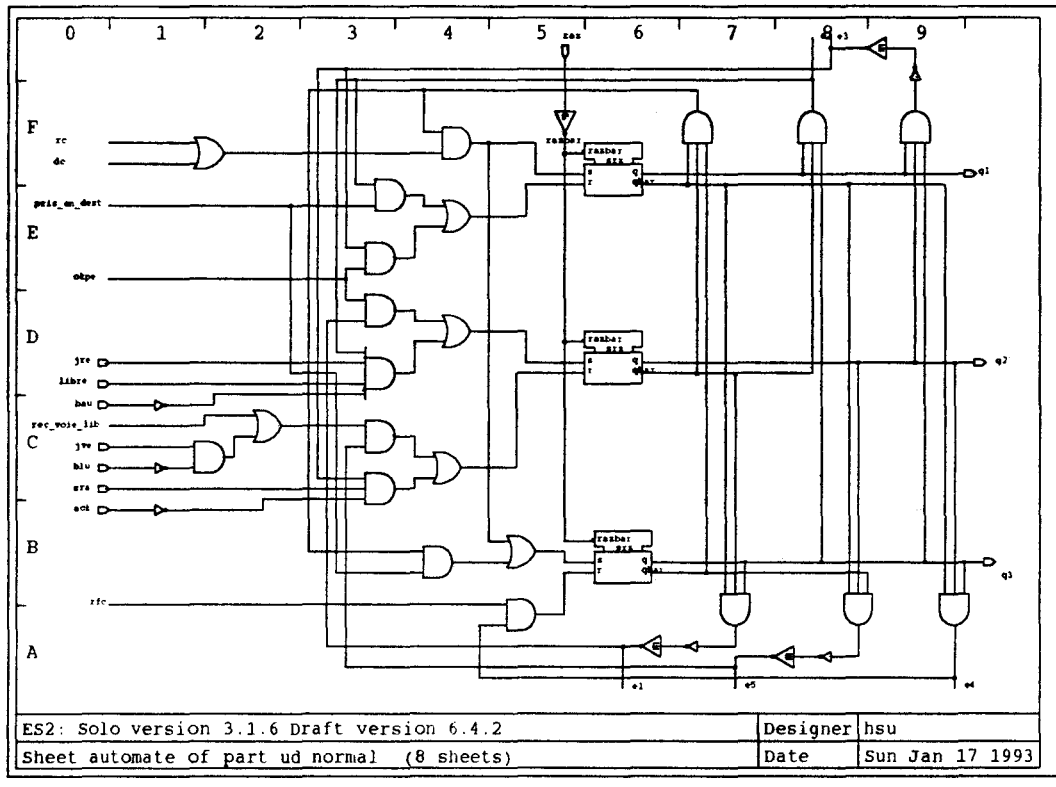

Figure 5.10 : Réalisation de l'automate d'unité de traitement sous Solo1400

Nous allons, dans un premier temps, donner la liste des actions effectuées par l'UT, puis le tableau de contrôle qui nous a permis de réaliser le circuit logique, et nous terminerons par la présentation du chronogramme ainsi que par les temps de réaction de l'automate.

5.7.2.1 Liste des actions effectuées par l'UT

A1 : lire le numéro du processeur destinataire provenant de l'interface Pe/UD.

mémoriser la requête de communication dans le registre DC.

A2 : attendre le jeton de requête interne.

A3 : attendre le jeton de requête externe.

A4 : libérer le jeton de requête interne.
informer le processeurs qu'il est pris en destinataire.

A5 : mette la ligne BAU à un pour devenir le maître du BCA.
libérer les jetons de requête interne et externe.
mettre sur BCA le numéro du processeur destinataire.

A6 : mettre BAU à zéro pour libérer le BCA.
mettre SRA à zéro pour signaler au destinataire que sa réponse a été reçue.

A7 : lire le numéro de voie sur BVL et mémoriser ce numéro.
mettre BAU et SRA à zéro.
mettre le registre DC à zéro, c'est à dire que la requête de communication est satisfaite.
informer le processeur qu'il est pris en expéditeur.
envoyer le numéro de voie à l'unité de commande du contrôleur de canaux pour établir la connexion.

A8 : émettre Nack comme réponse.
mettre la ligne SRA à un pour valider la réponse.

A9 : attendre le jeton de voie interne.

A10 : attendre le jeton de voie externe.

A11 : mettre la ligne BLU à un pour devenir maître du BCL.
libérer les jetons de voie interne et externe.
mettre le numéro de voie à libérer sur le BCL.
envoyer le numéro de voie à l'unité de commande du contrôleur de canaux pour détruire la connexion établie.

A12 : mettre la ligne BLU à zéro pour signaler à l'expéditeur que le numéro de voie a été reçu.
libérer le jeton de voie interne.

A13 : lire le numéro de voie sur BVL et mémoriser ce numéro.
émettre Ack comme réponse.
mettre la ligne SRA à un pour valider la réponse.
envoyer le numéro de voie à l'unité de commande du contrôleur de canaux pour établir la connexion.

5.7.2.2 Le tableau de contrôle de l'automate de l'UT

Nous donnons ici le tableau de contrôle de l'automate de l'UT. Les états de l'automate sont codés par les bascules Q_1, Q_2 et Q_3, et le codage des états utilisé est le code de Gray. Le codage de Gray permet de limiter le nombre de changement d'états des bascules lors du changement d'état de l'automate, afin de réduire les possibilités d'instabilité des bascules.

Tableau 5.1 : Le tableau de contrôle de l'automate de l'unité de traitement

Etat courant			Conditions	Signaux générés	Etat suivant		
Q_3	Q_2	Q_1			Q_3	Q_2	Q_1
0	0	0	rc	A1, A2, A3	1	0	1
0	0	0	dc	A2, A3	1	0	1
0	0	0	pris_en_dest	A4	0	0	1
0	0	1	ok_pe, libre	A13	0	1	1
1	0	1	pris_en_dest	A4	0	0	1
1	0	1	jre, $\overline{bau}$	A5	1	1	1
1	1	1	ok_pe,ack	A7	0	1	1
1	1	1	nack	A6	1	0	1
0	1	1	pris_en_dest	A8	0	1	1
0	1	1	rfc	A9,A10	0	1	0
0	1	0	pris_en_dest	A8	0	1	0
0	1	0	rec_voie_lib	A12	0	0	0
0	1	0	jve, $\overline{blu}$	A11	0	0	0

5.8 Quelques chiffres sur le circuit MCB

Nous donnons, dans cette section, les valeurs des divers paramètres liés au circuit MCB que nous avons défini et réalisé dans le cadre de ce travail. Cette réalisation a pour objectifs :

- d'obtenir des valeurs sur les performances du circuit MCB. A l'aide de ces valeurs, nous pouvons effectuer des simulations avec un plus grand nombre de processeurs, afin d'étudier le comportement du système en fonction des

paramètres N (le nombre de processeurs), V (le nombre de voies), D (le débit de la liaison), P (la puissance du processeur), I (le nombre d'instructions) et L (la longueur du message échangé).

- de vérifier sa faisabilité dans le cadre d'une intégration en VLSI et d'estimer le coût d'un tel circuit.

Le MCB réalisé connecte 4 processeurs à un réseau de crossbar CCM de 64 liens. Il est réalisé sous Solo1400 de ES2.

5.8.1 Le nombre de broches

La technologie utilisée pour sa réalisation est en 1.5 μm. Le boîtier, dans lequel est intégré le MCB, est un boîtier PGA de 144 broches, dont 114 broches ont été effectivement utilisées. L'affectation des broches est la suivante :

- 64 broches pour se connecter au réseau de communication,
- 11 broches pour le bus d'acquisition (BCA+BAU),
- 9 broches pour le bus de libération (BCL+BLU),
- 4 broches servant de ligne de réponse du destinataire (Ack/Nack+SRA),
- 4 broches pour le jeton de requêtes externe,
- 4 broches pour le jeton de voie externe,
- 8 broches pour la communication entre les processeurs et les unités de décision,
- 2 broches pour l'alimentation +5V,
- 5 broches pour la masse,
- 3 lignes de contrôle et d'initialisation.

