THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS VI

SPÉCIALITÉ INFORMATIQUE

Présentée par Abdelhafid BOUHRAOUA

Pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ PARIS VI

MISE EN OEUVRE, ÉVALUATION DE PERFORMANCES ET VÉRIFICATION DE LA VALIDITÉ DE TOPOLOGIES ET DE SCHÉMAS DE ROUTAGE POUR L'AIDE À LA CONCEPTION DE RÉSEAUX D'INTERCONNEXION POUR ARCHITECTURES PARALLÈLES

À soutenir, le 13 Mai 1998, devant le jury composé de

PIERRE FRAIGNIAUD	Rapporteur
ALAIN GREINER	Directeur de Thése
DANIEL LITAIZE	Rapporteur
JEAN-LOUIS PAZAT	Examinateur
ISAAC D. SCHERSON	Examinateur

Résumé

Cette thèse présente plusieurs travaux visant la fourniture d'outils permettant la conception de réseaux d'interconnexion pour des machines parallèles qui sont des réseaux de stations de travail. Plus particulièrement, cette thèse tourne autour de deux axes: le routeur RCUBE et la machine Multi-PC respectivement développé et en cours de développement au sein de l'équipe ASIM du laboratoire LIP6.

L'étiquetage de plusieurs topologies régulières utilisant RCUBE et conformes aux spécifications de Multi-PC sont présentées. Les résultats de l'évaluation des performances de ces topologies pour plusieurs tailles et sous différentes charges sont également présentés ici. Le coût de l'adaptativité est discuté au vu des résultats obtenus.

L'évaluation des performances de différentes topologies a permi d'effectuer une évaluation du routeur RCUBE. Cette thèse propose une extension de RCUBE par l'ajout d'une structure de mémorisation centralisée.

Le support de topologies non-régulières est rendu possible par la fourniture d'un algorithme de configuration automatique de topologies quelconques. Cet algorithme est une modification de l'algorithme de Van-Leeuwen dont le but est de produire un adressage linéaire des seuls processeurs dans des réseaux où il est possible que certains routeurs ne comportent aucun processeur.

Finalement, une théorie de vérification formelle de la validité d'un schéma de routage est présentée. Une méthode de vérification automatique, dérivée de cette théorie, et les principes de son implémentation logicielle sont également présentés.

Mots-clés

réseaux d'interconnexion
topologies
schémas de routage
evaluation des performances
interblocage

Abstract

This thesis presents several works targeting to provide a set of tools that will allow a proper design of interconnection networks for parallel machines that are networks of workstations. More particularly, this thesis turns around two axis: the RCUBE router and the Multi-PC machine respectively developed and in development within the ASIM team of the LIP6 laboratory.

The labelling of several standard topologies using RCUBE and compliant with the Multi-PC machine specifications are presented. The results of performance evaluation of these topologies for several sizes and under different workloads are presented too. The adaptivity cost is discussed as a consequence of the obtained results.

The performance evaluation of different topologies allowed to make an evaluation of the RCUBE router. This thesis suggests an extension of RCUBE with a central buffering structure.

Supporting non-standard topologies is made possible by providing an automatic configuration algorithm working on any given topology. This algorithm is a modification of the Van-Leeuwen algorithm. Its main goal is producing a linear adressing of processors only in networks where it is possible that some routers have no attached processor.

Finally, a complete theory of formal verification of the validity of a routing scheme is presented. An automatic verification method, derived from this theory, and the principles of its software implementation are presented too.

Keywords

interconnection networks
topologies
routing schemes
performance evaluation
deadlock

Remerciements

Tous ceux qui ont connu les affres de la rédaction d'un mémoire de thèse, savent combien ce travail est par essence un effort tout personnel. C'est là, le paradoxe de cet exercice réalisé dans un domaine — la microélectronique, où plus qu'ailleurs, le travail d'équipe est la clé de la réussite.

C'est pourquoi je tiens particulièrement à remercier Belkacem ZERROUK qui fut l'initiateur du projet MILE et dont les idées originales ont servi de base à ce travail. Je veux exprimer ici toute ma gratitude envers le professeur Alain GREINER qui a dirigé ces recherches, à la fois pour son travail d'encadrement, ses conseils avisés et sa très grande patience dans la lecture de mon mémoire.

Je remercie Daniel ÉTIEMBLE — professeur au LRI à Orsay et Daniel LITAIZE — professeur à l'IRIT à Toulouse qui ont accepté d'être les rapporteurs de mes travaux, ainsi que les professeurs Isaac SCHERSON, professeur à l'univeristé de Californie at Irvine et le professeur Jean-Louis PAZAT, maître de conférence à l'IRISA à Rennes qui ont bien voulu en être les examinateurs.

Je veux remercier les membres de l'équipe avec qui j'ai partagé pendant 4 ans et demi une ambiance chaleureuse. Je remercie particulièrement Mr Ilponse FABRICE, étudiant de DEA, dont le travail et le sérieux a été d'un apport appréciable à la finalisation de ce travail.

J'exprime ma plus grande gratittude à ma femme qui m'a profondément et continuellement soutenu tout au long de la redaction de ce mémoire. Enfin, il me tiens à cœur d'exprimer ma plus profonde reconnaissance envers mes parents, qui m'ont donné la chance de pouvoir poursuivre aussi loin mes études, et ce dans les meilleurs conditions qui soient.

À mes deux mères Anissa et Zoulikha
À ma douce épouse Naima
À mes deux pères Ali et Ali

Table des matières

Liste des figures

Liste des tables

CHAPITRE 1

INTRODUCTION

Depuis l'apparition des microprocesseurs, puis des micro-ordinateurs l'informatique a subi une transformation fondamentale. Ainsi, une nouvelle informatique composée de stations de travail autonomes connectées par un réseau local a vu le jour.

Aujourd'hui, nous assistons à une explosion de la demande due à l'intégration de plusieurs médias dans l'ordinateur ouvrant la porte à de nouvelles formes d'applications très gourmandes en puissance de calcul. Il devient de plus en plus difficile aux architectures conventionnelles de répondre à cette demande. Par ailleurs, les architectures parallèles possèdent des caractéristiques qui les rendent aptes à répondre efficacement à cette demande de puissance.

Depuis quelques années, la conception de machines parallèles s'est orientée vers les réseaux de stations de travail. Les principales caractéristiques de ce type de machine sont d'une part, le découplage introduit entre le réseau de communication et les noeuds de calcul, et l'utilisation de stations de travail comme noeud de calcul d'autre part.

Une machine parallèle est, par essence, constituée de plusieurs processeurs qui coopèrent en vue de réaliser une tâche commune.

Du point de vue architectural, puisque la communication inter-processeurs est fondamentale, elle doit être véhiculée par un support adéquat. Les principales caractéristiques d'un tel support sont: la latence, le débit, la fiabilité et le coût.

L'existence de communications entre les processeurs est due aux différents processus constituant l'application. Deux modes de communication sont possibles. Soit de manière explicite par l'échange de messages; soit de manière implicite par l'accès à une zone mémoire partagée. De là découle l'existence de deux modèles de programmation des machines parallèles:

- Le modèle à passage de message (*message passing*). Dans ce modèle, les processeurs possèdent chacun un espace d'adressage privé.

- Le modèle de mémoire partagée (*shared memory*): Ici, tout l'espace mémoire qui peut être physiquement réparti entre les processeurs est adressable par chacun d'eux.

Dans le second modèle, chaque accès mémoire peut être local ou distant. Dans le premier cas, l'accès se fera normalement comme dans un système monoprocesseur. Par contre si l'adresse est distante, l'accès sera transformé en requête envoyée au noeud où est située la zone mémoire contenant la donnée demandée. Le réseau de communication acheminera cette requête sous la forme d'un message. La réponse à la requête fera l'objet d'un second message.

Finalement, quel que soit le modèle de programmation, le dispositif de communication assure toujours une fonction de transfert de messages.

Le dispositif de communication, quelle que soit sa nature, est normalement composé de deux parties distinctes:

- une partie matérielle composée des liaisons physiques et du contrôleur de réseau câblé ou micro-programmé qui exécute le protocole de très bas niveau réalisé par le matériel.

- toutes les couches logicielles qui réalisent l'interface avec les applications.

On nomme généralement réseau d'interconnexion, l'ensemble de la partie composée par les composants matériels et le contrôleur de réseau. Dans ce travail, seul la partie réseau d'interconnexion sera abordée.

En général, dans une machine parallèle, tout processeur peut être amené à communiquer avec tout autre processeur. Naturellement, le réseau d'interconnexion idéal est le réseau complètement interconnect'e. Dans ce type de réseau, une connexion directe point à point est établie entre chaque paire de processeurs. Il est aisé de remarquer qu'une telle configuration devient vite irréaliste dès que l'on dépasse quelques dizaines de processeurs. Il faut donc que le réseau d'interconnexion puisse être capable d'assurer la communication entre tous les processeurs avec un nombre limité de liens par processeur.

Dans ce cas, il existe des noeuds qui font office d'intermédiaires entre d'autres noeuds non directement connectés. Ce qui introduit un surcoût (overhead) dû au routage de ces messages.

Pour pallier à cela, un découplage est effectué entre la fonction de calcul utile, liée à l'application, et la fonction de redirection des messages ou *routage*. Celui-ci est réalisé par des *routeurs* spécialisés distincts des processeurs. Ainsi, il est possible d'imaginer la connection des noeuds processeurs par n'importe quelle topologie composée de noeuds-routeurs. Les messages, dans cette approche, peuvent être amenés à traverser plusieurs routeurs avant d'atteindre leur destination finale.

Ce type de réseau d'interconnexion est constitué de trois entités distinctes:

- Les routeurs, qui possèdent une certaine capacité de traitement.

- La structure du réseau ou *topologie* qui lie les routeurs et les processeurs entre eux.

- Le plan directeur ou *schéma de routage* qui permet aux messages de se déplacer dans la topologie.

En 1994, le projet de réaliser un routeur de messages à hautes performances fut lancé au sein de l'équipe ASIM du laboratoire LIP6. Ce projet a abouti à la conception et à la fabrication du routeur RCUBE [1] [2]. En 1995, un projet visant à utiliser le routeur RCUBE pour réaliser le réseau d'interconnexion de machines parallèles dont les noeuds de calcul sont des cartes mères de PC a été lancé: le projet Multi-PC.

Ce travail, vise à fournir des outils pour construire des réseaux d'interconnexion pour des machines parallèles de type Multi-PC.

Ce document se décompose en 7 chapitres. Le chapitre 2 rappelle les notions fondamentales nécessaires à la compréhension de la problématique et du reste du travail. Le chapitre 3 définit les objectifs de ce travail et l'ensemble des questions auxquelles nous cherchons à apporter des réponses.

Plusieurs topologies de réseaux construites avec des routeurs RCUBE et conformes aux spécifications de la machine Multi-PC sont étudiés et évaluées dans le chapitre 4. Les résultats obtenus ont permis la détermination des topologies les plus performantes. L'apport de l'adaptativité est également discuté dans ce chapitre.

A partir de ces résultats, on effectue une analyse du routeur RCUBE. Le chapitre 5 présente cette analyse ainsi qu'une proposition d'amélioration de la stratégie de mémorisation de RCUBE.

Une méthode de configuration automatique de topologies quelconques composée de routeurs et de processeurs est présentée dans le chapitre 6. Cette méthode est dérivée de l'algorithme de Van Leeuwen-Tan.

Enfin, le chapitre 7 présente une théorie formelle qui traite de la validité d'un schéma de routage. Nous présentons également, dans ce chapitre, une méthode de vérification automatique de la validité d'un schéma de routage basée sur cette théorie.

CHAPITRE 2

NOTIONS FONDAMENTALES

2.1 Introduction

On décrit dans ce chapitre les constituants d'un réseau d'interconnexion. On présente les différentes formes que peuvent prendre ces constituants. On présente également les différentes caractéristiques de ces constituants. Ces constituants sont:

- la topologie
- le schéma de routage
- la technique de routage
- le contrôle de flux

Le modèle du réseau d'interconnexion qu'on se propose d'étudier est le suivant:

- Il est composé de noeuds reliés par des liens bidirectionnels. Chaque lien bidirectionnel comporte à chaque extrémité un port d'entrée et un port de sortie.
- Chaque noeud est un routeur auquel peuvent être rattachés un ou plusieurs processeurs.
- Chaque processeur est relié au routeur associé à travers un lien bidirectionnel local. Le port d'entrée du processeur est appelé **port terminal**. Le lien le reliant à un port de sortie du routeur est appelé **lien terminal**. Le port de sortie du processeur est appelé **port d'injection**. Le lien le reliant à un port d'entrée du routeur est appelé **lien d'injection**.
- Les routeurs ne sont pas adressables, seuls les processeurs le sont. Ceux-ci sont identifiés par des étiquettes distinctes prises dans un ensemble d'entiers croissants appelé **l'espace d'adressage**.

2.2 Topologie

La topologie est la structure géométrique reliant d'une certaine manière les différents noeuds d'un réseau d'interconnexion. Il existe deux types de topologies: les topologies **directes** et les topologies **indirectes**. Les topologies directes sont celles où à chaque routeur est attaché un processeur. Les topologies indirectes sont celles où quelques routeurs sont dépourvus de processeurs.

Certaines topologies sont **régulières** La figure 2.1 présente quelques-unes de ces topologies.

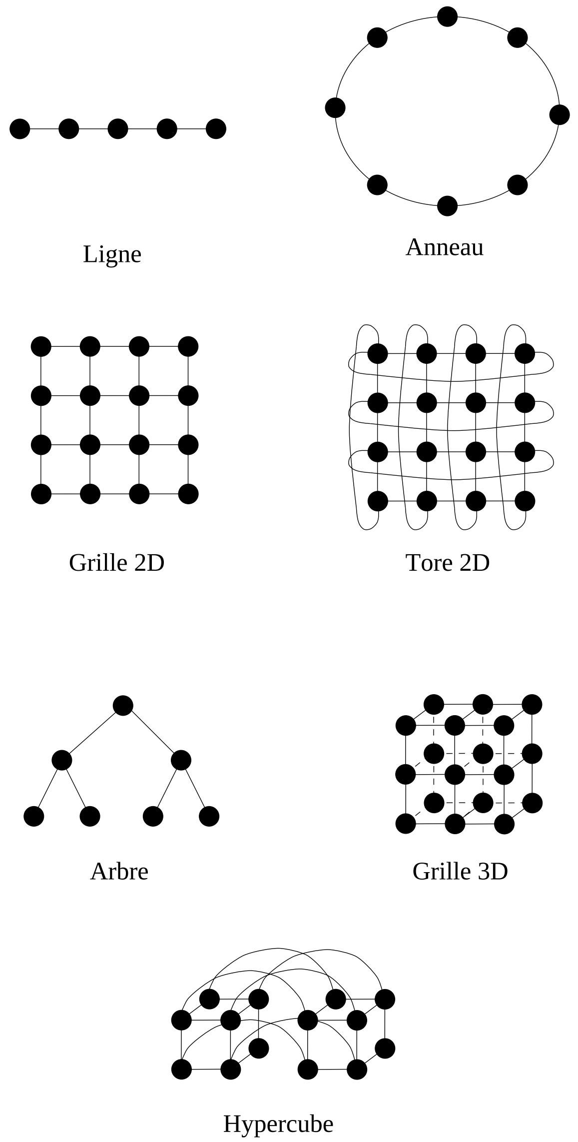


Figure 2.1: Quelques topologies régulières

Plusieurs paramètres caractérisent une topologie:

- **Diamètre:** C'est la distance maximale qui sépare les deux processeurs les plus éloignés l'un de l'autre dans une topologie donnée. Ce paramètre est exprimé en nombre de routeurs traversés.

- **Distance moyenne:** C'est la distance moyenne entre deux processeurs du réseau. C'est la moyenne arithmétique de la taille de tous les chemins du réseau.

- **Connectivité:** C'est le nombre maximal de ports par noeud que requiert une topologie.

- **Incrément minimal:** C'est le nombre minimal de processeurs (et aussi de routeurs) qu'on doit ajouter pour passer d'une taille donnée à la taille immédiatement supérieure d'une topologie régulière.

2.3 Schéma de routage

Les processeurs échangent des messages à travers le réseau. Chaque processeur peut injecter un message ayant n'importe quelle destination dans le réseau. Un message est injecté dans le réseau avec une adresse destination. Utilisant cette information, un message circule par bonds successifs d'un noeud (routeur) à l'autre (routeur) jusqu'a ce qu'il atteigne sa destination.

Le routage de ce message depuis le noeud source jusqu'au noeud destination nécessite la mise au point d'un plan de route. Ce plan de route doit être complet et pouvoir acheminer tout message émanant de n'importe quelle source et se dirigeant vers n'importe quelle destination. Ce plan de route s'appelle **schéma de routage**. Le schéma de routage représente l'information de direction nécessaire à l'acheminement des messages à travers le réseau. Au niveau de chaque noeud (routeur), il permet de déterminer le ou les ports de sortie que doit emprunter un message pour arriver à sa destination. La suite des ports de sortie empruntés successivement par un message se nomme **chemin de routage**.

Un message est généralement une suite de caractères. On appelle **en-tête** du message l'ensemble des caractères qui définit l'adresse destination du message.

Les schémas de routage sont **déterministes** ou **adaptatifs**. Un schéma de routage déterministe établit un seul chemin de routage possible entre deux noeuds. Par contre, un schéma de routage adaptatif fournit plusieurs chemins possibles entre deux noeuds. L'adaptativité permet de mieux répartir le trafic dans le réseau et ainsi de réduire les phénomènes de congestion.

Il existe deux types de schémas de routage: les schéma de routage **explicites** et les schémas de routage **distribués**.

2.3.1 Schéma de routage explicite

Dans les schémas de routage explicites, le schéma de routage est confiné au niveau des processeurs. On dresse une table dans laquelle sont décrits tous les chemins entre ce processeur et les autres processeurs du réseau. L'en-tête du message contient alors la liste des ports de sortie successifs décrivant le chemin entre la source et la destination. Le premier routeur sur le chemin du message consulte l'en-tête du message et en extrait le premier élément de la liste des ports de sortie. Cet élément contient le numéro du port de sortie par lequel on doit routeur ce message. Le i-ème routeur sur le chemin du message extrait le i-ème élément de l'en-tête. De même, cet élément donne le numéro du port de sortie par lequel doit sortir ce message.

L'opération d'extraction des numéros de ports de sortie à partir de l'en-tête peut être réalisée de diverses manières. La technique de suppression de l'en-tête (*header deletion*) simplifie considérablement cette opération. Ainsi, chaque routeur prend le premier élément de la liste et l'utilise pour attribuer au message le port de sortie correspondant. Ensuite, cet élément est détruit et le message envoyé au routeur suivant arrive avec l'élément suivant comme premier élément de la liste. Cette opération est répétée jusqu'a ce que le dernier élément est détruit ce qui correspond à l'arrivée du message à sa destination. Une autre technique consiste à effectuer un décalage circulaire de la liste de manière a mettre toujours l'élément suivant en tête de liste.

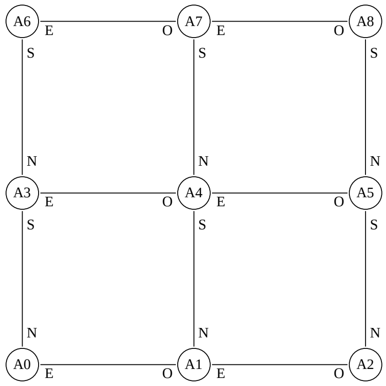


Figure 2.2: Exemple de schéma de routage explicite pour une grille 2D

Dans ce type de schéma de routage, on implémente l'adaptativité en mémorisant explicitement au niveau de la source plusieurs chemins permettant d'atteindre une destination donnée. A chaque envoi d'un message vers cette même destination, on choisit un chemin parmi les différents chemins disponibles. Celaà produit un résultat adaptatif dans le temps. Nous qualifions celaà d'adaptativité statique.

La figure 2.2 présente une grille à 2 dimensions de 3×3 noeuds. Chaque noeud comporte quatre ports (les noeuds extrémités ont certains ports inutilisés). Les ports sont identifiés par les quatre points cardinaux (Nord, Sud, Est et Ouest). Le tableau 2.1 donne un extrait du contenu de la table de routage explicite au niveau du noeud A_2.

Noeud émetteur	Noeud récepteur	Chemins possibles
A_2	A_0	(O,O)
	A_1	(O)
	A_3	(N,O,O);(O,N,O);(O,O,N)
	A_4	(N,O);(O,N)
	A_5	(N);(O,N,E)
	A_6	(N,N,O,O);(N,O,N,O);(O,N,O,N)
	A_7	(N,N,O);(O,N,N)
	A_8	(N,N)

Table 2.1: Extrait d'une table de routage en routage explicite

2.3.2 Schéma de routage distribué

Dans ce type de schéma de routage, l'information de routage est distribuée dans l'ensemble des routeurs. Les processeurs injectent dans le réseau des messages dont l'en-tête contient l'adresse du noeud destination. Le routeur utilise cette adresse pour sélectionner le (ou les) port(s) de sortie par lequel le message peut être envoyé au routeur suivant (ou acheminé vers le processeur destination). Chaque routeur dispose, pour celaà, d'une structure qui indique l'ensemble des ports de sortie associés à toute destination dans le réseau. L'association entre une adresse destination et un ensemble de ports de sortie peut dépendre uniquement de cette adresse. Dans ce cas, on associe à un message arrivant au routeur à travers n'importe quel port d'entrée toujours le même ensemble de ports de sortie. Le schéma de routage est dit **dépendant du noeud**. Dans le cas général, l'ensemble des ports de sortie associés à une destination donnée peut dépendre du port d'entrée par lequel le message arrive. Dans ce cas, le schéma de routage est dit **dépendant du port d'entrée**.