30 broches ne sont pas utilisées par le circuit. Il serait donc intéressant d'utiliser ces broches libres pour connecter d'autres processeurs au réseau. Ces 30 broches autorisent, dans notre cas, la connexion de 15 processeurs supplémentaires. Mais malheureusement, la surface limite d'intégration en technologie 1.5 μm chez ES2 est de 100 mm^2 et comme le MCB actuel a déjà une surface de 106 mm^2, il n'est donc pas possible d'ajouter des UD supplémentaires. Par contre, en technologie 1.2 μm, la surface maximale admissible est de 200 mm^2 chez ES2, donc, comme la surface du MCB actuel en 1.2 μm est de 65.95 mm^2, il est possible d'ajouter au moins 4 UD au MCB pour connecter des processeurs supplémentaires. Nous obtenons, ainsi, un MCB de 8 processeurs et de 64 liens.

5.8.2 La taille du circuit

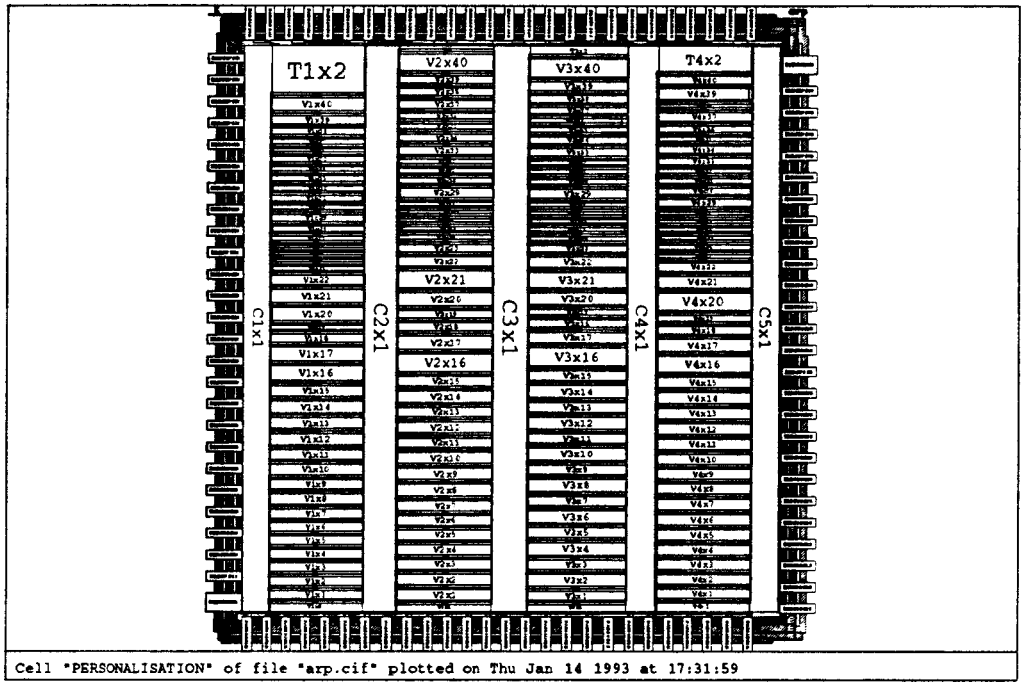

Figure 5.11 : Vue générale du circuit MCB

La figure 5.11 montre une vue du circuit MCB après le placement. Nous voyons, ici, l'approche modulaire adoptée, car le MCB réalisé connecte 4 processeurs au réseau de communication et le circuit dispose de 4 colonnes. Chaque colonne contient une UD, un contrôleur de canaux et une partie de l'unité d'arbitrage, qui sont destinés à un processeur. Quant au gestionnaire de liens, il est réparti dans les 4 colonnes. Cette solution présente l'avantage de limiter les connexions inter-colonnes permettant, ainsi, de réduire l'espace de routage qui est souvent la cause d'une consommation exagérée de surface en silicium. Tous les circuits du MCB ont été placés manuellement ligne par ligne et le gain en surface silicium d'une telle opération est de 36.5%. Nous trouverons, dans l'annexe, une vue du circuit dont le placement a été effectué automatiquement.

Les contrôleurs de canaux, possédant une structure régulière, sont placés dans la partie inférieure de chaque colonne. Les UD sont placées dans la partie supérieure de chaque colonne, et entre les deux, nous avons placé le gestionnaire des liens. Le circuit MCB nécessite 30741 étages, soit 61482 transistors. Quant à sa taille, elle dépend de la technologie utilisée. Le tableau 5.2 donne les dimensions du circuit en fonction de la technologie.

Tableau 5.2 : Les dimensions du circuit MCB en fonction de la technologie

		Technologie	
		1.5 μm	1.2 μm
en	Array area	$8.62 \times 9.2 = 79.31mm^2$	$6.54 \times 6.94 = 45.36mm^2$
	Active chip area	$9.79 \times 10.42 = 101.96mm^2$	$7.69 \times 8.13 = 62.58mm^2$
mm	Die Size	$10 \times 10.63 = 106.24mm^2$	$7.9 \times 8.34 = 65.95mm^2$

5.8.3 Les performances du MCB

Nous fournissons, dans cette section, les temps de réaction des circuits du MCB lors d'une acquisition et d'une libération de voie. L'échelle des temps des chronogrammes, donnée ici, est en nanoseconde. A cause du nombre élevé de signaux, un tri a été effectué, donc seuls les signaux, qui interviennent dans le traitement de la requête, sont représentés.

Les deux chronogrammes donnés ici représentent le traitement d'une requête de communication du processeur 4 vers le processeur 3. Pour simplifier le traitement, nous supposons qu'il existe une voie libre et que le processeur est libre.

5.8.3.1 Acquisition d'une voie

Le chronogramme (figure 5.12) nous donne l'évolution des signaux du MCB au cours d'une demande de communication. Le processeur 4 qui désire effectuer une demande de communication vers le processeur 3 envoie sur l'entrée de l'UD_4 (*ENT_UD(4)*) le message de requête. Lorsque l'UD_4 a reçu la totalité du message (t=1650), elle passe dans l'état 2 (t=1722), et effectue (t=1727) une demande de jeton (*REQ_JRI(4)*). Cette demande lance le jeton de requête interne qui, à l'état repos, se trouve dans l'UD_1. Le jeton interne démarre de l'UD_1 (t=1742) et arrive à l'UD_4 au bout de 15 ns, après avoir franchi 3 places. L'arrivée du JRI lance la demande du jeton externe (*REQ_JRE*) au temps t=1760. Comme le test s'effectue uniquement sur un seul MCB, le jeton externe ne peut être généré que manuellement (t=2150). Lorsque l'UD_4 reçoit le JRE, elle passe à l'état 3 (t=2164), positionne le signal BAU pour devenir maître du bus au temps t=2186, et dépose le numéro du processeur destinataire sur le BCA. L'UD_3 reconnaît son numéro sur le BCA (*PRIS_DEST(3)*) au temps t=2192, elle envoie alors l'ack (t=2207) sur la ligne ACK. Et enfin, au temps t=2223, le canal(0) devient passant et les processeurs peuvent commencer la communication.

Ce qui donne 110 ns pour émettre la requête de jeton externe, une fois la requête

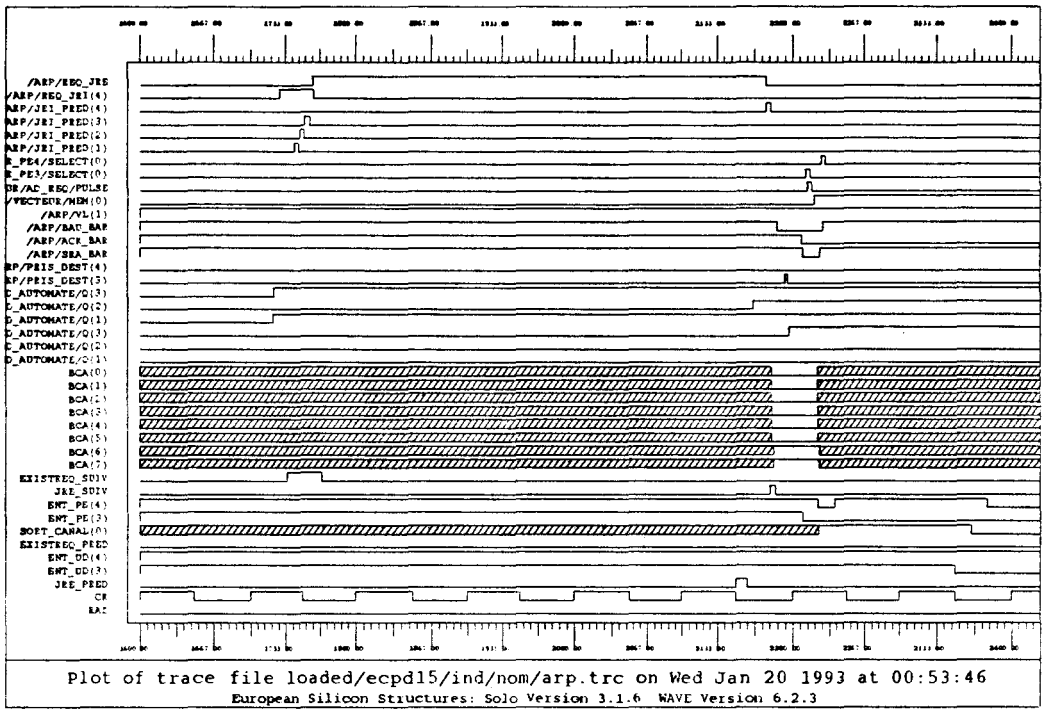

Figure 5.12 : Chronogramme du MCB lors d'une acquisition de voie

de demande de communication reçue, et 73 ns pour établir la communication, une fois le jeton externe reçu.