Au niveau des schémas de routage distribués, l'adaptativité est implémentée de manière dynamique. En effet, le chemin qu'empruntera un message dépends de conditions dynamiques que constitue l'état des ports de sortie à chaque instant. Le message choisit le premier port de sortie disponible parmi l'ensemble de ports de sortie associé à sa destination. Le chemin est ainsi déterminé étape par étape au fur et à mesure que le message avance vers sa destination. Celaà conduit au fait que deux messages partant de la même source peuvent emprunter des chemins différent pour arriver à la même destination.

Il existe deux manières d'implémenter les structures de données destinées à contenir les informations de routage au niveau des routeurs. Le schéma de routage établit une correspondance entre chaque destination et le ports de sortie à travers lesquels il est possible d'atteindre cette destination. Implicitement, il dresse donc une table de correspondance entre toutes les adresses destination et les ports de sortie qui leur sont associés. Cette table possède la même taille que l'espace d'adressage et est indexée par l'adresse destination. Chaque entrée contient les ports de sortie associés à cette destination. On appelle cette table: la **table complète**.

Cette représentation intuitive présente deux inconvénients majeurs. Le premier est sa taille considérable qui entraine un coût important au niveau des structures matérielles. Le second inconvénient est la limitation de l'espace d'adressage. La cause est qu'une fois la table implémentée dans le routeur, sa taille ne peut être étendue interdisant de ce fait d'étendre le réseau.

Plusieurs techniques de compaction de cette table ont été développées.

Schéma de routage "câblé"

Pour certains schémas de routage, il est possible de définir une fonction qui calcule les ports de sortie suivant la destination d'un message. En effet, lorsque la topologie est fixe et régulière, le schéma de routage présente, dans la plupart des cas, des caractéristiques de régularité aussi. On traduit cette régularité du schéma de routage en mettant en évidence une relation établie entre l'ensemble des destinations et l'ensemble des ports de sortie. La réalisation d'un circuit qui reproduit le fonctionnement de cette fonction constitue une façon de compacter l'information de routage.

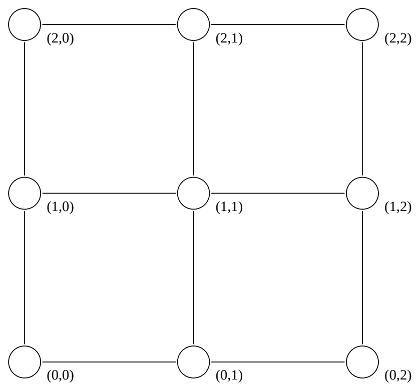


Figure 2.3: Exemple de schéma de routage "câblé" pour une grille 2D

La figure 2.3 présente un schéma de routage "câblé" pour la grille 2D. Chaque noeud est identifié par un couple (x, y). x représente l'abscisse du noeud dans la grille et y représente l'ordonnée. On considère un schéma de routage déterministe. Le routage entre deux noeud s'effectue d'abord verticalement jusqu'a atteindre la ligne qui contient le noeud destination. Ensuite, on effectue un routage horizontal pour acheminer le message jusqu'a sa destination. Soit (x_c, y_c) l'adresse du noeud courant et (x_d, y_d) l'adresse du noeud destination. On définit une fonction de routage $R(x_c, y_c, x_d, y_d)$ de la manière suivante:

- Si $x_c > x_d$ alors emprunter le lien Sud.
- Si $x_c < x_d$ alors emprunter le lien Nord.

- Si $x_c = x_d$
  - si $y_c = y_d$ alors délivrer le message vers le processeur local.
  - si $y_c < y_d$ alors emprunter le lien Est
  - si $y_c > y_d$ alors emprunter le lien Ouest

Une implémentation matérielle de cette fonction constitue une compaction de l'information de routage.

Schéma de routage par intervalles (*interval labelling*)

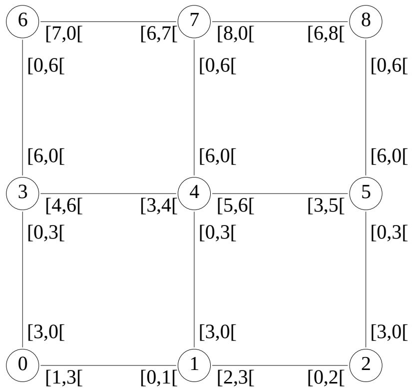


Figure 2.4: Schéma de routage par intervalle d'une grille 2D

A partir d'une table de routage complète, regroupons dans un même groupe les destinations qui possèdent les mêmes ports de sortie. On rappelle qu'on assigne à chaque processeur une adresse d'identification unique. On obtient un schéma de routage par intervalles en choisissant les adresses des processeurs de manière à obtenir le maximum de contiguoté entre les adresses d'un même groupe. On peut alors coder un groupe d'adresses par un ou plusieurs intervalles. Cette technique a été mise au point par Santoro et Khatib [3].

La figure 2.4 représente un schéma de routage par intervalle d'une grille 2D. Un seul intervalle est associé a chaque port de sortie.

Il faut noter que dans cette approche, les adresses des processeurs sont imposées par la topologie.

Considérons un schéma de routage où chaque port de sortie d'un routeur est associé à une table contenant les adresses des noeuds pouvant être atteints à travers ce port, tel que:

- La table d'un port terminal du routeur vers le processeur local doit contenir uniquement l'adresse de ce processeur.

- L'union de toutes les tables associées aux ports de sortie d'un même routeur correspond à l'espace d'adressage.

Un schéma de routage par intervalle est obtenu si chaque table associée à un lien de sortie peut être représentée par un ou plusieurs intervalles éventuellement cycliques. On aboutit, dans la plupart des cas, à associer un seul intervalle à un port de sortie. L'opération qui associe une borne inférieure d'un intervalle à un port de sorties nomme **étiquetage du lien connecté à ce port**.

2.4 Technique de routage

Tout processeur émet et reçoit des messages à travers le réseau. Le message comprend une partie en-tête contenant des informations d'indication sur sa destination (adresse destination dans le cas d'un schéma de routage distribué; suite de ports de sortie dans le cas d'un schéma de routage explicite). Le message est terminé par une marque indiquant sa fin. La technique de routage détermine la façon dont les messages sont transmis de l'émetteur vers le récepteur. Il existe trois techniques de routage: La technique *store and forward*, la technique *virtual cut-through* et la technique *wormhole*. Ces techniques établissent des règles sur la manière dont un message est transporté le long du chemin vers sa destination. On les appelle à ce titre techniques de **commutation de messages (ou paquets)**.

La technique *store-and-forward* consiste à stocker entièrement le message dans chacun des routeurs intermédiaires avant de le réémettre. Il est alors possible d'effectuer un contrôle d'intégrité des messages dans chaque routeur. Par contre la latence est directement proportionnelle au nombre de routeurs traversés.

Afin de diminuer la latence lorsqu'il n'y a pas de contention, Kermani et Kleinrock [4] ont introduit la technique *virtual-cut-through*. Dans cette technique, l'émission d'un message n'est pas subordonnée à sa complète réception si la sortie qu'il doit emprunter est libre. Par contre il faut pouvoir stocker entièrement le message dans le routeur s'il ne peut être routé immédiatement. Cela impose que les messages aient une taille limitée. La latence totale de routage d'un message est réduite grâce à l'effet de pipe-line.