5.8.3.2 Libération d'une voie

Le chronogramme (figure 5.13) nous donne l'évolution des signaux du MCB au cours d'une libération de voie. Les processeurs 3 et 4, qui désirent libérer la voie, envoient le message de libération vers l'entrée série *ENT_UD* de leur UD. Lorsque les UD ont reçu la totalité du message (t=6850), elles passent dans l'état 5 (t=6921), effectuent (t=6925) une demande de jeton (*REQ_JVI(3,4)*), et se déconnectent (t=6945). Ces demandes lancent le jeton de voie interne qui, à l'état repos, se trouve dans l'UD_1. Le jeton interne démarre de l'UD_1 (t=6938) et arrive à l'UD_3 au bout de 10 ns, après avoir franchi 2 places. L'arrivée du JVI lance la demande du jeton externe (*REQ_JVE*) au temps t=6952. Comme précédemment, le jeton de voie externe est généré manuellement (t=7050). Lorsque l'UD_3 reçoit le JVE, elle passe à l'état 0 (t=7067), positionne le signal BLU pour devenir maître du bus au temps t=7077, et dépose le numéro de voie à libérer sur le BCL. L'UD_4 reconnaît ce numéro sur le BCL (*REC_VOIE(4)*) au temps t=7085, elle remet alors la ligne BLU à zéro (t=7098) marquant ainsi la fin de la libération.

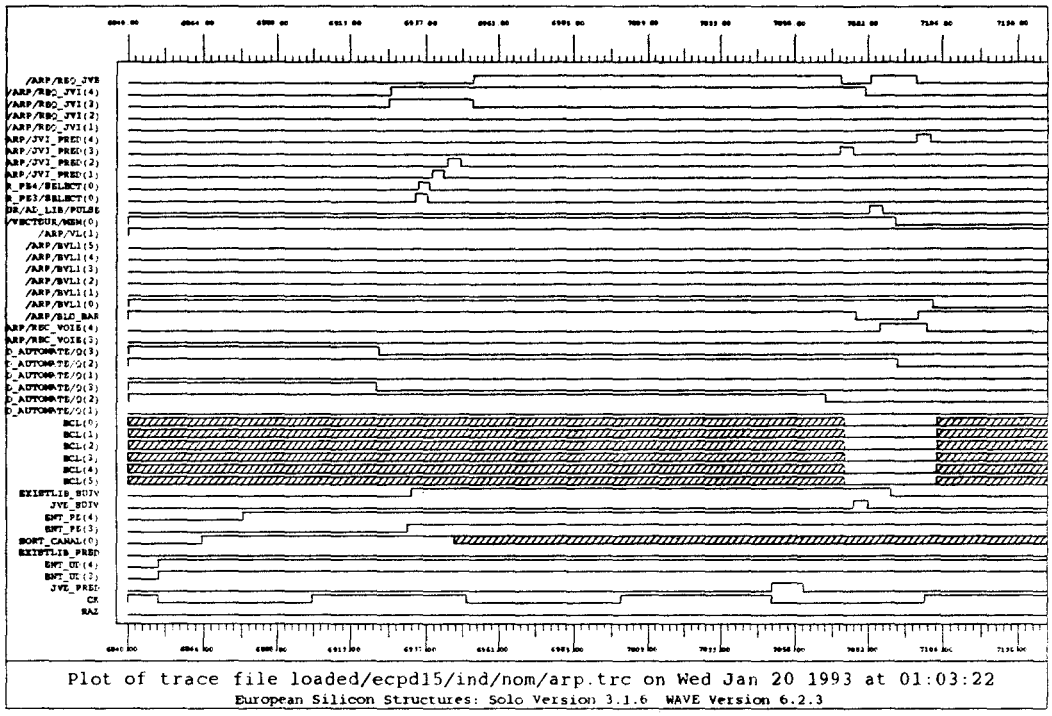

Figure 5.13 : Chronogramme du MCB lors d'une libération de voie

Ce qui donne 102 ns pour émettre la requête de jeton externe, une fois la requête de fin de communication reçue. Contrairement au cas précédent, la voie est physiquement libérée dès que l'UD a reçu le message de libération. La libération est signalée aux autres UDs 48 ns après que l'UD ait reçu le jeton externe.

5.9 Conclusion

La réalisation du MCB en Solo1400 permet de montrer sa faisabilité. En technologie 1.5 μm le MCB réalisé ne permet pas de connecter plus de 4 processeurs pour un réseau de 64 liens. Il est envisagé de réduire la taille du réseau de communication afin d'ajouter des UD supplémentaires au module. Le but est naturellement de faire réaliser le MCB par ES2, de façon à pouvoir effectuer des tests réels de performances. Pour des raisons économiques la technologie 1.2 ne sera pas utilisée. Bien qu'elle démontre parfaitement la faisabilité du MCB puisque en surface de silicium intégrable, elle permet de réaliser un MCB à 8 processeurs et 64 liens.

Les simulations par Solo1400 nous a permis de mesurer les performances du réseau

ARP. Ces résultats ont été utilisés pour effectuer des simulations avec Qnap sur le comportement du système en fonction des paramètres du système.

Chapitre 6

Directions futures

2 février 1993

Nous venons de présenter, du point de vue conception, le projet ARP dans ses grandes lignes. Le projet ARP définit une architecture d'un réseau d'interconnexion pour un multicalculateur, où les processeurs communiquent par échange de messages sur un lien physique direct. Le réseau de communication est à reconfiguration dynamique et asynchrone, c'est à dire que l'établissement et la destruction des liaisons ne sont faits qu'à la demande explicite des processeurs concernés.

Le travail, que nous venons de présenter, se limite dans un premier temps à la présentation du projet et à la définition d'un module de communication de base, puis, à la validation du système à l'aide d'un outil d'aide à la modélisation, et enfin, à la réalisation en VLSI du circuit de base, le MCB, sous Solo1400, permettant de démontrer sa faisabilité. Tout ceci constitue la première "phase" de ce projet. Nous allons, dans ce qui suit, présenter les optimisations et les améliorations futures du MCB, et proposer des idées architecturales utilisant les MCB comme briques de base pour évoluer vers une architecture interconnectant un très grand nombre de processeurs.

6.1 Réduire encore le temps de contrôle

Le temps de contrôle constitue véritablement l'élément clef du réseau ARP. Bien que la solution proposée dans §3.7 présente de bons résultats dans le cas des applications à grains moyens, nous présentons ici une amélioration qui consiste à réduire le nombre de requêtes qui n'aboutissent pas à un succès, afin de réduire la charge effective du bus d'acquisition.

Les résultats de l'évaluation, d'un réseau à 128 processeurs par Qnap, nous a permis de chiffrer le nombre des requêtes traitées par les UD, qui aboutissent à un échec pour cause de destinataire occupé. Ces requêtes retourne alors dans l'UD avec la classe *non expéditeur* (RNCD). Leur nombre permet d'évaluer le temps passé par l'UD à effectuer des traitements inutiles, c'est à dire qui n'aboutissent pas à une communication. Ce nombre augmente et devient souvent prépondérant lorsque le réseau est fortement chargé. La réduction de ce nombre permet à l'UD de traiter des requêtes utiles supplémentaires, par conséquent elle autorise une diminution du rapport $\frac{I}{P}$, donc de T_R (§4.4.4).

Pour réduire le nombre des RNCD, c'est à dire les requêtes qui ont aboutient à un échec pour cause de destinataire occupé, il faut donner la possibilité à l'UD, avant qu'elle ne deviennent maître du bus d'acquisition, de connaître l'état du processeur destinataire sans accéder au bus. Ainsi elle n'accédera au bus de contrôle que si le destinataire est libre.

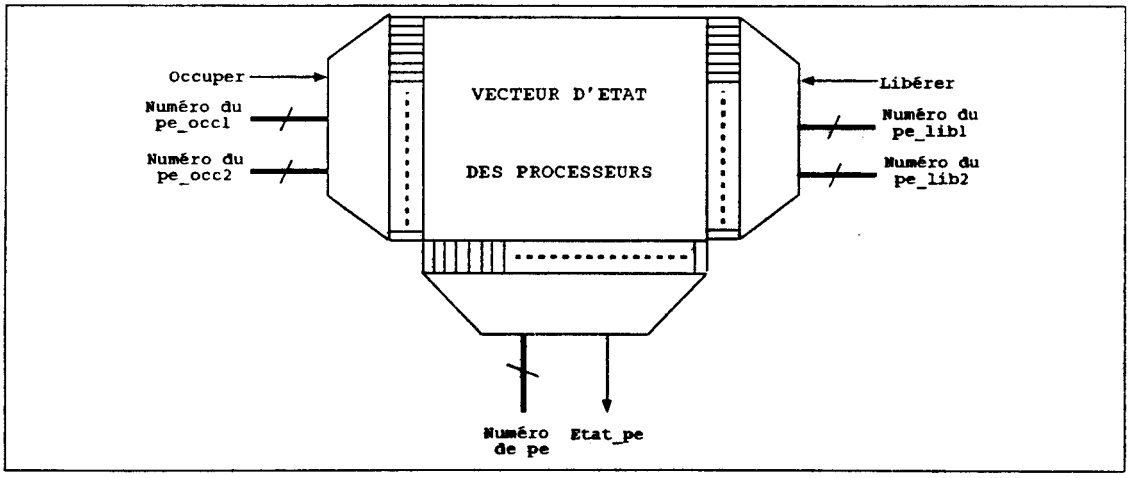

Figure 6.1 : Schéma complet du vecteur d'état des processeurs

La solution proposée consiste à utiliser un vecteur d'état des processeurs dans chaque MCB de manière similaire au vecteur de liens. Une UD ayant une demande de communication à effectuer envoie le numéro du processeur destinataire sur le

bus interne **numéro de pe** et récupère l'état du processeur par la ligne interne **Etat_pe**. Elle n'effectuera de demande que si le processeur destinataire est libre.

Le vecteur d'état des processeurs dispose de quatre bus d'entrées de mise à jour : **numéro pe_occ1** et **numéro pe_occ2** utilisés pour désigner les processeurs qui entrent en communication, **numéro pe_lib1** et **numéro pe_lib2** utilisés pour les processeurs qui terminent la communication. Un processeur ne peut simultanément rentrer en communication et terminer la communication, les numéros présents sur les 4 bus sont toujours différents, l'accès au vecteur ne pose donc pas de problème conflit. Par contre, le bus **numéro de pe** étant unique, son accès doit être contrôler par le jeton de requête interne.

Cette solution nécessite de l'espace supplémentaire pour ajouter le vecteur, et des broches supplémentaires pour pouvoir passer en même temps les numéros des 2 processeurs non seulement lors de l'acquisition, mais aussi lors de la libération. Mais elle permet de réduire de façon significative le nombre de RCND (voir tableau 6.1). De plus, la possibilité de connaître l'état en communication d'un processeur du réseau au niveau local apporte des facilités pour la gestion du réseau de communication par les processus système.

Tableau 6.1 : Les nombre des RCD et des RCND servis par l'UD au cours de la simulation avec et sans le vecteur d'état des processeurs

	T_R=4000, T_L=400		T_R=4000, T_L=1000		T_R=4000, T_L=2000	
	V=32	V=64	V=32	V=64	V=32	V=64
Sans le vecteur						
Demande d'acquisition	74.34	74.34	60.30	60.38	36.77	43.30
Non destinataire	54.72	54.72	67.42	68.40	55.74	86.59
Avec le vecteur						
Demande d'acquisition	74.32	74.32	63.34	73.91	36.93	53.32
Non destinataire	4.72	4.72	55.78	23.19	39.96	31.48

6.2 Augmenter le débit du réseau

Le réseau de communication d'ARP est un réseau dans lequel les processeurs disposent des liaisons directes pour effectuer leur communication. La contrainte en nombre de broches, nous obligent à utiliser des liaisons séries à haut débit, pour supporter le besoin en communication des processeurs. Les processeurs de plus en plus puissants sont consommateurs de données de plus en plus importantes. L'augmentation du débit du réseau peut se faire en utilisant des solutions tech-

nologiques, comme des composants en BiCmos et en AsGa et la fibre optique, ces solutions ne sont actuellement qu'au stade d'expérimentation : une liaison série à 6 Gbits/s avec tranfert de l'horloge utilisant les composants commerciaux de Siemens est obtenue chez Bull.

Une autre solution au problème peut être apportée par la mise en parallèle de réseaux ARP, afin de permettre la transmission en parallèle des données. Le réseau de communication final est constitué par un empilement de réseaux ARP de base. Nous définissons la première couche de réseau ARP comme étant le réseau **maître** et tous les autres réseaux comme étant des réseaux **esclaves**. Ces réseaux esclaves reçoivent directement des ordres d'établissement ou de destructions des liaisons du réseau maître. Ainsi le traitement des requêtes de communication se fait de la même manière, c'est à dire par l'envoi des requêtes du processeur vers le MCB du réseau maître. Lorsque la requête est satisfaite, le réseau maître envoie des ordres aux réseau esclaves pour établir une liaison **parallèle** entre l'expéditeur et le destinataire. Ainsi, nous conservons la propriété reconfigurable du réseau et nous disposons d'un réseau de débit plus importante.

6.3 Un processeur a plusieurs ports d'E/S

Dans toute la description du projet ARP, nous avons supposé que le processeur ne dispose qu'un seul port entrée/sortie. Cette hypothèse, qui était vraie il y a quelques dizaines d'années, est aujourd'hui peu réaliste, Les processeurs actuels comme le T9000 ou le TMS320C40 de Texas disposent de multiples ports d'E/S séries ou parallèle, qui lui permettent d'effectuer plusieurs communications simultanées. Par ailleurs des topologies comme la grille, l'arbre ou l'hypercube semblent bien adaptées dans certaines applications dédiées, et de très nombre algorithmes de résolution ont été développés sur ces topologies. Il convient donc, à un moment donné, de donner à un ensemble de processeurs la possibilité d'utiliser de telles topologies. Nous proposons plusieurs solutions permettant d'obtenir ces possibilités de fonctionnement.

6.3.1 Cas à 2 ports

Nous supposons dans ce qui suit que chaque processeur dispose de 2 ports d'E/S. Les seules topologies utilisables par ce type de processeur ont une structure linéaire comme l'anneau ou le pipeline.

La première solution consiste à attribuer 2 ports d'E/S d'un MCB à chaque processeur. Les processeurs peuvent alors grâce à ces 2 ports établir des liaisons

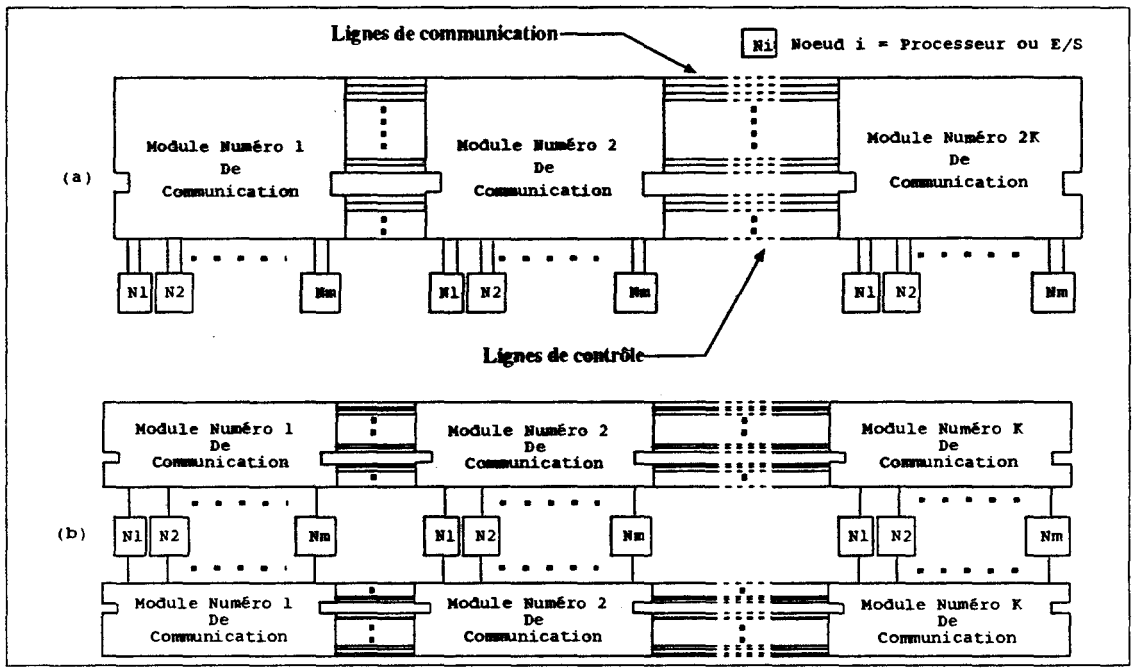

Figure 6.2 : Exemples de réseaux utilisant des processeurs à 2 ports d'E/S : (a) 2 ports d'E/S pour un pe par réseau, (b) un port d'E/S pour un pe par réseau.