Deux chercheurs du Caltech: Dally et Seitz[5] ont proposé la technique de routage *wormhole* dans laquelle le message est divisé en unités insécables appelées *flit* pour *Flow Control Unit* composées de un ou plusieurs caractères. Le *flit* de tête contient la destination du message et progresse de routeur en routeur en entrainant à sa suite les *flits* constituant le reste du message. Le chemin créé par le *flit* de tête se referme derrière la queue du message au fur et à mesure de la progression de la queue du message, permettant ainsi aux canaux libérés d'être utilisés par d'autres messages. La décision de routage est prise dès que le *flit* de tête arrive sur un port d'entrée du routeur. Un message peut donc être réparti sur plusieurs routeurs à un moment donné (figure 2.5) et la tête d'un message peut arriver à destination avant que la fin n'ait été émise par la source. La latence est donc diminuée et il suffit que chaque routeur ait la place suffisante pour stocker au moins un *flit*. En contrepartie, le contrôle d'intégrité des messages ne peut plus se faire au niveau de chaque routeur. Enfin, en cas de contention, le message reste bloqué interdisant à d'autres messages d'utiliser les canaux qu'il occupe.

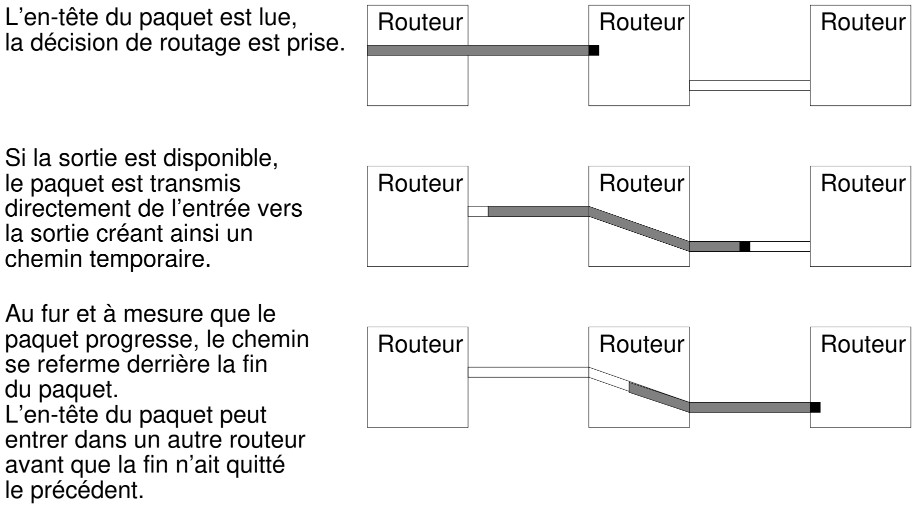


Figure 2.5: Routage *wormhole*

Le tableau 2.2 résume les différentes caractéristiques de ces techniques de routage en terme de latence, mémorisation et contrôle d'intégrité.

Les paramètres utilisés sont:

- $\mathbf{L_h}$: longueur de l'en-tête,
- **L**: longueur du message (ou du paquet),
- **B**: bande passante de la liaison
- **D**: nombre de routeurs traversés.

Technique de routage	Latence sans contention	Mémorisation minimale	Cont rôle d'in tégrité
store-and-forward	$\frac{L+L_h}{B} \times D$	1 message	niveau routeur
virtual-cut-through	$\frac{L_h}{B} \times D + \frac{L}{B}$	1 message	niveau récepteur
wormhole	$\frac{L_h}{B} \times D + \frac{L}{B}$	1 flit	niveau ré cepteur

Table 2.2: Techniques de routage de type commutation de messages

2.5 Contrôle de flux

Un réseau à commutation de messages consiste en un ensemble de ressources (liaisons, tampons mémoire) qui doit être partagé dynamiquement entre plusieurs nœuds désirant communiquer les uns avec les autres. Si l'attribution dynamique des ressources offre un plus grande flexibilité et de meilleures performances que l'approche commutation de circuit, cela s'obtient au prix d'un coût qui est le contrôle de l'allocation de ces ressources. Le nombre des ressources et la taille de chaque ressource à partager sont fixés à la définition de l'architecture du routeur. Par conséquent, le noeud courant doit être en mesure de déterminer que l'envoi d'un message (ou d'une partie d'un message) vers le noeud suivant ne générera pas de conflit sur les ressources. En effet, en absence de contrôle de l'allocation des ressources, la transmission incontrôlée des messages peut créer une perte d'information. Chaque noeud doit donc en permanence communiquer les informations sur son état courant à ses voisins. Cependant, cette connaissance de l'état du réseau n'est pas instantanée et consomme de la bande passante puisqu'il faut véhiculer ces informations. L'ensemble des techniques et mécanismes de transmission et de traitement de ces informations de service se nomme **contrôle de flux**. Le rôle de ces techniques est:

- éviter les pertes d'informations,
- prévenir la dégradation du débit global et de l'efficacité du réseau dues à la surcharge,
- allouer de façon équitable les ressources aux différents utilisateurs du réseau (prévention des famines),
- ajuster les vitesses de fonctionnement entre le réseau et les nœuds qui y sont attachés.

Afin de réaliser ces fonctions, les procédures de contrôle de flux incluent un ensemble de contraintes sur, par exemple, l'allocation des liaisons, des tampons mémoire, sur le taux d'émission de messages par les noeuds etc... Ces contraintes ont pour but de limiter l'accès au réseau ou à certaines parties du réseau pour prévenir la surcharge, de conditionner l'allocation de ressources pour éviter les interblocages ou restreindre la quantité de ressources que peut acquérir un message afin de prévenir les famines. Ces contraintes peuvent être fixes ou ajustées dynamiquement selon le trafic.

Le contrôle de flux peut s'exercer à plusieurs niveaux. Ainsi on peut distinguer:

- le niveau liaison: il assure que toute donnée émise pourra être stockée par le récepteur si elle ne peut être routée immédiatement. Ce contrôle s'effectue sur la base d'unité de contrôle de flux qui peut être l'octet, le *flit* en routage *wormhole* ou le message en routage *store-and-forward*. Cette unité de contrôle de flux est la plus petite unité indivisible qui peut être acceptée par le récepteur.
- le niveau message: il définit le format et le protocole d'échanges des messages entre nœuds. Il vérifie que les messages reçus sont complets, non corrompus et demande le cas échéant la retransmission du message.
- le niveau accès au réseau: il régule le taux d'émission des nœuds attachés au réseau afin de ne pas le surcharger. Ceci est effectué sur la base d'information envoyées explicitement par les routeurs (par ex: taux d'occupation des tampons, des liaisons) ou par effet rétroactif du contrôle de flux au niveau liaison.

CHAPITRE 3

PROBLÉMATIQUE

3.1 Introduction

Ce travail se propose de fournir des outils pour construire les réseaux d'interconnexion pour des machines parallèles qui sont des réseaux de stations de travail. Le routeur RCUBE et la machine MPC, développé en en cours de développement au sein du département ASIM du LIP6, sont les principaux axes autour desquelles gravitent tous les éléments de cette étude.

3.2 Propriétés requises pour un réseau d'interconnexion

Les objectifs principaux qu'on doit chercher à atteindre lors de la conception d'un réseau d'interconnexion sont:

- Performances: C'est la propriété principale d'un réseau d'interconnexion. Toute conception a pour but d'aboutir aux meilleurs performances possibles. Deux paramètres permettent de mesurer les performances:
  - Latence: c'est le délai d'acheminement d'un message depuis sa source jusqu'a sa destination. Ce délai inclut tous les temps d'attente dûs à la manière dont les messages avancent dans un réseau distribué. La *latence moyenne* est la principale mesure caractérisant un réseau d'interconnexion.
  - Débit: c'est la quantité d'informations que le réseau transporte par unité de temps. Ce paramètre est généralement exprimé en fraction de la bande passante théorique maximale.

- Fiabilité: Le réseau d'interconnexion doit offrir une garantie de fonctionnement correct. Même si le taux de pannes est faible, des mécanismes de détection d'erreurs doivent exister, car la probabilité d'erreurs augmente rapidement avec le nombre de noeuds. Il doit également supporter des mécanismes de récupération en cas d'erreurs pour éviter de réinitialiser la machine à chaque panne.

- Coût: Il faut éviter que le faible coût associé à l'utilisation de noeuds de calcul standard soit compromis par le coût élevé du réseau d'interconnexion.

Ainsi, il est important de dégager les propriétés d'un réseau d'interconnexion en cernant celles de chacune de ses composantes.

- Les routeurs:
  Il est important de noter que parmi les 3 composantes du réseau d'interconnexion, les routeurs sont les seuls éléments matériels. La topologie et le schéma de routage peuvent et doivent être redéfinis pour chaque nouveau réseau. L'architecture du routeur a ainsi un impact direct sur la flexibilité (possibilité de reconfigurer le réseau), l'extensibilité (possibilité d'augmenter le nombre de processeurs), les performances et la fiabilité du réseau d'interconnexion. Elle introduit par ailleurs une série de contraintes et de restrictions sur la topologie et le schéma de routage.

- La topologie:
  Il est utile de définir ses caractéristiques.

  – Existence d'un schéma de routage: Les deux seules conditions qu'une topologie doit remplir pour qu'elle puisse admettre un schéma de routage sont:
    * Connexité: Une topologie formée de plusieurs groupes de noeuds complètement disjoints n'admettera jamais de schéma de routage global.
    * Absence de boucle: Un lien qui connecte deux ports d'un même routeur est inutile.
  – Connectivité: C'est la valence minimale du routeur pour qu'il puisse être possible de construire la topologie en question.
  – Extensibilité: C'est la possibilité de construire une version de taille supérieure tout en gardant la même structure que la topologie initiale.
  – Coût: Il se décompose comme suit:
    * nombre de routeurs nécessaires pour connecter n processeurs (pour établir une comparaison statique entre plusieurs topologies).
    * Taille de l'incrément minimal lors de l'extension.
    * Nombre de ports utilisés (connectivité).

- Schéma de routage:
  Il est difficile de séparer le choix de la topologie et du schéma de routage. Celui-ci doit satisfaire à un certain nombre d'exigences:

- Exactitude: C'est la capacité d'un schéma de routage de router tout message émis de n'importe quelle source vers sa destination. Il ne doit pas se tromper de destinataire.
- Validité: C'est la garantie que le schéma de routage ne conduit pas vers des situations d'interblocage et/ou de boucle infinie.
- Codabilité: C'est la possibilité pour celui-ci d'être codé de manière à ce qu'il puisse être contenu dans les structures de données du routeur.
- Adaptativité (cette caractéristique est optionnelle): Un schéma de routage adaptatif est un schéma qui permet d'avoir plusieurs chemins possibles entre deux noeuds.

3.3 Contexte

3.3.1 Le routeur RCUBE

RCUBE [1] [2] est un routeur de paquets à hautes performances intégrant 8 liaisons série point-à-point asynchrones bidirectionnelles à 1 Gbit/s conformes au standard IEEE 1355 [6].

RCUBE possède 8 liens de communication connectés par un crossbar non-bloquant. Un lien est composé de deux ports à chaque extrémité. L'un en entrée et l'autre en sortie.

RCUBE route des paquets de taille quelconque. Un paquet est constitué d'une entête contenant les informations sur sa destination suivie par une séquence de caractères composant les informations utiles du paquet terminées par un caractère de fin de paquet.

Les paquets sont routés suivant une technique dite *wormhole*. Suivant cette technique, l'en-tête du paquet peut quitter un noeud n pour atteindre un noeud $n+1$ sans attendre que le corps du paquet ait été entièrement reçu dans le noeud n. L'en-tête est utilisée pour déterminer , à partir du schéma de routage, les ports de sortie que le paquet pourra emprunter. Si l'un de ces ports est libre, il sera alloué au paquet et l'en-tête est autorisée à progresser. Dans ce cas, le reste du paquet (qui peut être distribué sur plusieurs routeurs déjà traversés par l'en-tête) progresse suivant un schéma pipe-line. Si tous les ports de sortie sont indisponibles, l'en-tête reste bloquée en attente de la libération de l'un d'eux. Dans ce cas, le reste du paquet se bloque aussi. Par ailleurs, tous les ports de sortie alloués à une en-tête de paquet le restent tant qu'ils n'ont pas été traversés par la marque de fin de paquet.

Un paquet est ainsi distribué sur plusieurs routeurs. Pour éviter les pertes d'informations, il existe un mécanisme de contrôle de flux au niveau matériel utilisant une technique de crédit: un crédit représente 32 caractères. Le port de sortie d'un lien contient un compteur représentant le nombre de caractères qu'il peut envoyer. Il

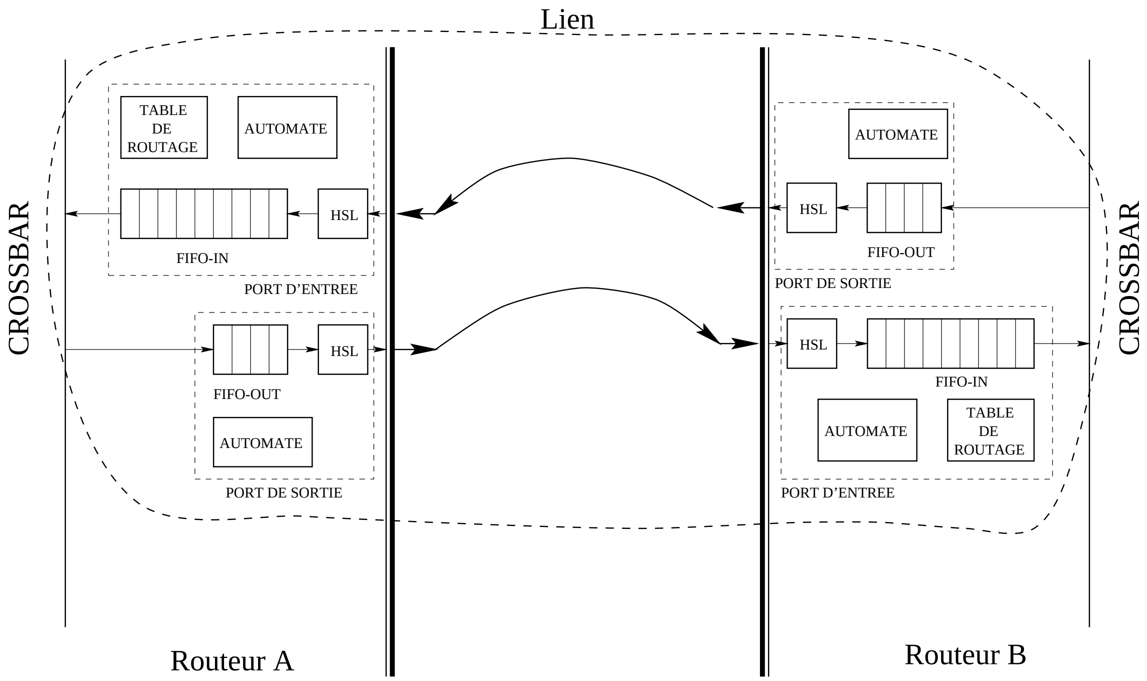


Figure 3.1: Définition du lien et du port dans RCUBE

est decrémenté d'une unité à chaque envoi d'un caractère. S'il atteint la valeur 0, le port associé se bloque. Le port d'entrée contient aussi un compteur donnant le nombre de caractères lus de la FIFO d'entrée. Il est incrémenté d'une unité à chaque lecture. Quand la cellule d'entrée aura consommée (routé) 32 caractères, un caractère spécial nommé FCC *Flow Control Character* est inséré dans le flux de données en sens inverse. Celui-ci n'est pas comptabilisé comme caractère transmis. Quand un port d'entrée reçoit un caractère FCC le port de sortie correspondant incrémente son compteur de crédit de 32.