pour réaliser toutes les structures linéaires possibles. Pour conserver la même puissance de traitement, c'est à dire le même nombre de processeurs connectés, le nombre de liens du réseau doit être doublé, la limitation provient surtout de la taille maximale du réseau ARP réalisable.

Soit V la taille maximale du réseau ARP, le nombre de ports d'E/S du réseau est en fonction de V. A valeur de V constant, en affectant 2 ports par processeurs, nous diminuant le nombre de processeurs connectables de moitié. Une autre possibilité est de doubler le réseau d'ARP, le processeur dispose d'un port dans chaque réseau. Cette solution permet de conserver la puissance de traitement du système et de doubler le nombre de requêtes pouvant être traitées simultanément. Mais l'utilisation de 2 réseaux distincts génère des situations de conflits : la réalisation d'une structure en anneau n'est pas toujours possible, elle dépend de la parité du nombre de processeurs. Si le nombre est pair le réseau peut être réalisé (figure 6.3.b), par contre si le réseau est impair, la connexion entre le P_1 et P_{2n+1} est impossible, la solution à ce problème nécessite l'utilisation d'un processeur intermédiaire qui sert de passerelle entre P_1 et P_{2n-1} (figure 6.3.c). Malgré tout nous pensons que cette solution est meilleure, puisqu'elle conserve la puissance de calcul, et accélère la vitesse de traitement des requêtes.

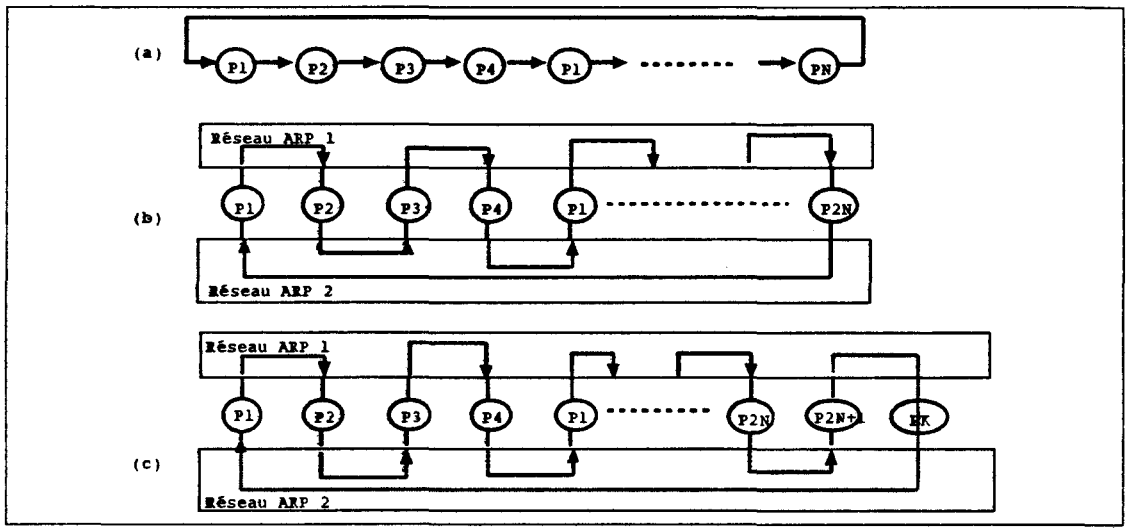

Figure 6.3 : (a) un anneau de N processeurs, (b) réalisation d'un anneau de 2N pe et (c) réalisation d'un anneau de 2N+1 pe

6.3.2 Cas à 3 ports ou plus

Dans le cas où un processeur possède plus de 3 ports, il est plus intéressant d'utiliser autant de réseau que de ports d'E/S que dispose le processeur, pour les mêmes raisons citées ci-dessus. Le problème qu'on peut se poser est plutôt d'ordre théorique : soit D le degré de chaque processeur (c'est à dire le nombre de ports), nous disposons de D réseaux distincts, dans lequel chaque processeur possède un port d'E/S. L'organisation des réseaux ainsi proposée est-elle capable de générer toutes les topologies possibles de degré D ?

Ce problème se ramène à un problème de coloration des arêtes en théorie des graphes [CJ91]. C'est à dire que, étant donné un graphe de degré D, la coloration consiste à colorier chaque arrête du graphe, en n'utilisant que D couleurs, de façon à ce que toutes les arêtes adjacentes du graphe, soient coloriées par une couleur différente. La coloration est possible pour les graphes bipartis, dont l'arbre binaire et l'hypercube sont des exemples. Pour un graphe quelconque le problème est NP-complet.

La technique de la coloration des arêtes permet dans notre cas, de déterminer si une topologie peut être réalisée. En effet, en attribuant une couleur pour chaque réseau, et si la coloration d'un graphe donné est possible, alors ce graphe peut être réalisé par les réseaux ARP. Il suffit pour cela, que chaque couple de processeurs établissent leur liaison dans le réseau désigné par la couleur de leur arête commune.

Le résultat de la coloration sur les graphes bipartis, nous intéresse particulièrement puisque l'arbre binaire, l'hypercube et la grille font parties de ce type de graphe, et ces 3 topologies sont les plus utilisés dans les réseaux de communication à topologie statique.

6.4 Vers un très grand nombre de processeurs

La taille du réseau de communication d'ARP est limité à la fois par le nombre de broches disponibles et par la capacité de son gestionnaire de communication à traiter les requêtes venant des processeurs.

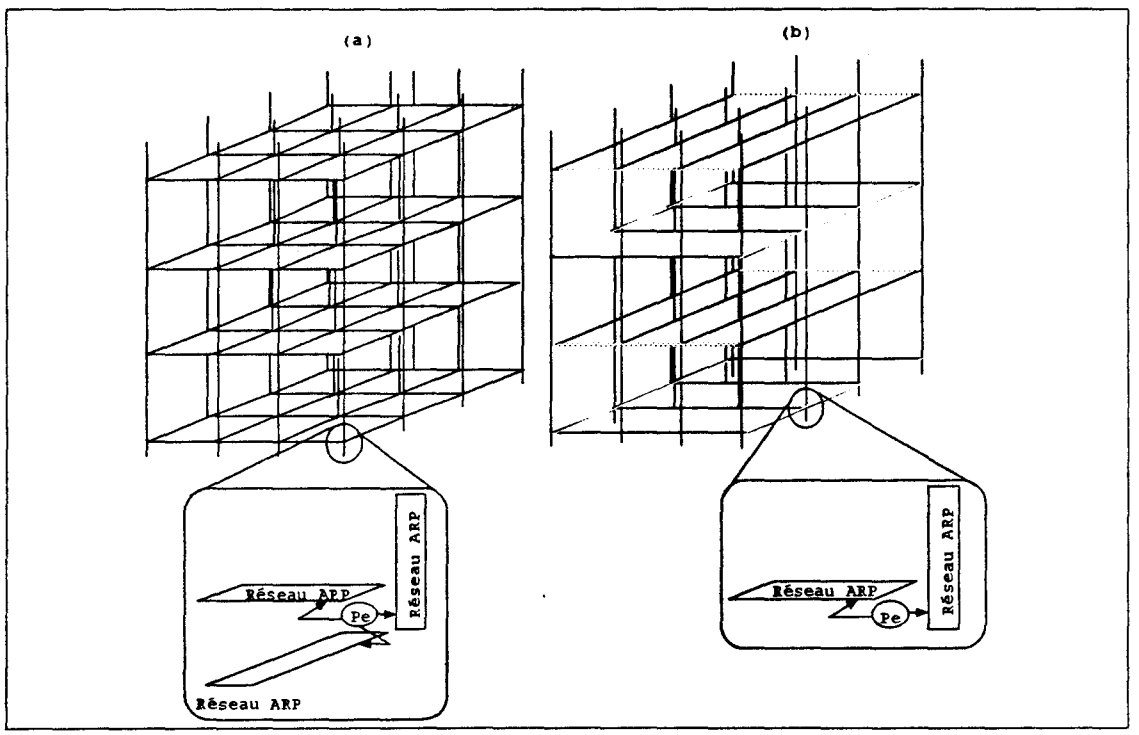

Figure 6.4 : (a) le spanning bus hypecube (b) le dual bus hypercube

Pour obtenir un gain de performance vraiment significatif du système, il est nécessaire de passer à la réalisation en 2 ou 3 dimensions. Parmi les topologies existants, le **spannning bus hypercube (SBH)** et le **dual bus hypercube (DBH)** proposés par Wittie ([WIT81]) semblent particulièrement bien adaptés à notre cas. La figure 6.4 représente un exemple de SPH et de DBH. Les lignes sur le schéma représentent les bus et les noeuds du réseau sont connectés à 2 bus (dans le cas du DBH), ou à 3 bus dans le cas du SBH.

En remplaçant chaque ligne du schéma par un réseau de communication d'ARP, nous obtenons une architecture à 2 ou 3 dimensions. Le nombre de noeuds interconnectables passe théoriquement de N, à N^2 ou N^3. En d'autre termes pour N=128, nous obtenons un système pouvant comporter de 16384 à 2 millions de noeuds. La réalisation physique d'un tel système soulève, bien évidemment, bien d'autres problèmes.

L'intérêt de ce type de réseau est que chaque noeud est connecté sur un réseau ARP, cela lui permet d'accéder n'importe lequel des noeuds se trouvant sur sa ligne avec une distance de un. Pour un réseau de communication de dimension 3 comme le SBH, le diamètre du réseau est de trois, c'est à dire qu'un noeud du réseau peut théoriquement atteindre n'importe lequel autre, en passant que par 2 noeuds intermédiaires et ceci dans un réseau interconnectant 2 millions de processeurs.

Le routage dans un réseau à 3 dimensions peut se faire soit par une technique dérivé du store and forward, soit par commutation de circuits. La première méthode conserve le protocole de communication définie dans le cadre de ce travail, mais elle nécessite au niveau du noeud des capacités de tamponnement importantes. La deuxième solution consiste à envoyer un entête pour établir le chemin entre l'expéditeur et le destinataire, une fois le chemin établit le message pourra transiter le long de ce chemin.

6.5 Conclusion

Nous venons de présenter dans ce chapitre quelques améliorations possibles du réseau de communication d'ARP, comme la réduction du temps de contrôle ou l'augmentation du débit possible sur le réseau. De plus nous proposons ici des organisations architecturales qui utilisent le réseau d'ARP comme brique de base. La première proposition conserve la même performance de traitement, mais définit un réseau de communication qui génère des topologies plus complexes. La deuxième proposition permet d'augmenter considérablement la puissance de traitement, en utilisant une topologie à plusieurs dimensions.

Conclusion

2 février 1993

Dans une machine sans mémoire commune, les processeurs communiquent par échanges de messages via des liaisons point à point.

Les machines à connectique fixe utilisent des liaisons permanentes organisées selon un graphe d'interconnexion régulier tel qu'une grille ou un hypercube. Les messages qui ne bénéficient pas d'une liaison directe doivent être routés de voisin en voisin jusqu'à leur destination. Le routage de ces messages pose des problèmes de conflits et d'interblocages, ce qui réduit fortement la bande passante du réseau.

Dans ce type de stratégie de communication, la distance entre 2 processeurs n'est pas identique et dépend de la topologie du réseau. La vitesse de l'exécution d'une application est tributaire de l'adéquation entre son graphe des communications entre processus et le graphe d'interconnexion physique des processeurs.

La recherche du placement optimal des processus est doublement difficile :

- La nature dynamique des applications peut fait évoluer le graphe initial des processus rendant le placement initiale inefficace.
- La nature statique de la topologie de la machine ne permet pas toujours l'obtention du meilleur placement possible pour une application.

Une solution à ces problèmes est donnée par des machines à réseau reconfigurable. La nature reconfigurable du réseau permet de générer un ensemble de topologies pour offrir la possibilité, d'avoir la meilleure machine possible pour l'exécution d'une application donnée et au réseau des processeurs d'évoluer avec le graphe des processus au cours de l'exécution.

Nous avons dans le cadre du projet ARP défini une architecture de réseau d'interconnexion à reconfiguration dynamique et asynchrone. Une liaison ne sera créée ou détruite qu'en fonction des requêtes des processeurs, et la modification d'une liaison n'affecte pas les communications en cours. Les accès au réseau sont gérés par

coopération entre les unités de communication associées à chaque processeur, par ailleurs nous avons adopté une approche modulaire dans le but de construire une machine facilement extensible à la demande.

L'intérêt des réseaux reconfigurables est que les communications s'effectuent sur une liaison physique directe, donc plus rapide. La transmission des données se fait qu'une fois le chemin établi, la bande passante du réseau n'est utilisée que pour la transmission des données de l'expéditeur vers le destinataire. L'absence des données vers un (ou d'un) noeud intermédiaire permet de réduire fortement la charge du réseau de communication. Le gain apporté par cette solution se fait au prix d'une commande plus complexe et en général moins rapide.

De plus, la possibilité qu'offre la reconfiguration dynamique asynchrone aboutit à une commande séquentielle du réseau, ce qui est semble contradictoire avec la philosophie des machines parallèles. L'effort est donc surtout porté pour masquer le temps de contrôle nécessaire à la commande par le temps de communication. Les simulations effectuées montrent que pour des applications à grains moyens, le masquage du temps de contrôle par le temps de communication peut être réalisé.

Par ailleurs, des études ont montré que le nombre de communications simultanées dans un réseau de crossbar CCM ne dépasse pas 63

La faisabilité de l'ensemble est en grande partie démontrer par la réalisation en VLSI du module de communication de base. Etant données le nombre limité de broches exploitables, nous sommes amenés à adopter la transmission série à très haut débit. En ce qui concerne sa faisabilité nous sommes basés sur les travaux issues du projet M3S et du laboratoire Bull.

Le réseau de crossbar CCM utilisé offre théoriquement la possibilité d'effectuer facilement la diffusion, qui peut être de 1 vers N, ou plus finement un vers k avec $k \leq N$. Le réseau peut ainsi offrir des primitives de communications globales intéressantes.

L'approche adoptée pour réaliser la commande du réseau est résolument distribuée et asynchrone. Elle permet efficacement d'établir ou de détruire des connexions de façon asynchrone. Cette possibilité facilite le placement dynamique des processus : au lieu de déplacer le contexte d'un processus, elle permet dans certains cas (par exemple surcharge en communication) de conserver le contexte en ne modifiant que les connexions.

Le réseau est conçu en supposant que chaque processeur ne dispose que d'un port d'E/S. Le réseau obtenu ne permet pas de générer des topologies comme l'hypercube, l'arbre binaire ou la grille. Des solution, qui sont proposées dans le chapitre 6, utilise les MCB réalisés comme des modules de bases pour construire des systèmes plus complets qui interconnecte des processeurs munient des ports

d'E/S multiples (cas des transputers). Il conviendra par la suite d'examiner plus en détaille ces solutions, ainsi que les possibilités d'extension en 2 voir 3 dimensions.

D'autres solutions permettant d'optimiser le temps de contrôle sont également proposées. Il faut à celle-ci ajouter des solutions qui recouvrent les 2 informations nécessaires pour l'acquisition d'une voie, ou des solutions technologiques apportées par l'utilisation de l'hyperfréquence, qui offrent la possibilité d'effectuer des demandes en parallèles.

Références

[Agr82] D. Agrawal. Testing and fault tolerance of multistage interconnection network. *IEEE Computer*, pages pp.41–53, April 1982.

[All80] A.O. Allen. Queueing models of computer systems. *IEEE Computer*, pages pp.13–24, April 1980.

[And77] S. Andresen. The looping algorithm extended to base 2^t rearrangeable switching networks. *IEEE Transactions on Computers*, vol. C25(no.10):pp.197–203, October 1977.