Le schéma de routage utilise une technique par intervalles pour représenter les tables de routage. A chaque noeud de calcul (processeur) est attribuée une adresse. Le codage par intervalles permet d'associer à un intervalle formé d'adresses contiguës une même liste de ports de sortie limitant la taille des structures de données nécessaires à la représentation du schéma de routage. RCUBE possède 8 tables de routage indépendantes, une par port d'entrée. Chacune d'elles possède 8 intervalles. A chaque intervalle est associée une liste de ports de sortie possibles (8 au maximum) permettant de supporter des schémas de routage adaptatifs. RCUBE implémente un autre type de schéma de routage: le routage préfixé. Dans ce mode, le réseau est logiquement divisé en sous-réseaux. L'adresse de destination est composée d'une liste d'adresse. Chacune d'elles est utilisée pour le routage dans chaque sous-réseau traversé. Pour supporter ce type de schéma de routage chaque port de sortie peut être programmé pour détruire la première adresse de la liste (*Header deletion*).

En résumé RCUBE impose les contraintes suivantes sur les topologies et les schémas de routage:

- RCUBE possédant 8 liens, on ne peut construire avec des RCUBES que des topologies ayant un nombre maximal de liens par noeud inférieur ou égal à 8.

- Le processus de contrôle de flux au niveau du lien impose que les topologies compatibles avec RCUBE doivent obligatoirement utiliser des liens bidirectionnels.

- Les schémas de routage qu'un réseau de RCUBE peut supporter doivent avoir les caractéristiques suivantes:

  – La technique de routage étant le wormhole, il n'y a aucune contrainte sur la taille des messages.
  – Comme RCUBE possède une table de routage pour chaque entrée, il n'y a aucune contrainte sur les schémas de routage car c'est le cas le plus général.
  – Chaque table de routage utilise un codage par intervalles à 8 intervalles au maximum. Tous les schémas de routage nécessitant un codage de plus de 8 intervalles sont exclus.

3.3.2 La machine Multi-PC

La machine Multi-PC est une machine parallèle à passage de messages utilisant comme noeuds de calcul des cartes mères de PC standard. Des routeurs RCUBE sont utilisés pour réaliser l'interconnexion entre les noeuds.

Chaque processeur communique avec le réseau HSL via une interface dédiée qui établit le lien entre le bus PCI et un routeur RCUBE. Une carte d'extension regroupe un circuit spécialisé réalisant l'interface: le composant PCI-DDC [7] et le routeur RCUBE. Ce composant accède en mode maître sur le bus PCI et fournit une primitive d'ecriture directe en mémoire distante sans recopie. Les périphériques connectés sur le bus PCI ne sont pas autorisés à détenir le bus au delà d'un certain temps.

Le circuit d'interface PCI-RCUBE peut perdre le contrôle du bus PCI à tout moment. Un message représentant normalement un transfert d'une quantité mémoire donnée peut donc être acheminé en plusieurs paquets. Chacun d'eux correspond à une phase de transfert interrompue par la perte du contrôle du bus PCI. Comme, le message en question constitue une entité logique unique, il est impératif de pouvoir le reconstituer à l'arrivée. On le fait à l'aide du mécanisme décrit ci-dessous:

- Afin de les identifier, tous les paquets d'un même message possèdent un champs de leur en-tête qui est identique: c'est le champs MI (*Message Identifier*).

- Le dernier paquet est marqué. Il contient le nombre total de paquets constituant le message.

Conséquence pour cette étude:

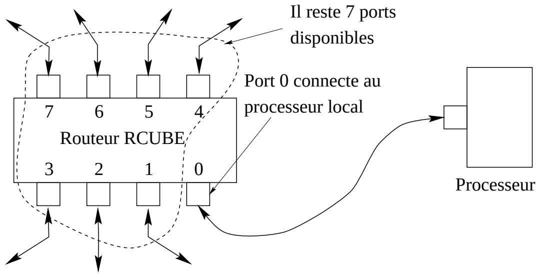


Figure 3.2: Valence 7

- Les noeuds de la machine MPC ont une valence 7.
- Il existe un routeur par processeur

3.4 Objectifs de l'étude

Dans cette thèse, nous nous sommes intéressés à trois problèmes:

- Évaluation des performances des différents réseaux d'interconnexions utilisant RCUBE, pour différentes topologies et différents schémas de routage. Ici, on cherche à utiliser une méthode applicable pour d'autres routeurs que RCUBE. Ce point est abordé dans les chapitres 4 et 5.
- Configuration automatique de topologies quelconques On cherche ici à définir un algorithme de calcul des tables de routage d'un réseau d'interconnexion utilisant le routage par intervalles, pour n'importe quelle topologie régulière ou irrégulière. Ce point fait l'objet du chapitre 6.
- Définition d'une méthode générale de vérification de la validité d'un schéma de routage. Cette méthode doit être indépendante de la topologie choisie. Ce point est traité dans le chapitre 7.

3.4.1 Détermination de topologies et de schémas de routage performants

Dans notre contexte, le routeur RCUBE impose des contraintes au niveau du choix des topologies et schémas de routage. Il s'agit de déterminer parmi les topologies et schémas de routage conformes aux contraintes imposées par l'utilisation de RCUBE ceux qui présentent les meilleures performances. On vise une meilleure compréhension du comportement des réseaux de communications construits avec des routeurs RCUBE

La machine Multi-PC impose quant à elle des contraintes supplémentaires. La carte d'interface entre le PC et RCUBE s'insère dans une carte mère de PC. L'un des ports de RCUBE est utilisé de manière spéciale. Il est relié au composant d'interface et c'est par lui que transitent les données destinées au processeur local. Cette configuration conduit vers plusieurs situations:

- L'ensemble de topologies les plus évidentes sont celles où chaque routeur est connecté à un processeur. Ces topologies sont construites en interconnectant les 7 ports restant de chaque routeur.
- Il est possible d'utiliser les 8 liens de certains routeurs en considérant que les cartes mères sur lesquelles ils sont connectés ne sont utilisées que pour configurer les tables de routage.
- Il n'est pas possible de connecter directement plusieurs processeurs à un même routeur.

A la lumière de ce qui vient d'être dit, il s'agit de déterminer quelles sont les topologies et schémas de routage à valence 7 qui sont les plus performants. Il est souhaitable de le faire pour toutes les tailles de réseau susceptible d'être à la base d'une architecture de type Multi-PC.