[AS83] G.R. Andrews and F.B. Schneider. Concepts and notations for concurrent programming. *ACM Computing Surveys*, vol 15(no. 1):pp.3–43, March 1983.

[AS88] W. C. Athas and C. L. Seitz. Multicomputers: Message–passing concurrent computers. *IEEE Computer*, pages pp.9–24, August 1988.

[Bac78] J. Backus. Can programming be liberated from the Von Neumann style ? a functional style and its algebra of programs. *ACM*, vol. 21(8):pp.613–641, August 1978.

[Bal84] R.V. Balakrishnan. The proposed ieee 896 futurebus - a solution to the bus driving problem. *IEEE Micro*, pages pp.23–27, August 1984.

[Bat76] K.E. Batcher. The flip network in Staran. In *Proceeding of the 1976 International Conference on Parallel Processing*, pages pp.65–71. IEEE, 1976.

[Ben62] V. E. Benès. On rearrangeable three–stage connecting network. *Bull System Technical Journal*, XLI(5):pp.1481–1492, September 1962.

[Bin90] M. Binse. Systèmes connexionnistes. *Publication interne LIFL*, (no. I.T. 178), January 1990.

[Bon87] I. Bond. Grands réseaux d'interconnexion. In *Thèse de doctorat en informatique*. Université de Paris-sud, Centre d'orsay, 1987.

[Bra83] G.W. Brams. Réseaux de pétri, tome 1 : Théorie et analyse. *Masson*, 1983.

[CB91] F. Capello and J.-L. Béchennec. PTAH : un réseau de processeurs à géométrie compilable pour les applications de traitement numérique intensif. In *Troisième symposium sur les Architectures Nouvelles de Machines*, pages pp. 249–270. PRC ANM - CNRS - MRT, June 1991.

[CF80] C.Wu and T. Feng. On a class of multistage interconnexion networks. *IEEE Computer*, vol. C29(no.8):pp.694–702, August 1980.

[CJ91] C.Berge and J.C.Fournier. A short proof for a generalization of vizing's theorem. *Journal of Graph Theory*, vol.15(no.3):pp.333–336, March 1991.

[Clo53] C. Clos. A study of non-blocking switching networks. *Bull System Technical Journal*, 32:pp.406–424, March 1953.

[Cor91a] Intel Corporation. Paragon XP/S product overview. 1991.

[Cor91b] Thinking Machine Corporation. The Connection Machine CM5 technical summary. October 1991.

[CR87] M. Cosnard and Y. Robert. Algorithmique parallèle : une étude de complexité. *Technique et Science Informatique*, vol. 6(2):pp.115–125, June 1987.

[CS91] W. Chen and J. Sheu. Performance analysis of multiple bus interconnection networks with hierarchical requesting model. *IEEE Transactions on Computers*, vol.40(no.7):pp.834–842, July 1991.

[Dal90] W. Dally. Network and processor architecture for message-driven computers. In Morgan Kaufmann, editor, *VLSI and Parallel Computation*, pages pp. 140–222, 1990.

[DB85] C.R. Das and L.N. Bhuyan. Bandwidth availability of multiple–bus mutiprocessors. *IEEE Transactions on Computers*, c-34(10):pp.918–926, October 1985.

[DS86] W.J. Dally and C.L. Seitz. The torus routing chip. *Journal of Distributed Systems*, vol.1(no.3):pp.187–196, 1986.

[DS87a] W. J. Dally and P. Song. Design of a self-timed vlsi multicomputer communication controller. In ICCD-87, editor, *Proc. International Conference on Computer Design*, pages pp.230–234. IEEE, 1987.

[DS87b] W.J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, c-36(5):pp.547–553, May 1987.

[Dua91] J. Duato. Deadlock-free adaptative routing algorithms for multicomputers. *Technique et Science Informatique*, vol.10(no.4):pp.275–285, October 1991.

[Dyn62] E.B. Dynkin. Théorie des processus Markoviens. *Dunod*, 1962.

[ES88] M. Elderkin and P. Sweazy. Multiprocessing sur le Futurebus ieee-896. *Electronique Industrielle*, pages pp.53–57, October 1988.

[Fdi84] S. Fdida. Etude de systèmes à partage de ressources par réseaux de files da'ttente. application à l'architecture : Multiprocesseur sm90. *Thèse d'Université de Paris VI*, March 1984.

[Fel79] J.A. Feldman. High level programming for distributed computing. *Communications of the ACM*, pages pp.353–368, June 1979.

[Fen81] T. Feng. A survey of inteconnection networks. *IEEE Transactions on Computers*, pages pp.12–27, December 1981.

[Flo87] G. Florin. Les réseaux de petri stochastiques. *Thèse d'état*, 1987.

[FP88] F.André and JL. Pazat. Le placement de tâches sur des architectures parallèles. *Technique et Science Informatique*, vol.7(no.4):pp.385–401, September 1988.

[FP89] S. Fdida and G. Pujolle. Modèle de systèmes et de réseaux, tome 1 et 2. *Eyrolles*, 1989.

[FPM86] S. Fdida, G. Pujolle, and D. Mailles. Réseaux de files d'attente avec sémaphores. *Technique et Science Informatique*, vol. 5(no. 3):pp.187–196, May 1986.

[Fra81] M. A. Franklin. VLSI performance comparison of banyan crossbar communications networks. *IEEE Transactions on Computers*, c-30(no.4):pp.283–291, April 1981.

[Fra90] P. Fraigniaud. Communications intensives dans les architectures à m—'emoire distribuée et algorithmes parallèles pour la recherche de racines de polynômes. In *Thèse de doctorat en informatique.* Ecole Normale Supérieure de Lyon, December 1990.

[Fra92] P. Fraigniaud. Communications dans un réseau de processeurs. In Masson, editor, *Algorithmique parallèle*, pages pp. 133–147, 1992.

[FWT82] M.A. Franklin, D.F. Wann, and W.J. Thomas. Pin limitations and partitionning of VLSI interconnexion. *IEEE Transactions on Computers*, vol.31(no. 11):pp.1109–1116, November 1982.

[FZ86] D. Ferrari and S. Zhou. A load index for dynamic load balancing. *Proc 1986 Fall Joint Computer Conference*, November 1986.

[GCKW79] D.I. Good, R.M. Cohen, and J. Keeton-William. Principe of proving concurrent programs in gypsy. In *Proc. 6th ACM Symp. Principles of programming languages*, pages pp.42–52, New York, January 1979. ACM.

[GDM91] G.Bernard, D.Stève, and M.Simatic. Placement et migration de processus dans les systèmes répartis faiblement couplés. *Technique et Science Informatique*, vol.10(no.5):pp.375–392, October 1991.

[Gec77] J. Gecsei. Interconnection networks from three-state cells. *IEEE Transactions on Computers*, vol. C26(no.8):pp.705–711, October 1977.

[GGK+83] A. Gottlieb, R. Grishman, C. P. Kkruskal, K. P. McAuliffe, L. Rudolph, and M. Snir. The NYU ultracomputer–designing an mimd shared memory parallel computer. *IEEE Transactions on Computers*, c-32(2):pp.175–189, February 1983.

[Got92] A. Gottlieb. Architectures for parallel supercomputing. 1992.

[GR89] C. Germain-Renaud. Etudes des mécanismes de communication pour une machine massivement parallèle : MEGA. In *Thèse de doctorat en informatique.* Université de Paris-sud, Centre d'orsay, December 1989.

[Gui89] F.E. Guibaly. Design and analysis of arbitration protocole. *IEEE Transactions on Computers*, vol.38(no. 2):pp.161–171, February 1989.

[Gus84] D.B. Gustaveson. Computer buses - a tutorial. *IEEE Micro*, pages pp.7–22, August 1984.

[Hil85] W.D. Hillis. In The MIT Press, editor, *The Connexion Machine*, 1985.

[HJ91] J.L. Hennessy and N.P. Jouppi. Computer technology and architecture : An evolving interaction. *IEEE Computer*, pages pp.18–29, September 1991.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, vol. 21(8):pp.666–677, August 1978.

[HP92] J.L. Hennessy and D. A. Patterson. Mémoire principale. In Mc Graw Hill, editor, *Architecture des ordinateurs : une approche quantitative*, pages pp. 426–433, 1992.

[HS91] R.V. Hanxleden and L.R. Scott. Load balancing on message passing architectures. *Journal of Parallel and Distributed Computing*, pages pp.312–324, 1991.

[Inm88] Inmos. Occam2 reference manual. *Prentice-Hall International Series in Computer Science*, 1988.

[Inm91] Inmos. The T9000 transputer products overview manuel. 1991.

[JH89] S.L. Johnsson and C. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, vol.38(no. 9):pp.1249–1268, September 1989.

[Jon91] S.L. Peyton Jones. A Future interface from off-the-shelf parts. *IEEE Micro*, pages pp.38–51, February 1991.

[KK79] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, pages pp.267–286, 1979.

[LAD+92] C.E. Leiserson, Z.S. Abuhamdeh, D.C. Douglas, C.R. Feynman, and J.V. Hill. The network architecture of the Connection Machine CM5. *SPAA 92*, 1992.

[Law75] D.H. Lawrie. Access and alignment of data in an array processor. *IEEE Transactions on Computers*, vol. C-24(no. 12):pp.1145–1155, December 1975.

[Lei85] C.E. Leiserson. Fat-trees : Universal networks for harware-efficient supercomputing. *IEEE Transactions on Computers*, c-34(10):892–901, October 1985.

[Lei92] F. Thomson Leighton. Introduction to parallel algorithms and architectures : Array, trees, hypercubes. Morgan Kaufmann, 1992.

[Len78] J. Lenfant. Parallel permutation of data : A Benes networks control algorithm for frequently used permutations. *IEEE Transactions on Computers*, vol.C27(no.7):pp.204–214, July 1978.

[LHSP90] D. Litaize, O. Hmmami, P. Saintrat, and R. Pulou. Les liaisons série du multiprocesseur m3s justification, problèmes techniques, solutions. In *Deuxième symposium sur les Architectures Nouvelles de Machines*, pages pp 243–265. PRC ANM – CNRS – MRT, September 1990.

[Lit90] D. Litaize. Architecture multiprocesseurs à mémoire commune. In *Deuxième symposium sur les Architectures Nouvelles de Machines*. PRC ANM – CNRS – MRT, September 1990.

[Lit92] D. Litaize. La liaison série à ultra haut débit : une (la ?) solution pour les liaisons inter-module en environnement mutiprocesseur. In *Technologie Matérielles Futures de l'ordinateur*, March 1992.

[LNBA89] Q. Yang L. N. Bhuyan and D. P. Agrawal. Performance of multiprocessor interconnection networks. *IEEE Computer*, pages pp.25–37, February 1989.

[LS88] I. Lee and D. Smitlay. A synthesis algorithm for reconfigurable interconnection networks. *IEEE Transactions on Computers*, vol.37(no.6):pp.691–699, June 1988.

[LVA82] T. Lang, M. Valero, and I. Alegre. Bandwidth of crossbar and multiple-bus connections for multiprocessors. *IEEE Transactions on Computers*, vol.C-31(no.12):pp.1127–1234, December 1982.

[LVF83] T. Lang, M. Valero, and M. A. Fiol. Reduction of connections for multibus organization. *IEEE Transactions on Computers*, c-32(8):pp.707–716, August 1983.

[MFM+89] K. Murakami, A. Fukuda, S. Mori, T.Sueyoshi, and S. Tomita. The kyushu university reconfigurable parallel processor-design of memory and intercommunication architectures. In *Proc. of ACM SIGARCH 1989 Int. Conf. On Supercomputing*, pages pp.351–360, June 1989.

[MHW87] T.N. Mudge, J.P. Hayes, and D.C. Winsor. Multiple bus architecture. *IEEE Computer*, pages pp.42–48, June 1987.

[MMF+89] K. Murakami, S. Mori, A. Fukuda, T.Sueyoshi, and S. Tomita. The kyushu university reconfigurable parallel processor-design philosophy and architecture. In Elsevier, editor, *Proc. of the 11th World Comp. congress*, pages pp.995–1000, August 1989.

[MT91] T. Muntean and E. Talbi. Méthodes de placement statique des processus sur architectures parallles. *Technique et Science Informatique*, vol.10(no. 5):pp.355–373, October 1991.

[MW90] T. Muntean and P. Waille. L'architecture des machines supernode. *La lettre du transputer*, 7:pp.11–40, September 1990.

[Nat80] S. Natkin. Réseaux de Petri stochastiques. *Thèse de Docteur-Ingénieur à CNAM-Paris*, June 1980.

[NS81] D. Nassimi and S. Sahni. A self-routing Benès network and parallel permutation algorithms. *IEEE Transactions on Computers*, vol. C30(no.5):pp.332–340, May 1981.

[NXG85] L.M. Ni, C. Xu, and T.B. Gendreau. A distributed drafting algorithm for load balancing. *IEEE Transactions on Software Engeneering*, vol.11(10):1153–1161, October 1985.

[PBG90] H. Pujol, E. Belhaire, and P. Garda. Local interconnection through split busses for multilayered Boltzmann machines. In *Troisième symposium sur les Architectures Nouvelles de Machines*, pages pp 197–207. PRC ANM - CNRS - MRT, June 1990.

[Pet66] C.A. Petri. Communication with automata. In Tech. Rep. RADC-TR-65-377, editor, *Ph. D dissertation*, New York, January 1966.

[PKG92] H. Pujol, J. Klein, and P. Garda. Le bus sécable : Réalisation des communications dans un accélérateur de la machine de Bolzmann. In *RenPar4 4èmes Rencontres du Parallèlisme*, pages pp. 52–55, March 1992.

[PKM+82] U.V. Premkumar, R. Kapur, M. Malek, G.J. Lipovski, and P. Horne. Design and implementation of the banyan interconnection network in TRAC. *AFIPS Conference Proceedings 1982 National Computer Conference*, vol. 51:pp.643–653, 1982.

[Qui92] P. Quinton. Les architectures parallèles et leur mise en oeuvre matérielle. In *Technologie Matérielles Futures de l'ordinateur*, March 1992.

[Ray85] M. Raynal. In Eyrolles, editor, *Algorithmes distribués et protocoles*, 1985.

[RF87] D.A. Reed and R.M. Fujimoto. Multicomputer networks: message-based parallel processing. *MIT Press Cambridge*, March 1987.

[RT86] R. Rettberg and R. Thomas. Contension is no obstacle to shared-memory multiprocessing. *Communications of the ACM*, vol. 29(no. 29):pp.1202–1212, December 1986.

[Sai91] P. Sainrat. Réseau d'interconnexion du multiprocesseur M3S : étude et mise en oeuvre. In *Thèse de doctorat en informatique*. L'université Paul Sabatier, January 1991.

[SC91] H.S. Stone and J. Cocke. Computer architecture in the 1990s. *IEEE Computer*, pages pp.30–38, September 1991.

[SHH90] J.E. Smith, W.C. Hsu, and C. Hsiung. Future general purpose supercomputer architectures. *Supercomputing 90*, pages pp.796–804. November 1990.

[Sim88] Simulog. Manuel de référence du QNAP2. 1988.

[Sny82] L. Snyder. Introduction to the reconfigurable, highly parallel computer. *IEEE Computer*, vol.15:pp.47–56, January 1982.

[Tau76] D.M. Taub. Contension resolving circuits for computer interrupt systems. *proc. IEEE*, pages pp.28–41, September 1976.

[Tau84] D.M. Taub. Arbitration and control acquisition in the proposed IEEE 896 futurebus. *IEEE Micro*, pages pp.28–41, August 1984.

[TBA85] A. Thareja, J. Buzen, and S. Agrawal. BEST1/SNA: A software tool for modelling and analysis of IBM SNA networks. In North Holland, editor, *Proc. Modelling Techniques and Tools for Performance Analysis*, pages pp.81–98, Amsterdam, 1985.

[Try92] D. Trystram. Communication dans les grilles de processeurs. In Masson, editor, *Algorithmique parallèle*, pages pp.158–169, 1992.

[VC92] A. Varma and S. Chalasani. Fault-tolerance analysis of one-sided crosspoint switching networks. *IEEE Transactions on Computers*, vol.41(no.2):pp.143–158, February 1992.

[VM88] M.K. Vernon and U. Manber. Distributed round–robin and first–come first–serve protocols and their application to multiprocessor bus arbitration. *IEEE*, pages pp.269–277, 1988.

[VP84] M. Veran and D. Potier. QNAP2 : A portable environnement for queueing systems medelling. *Research Report N.314, INRIA*, June 1984.

[WIT81] L D. WITTIE. Communication structures for large networks of microcomputers. *IEEE Transactions on Computers*, vol. C30(no.4):pp.264–273, April 1981.

[YA85] S. Yalamanchili and J. K. Aggarwal. Reconfiguration strategies for parallel architectures. *IEEE Computer*, pages pp.44–61, December 1985.

Liste des figures

Table des tableaux