La fragmentation des messages en paquets est une caractéristique de l'interface PCI-DDC. Dans un réseau déterministe (non-adaptatif), puisque le chemin entre deux noeuds est unique, la précédence entre les paquets est garantie. Dans un réseau adaptatif elle ne l'est plus. A cause de cette différence, le protocole logiciel qui gère les échanges dans un tel contexte change considérablement suivant le type de réseau dans lequel on se trouve. Dans le cas de réseaux adaptatifs, celui-ci devient plus complexe. Il est donc utile de quantifier le gain en performances qu'apporte l'adaptativité dans certaines topologies pour fournir un élément d'appreciation précis au concepteur.

3.4.2 Configuration automatique de topologies quelconques

Dans le contexte de la machine Multi-PC, toute topologie utilisant des noeuds de valence inférieure ou égale à 7 est réalisable. La construction de schémas de routage adaptés est nécessaire pour pouvoir les exploiter. Il faut donc définir une méthode qui permet de calculer les tables de routage pour une topologie quelconque.

Le réseau d'interconnexion est une ressource partagée distribuée. Un système distribué est sujet aux pannes plus fréquemment que chaque composant pris à part. La gestion des pannes est donc une tâche importante. Deux types de pannes peuvent survenir:

1) La panne temporaire ou intermittente qui a pour effet la modification des données acheminées par le réseau. Ces erreurs sont détectables soit au niveau de chaque routeur, grâce aux bits de parité, soit au niveau des interfaces avec les noeuds de calcul grâce aux techniques de CRC.

2) La défaillance définitive d'un lien ou d'un noeud qui implique la deconnexion du reste du réseau.

Le second type de panne crée une situation nouvelle. La topologie initiale doit être modifiée par la suppression d'un lien ou d'un noeud ce qui fait que le schéma de routage n'est plus valable. La topologie initiale qui est généralement une topologie régulière devient une topologie quelconque.

Ainsi, la réalisation de topologies à valence 7 et la reconfiguration de topologies en panne requièrent la définition d'une méthode qui permet de calculer les tables de routage pour une topologie quelconque.

3.4.3 Vérification de la validité d'un schéma de routage

Deux types de situations particulièrement critiques peuvent arriver lors de l' exploitation d'un réseau sous un schéma de routage erroné: l'interblocage (*deadlock*) et la boucle infinie (*livelock*).

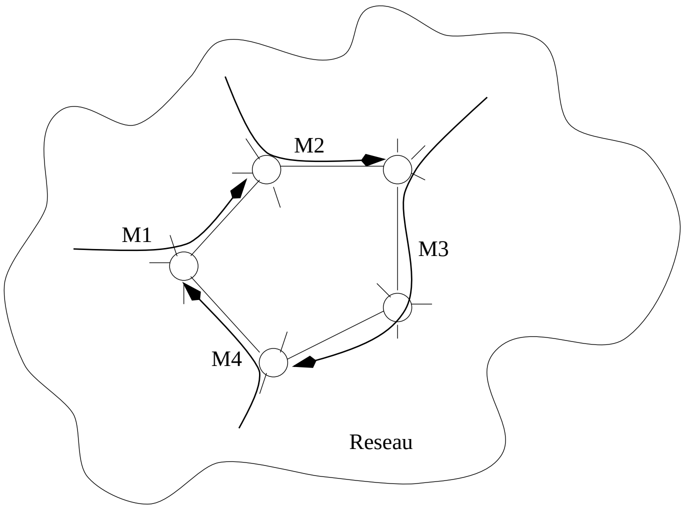


Figure 3.3: Situation d'interblocage

Un réseau d'interconnexion où les messages progressent suivant une technique de type wormhole autorise un message à détenir plusieurs ressources (liens) tout en ayant la possibilité d'en demander d'autres. De ce fait, la progression d'un message M_1 qui a fait la requête d'un lien l déjà occupé par un message M_2 ne peut se faire qu'après la libération de l par ce dernier. Ce mode de fonctionnement peut mener vers des situations où un message est bloqué en attente de ressources détenues par un autre message bloqué aussi en attente de ressources dont la libération est causée transitivement par le premier. Ceci crée une situation de blocage mutuel ou interblocage (voir Figure 3.3).

Un autre type de malfonctionnement arrive lorsqu'un message est indéfiniment routé à travers le réseau sans jamais atteindre sa destination. Son cheminement se comporte comme une boucle infinie.

Un schéma de routage est exact si il ne route pas des messages vers des destinations erronées. Il est valide s'il le fait sans jamais aboutir à une situation d'interblocage et/ou de boucle infinie.

Il existe déjà plusieurs travaux qui ont traité le problème de l'interblocage. Dans [8], seuls les schémas de routage non-adaptatifs ont été considérés. [9] et [10], ont proposé une condition nécessaire et suffisante pour qu'un schéma de routage n'admette pas d'interblocage. Leurs résultats s'appliquent aux schémas de routage adaptatifs mais ne prennent pas en compte les schémas de routage où la décision dépend du port d'entrée. Dans notre cas, ceci constitue une restriction puisque le routeur RCUBE permet d'avoir des tables de routage différentes pour chaque entrée.

Notre est but est de vérifier la validité et l'exactitude d'un schéma de routage donné dans un contexte pratique comme le notre. Il s'agit donc de développer une théorie générale prouvant qu'un schéma de routage est valide. C'est à dire d'abord exact et ensuite non sujet à l'interblocage et à la boucle infinie. De plus cette théorie devra pouvoir aboutir à une méthode de vérification formelle automatique.

3.5 Conclusion

La construction d'un réseau d'interconnexion conforme aux exigences de flexibilité, d'extensibilité, de performances, de fiabilité et de coût ne peut se faire qu'à travers la résolution des questions suivantes:

1) Détermination de topologies et de schémas de routage performants:

   - Quels sont les topologies et schémas de routage à valence 7 les plus performants pour des réseaux construits avec des routeurs RCUBE et ceci pour plusieurs tailles de réseaux ?
   - Que procure l'adaptativité en gains de performances ?
   - Quels sont les points forts de l'architecture de RCUBE ?
   - Quels sont les points nécessitant une amélioration ?

2) Configuration et reconfiguration en cas de panne: Peut-on définir une méthode générale permettant de configurer une topologie quelconque ?

3) Vérification de la validité d'un schéma de routage:

   - Peut-on définir une théorie générale qui permet de prouver qu'un schéma de routage est exact et non sujet à l'interblocage et à la boucle infinie ?

– Peut-on définir une méthode permettant de vérifier formellement et automatiquement la validité d'un schéma de routage ?

Cette thèse se propose de répondre à toutes ces questions.

CHAPITRE 4

RÉSEAUX D'INTERCONNEXION OPTIMAUX POUR LA MACHINE MPC

4.1 Introduction

L'objectif de ce chapitre est d'évaluer les performances de différents réseaux d'interconnexion utilisant la technologie HSL (*High Speed Link*) c'est à dire le routeur RCUBE [1] [2].

On cherchera en particulier à définir les meilleures topologies possibles pour la machine MPC. La machine MPC est une machine parallèle construite en interconnectant des cartes mères de PC. Chaque "noeud"d'une machine MPC est constituée d'une carte processeur et d'une carte d'interface réseau comportant un routeur RCUBE. Un noeud de la machine MPC possède donc une valence 7. Un autre objectif est d'évaluer

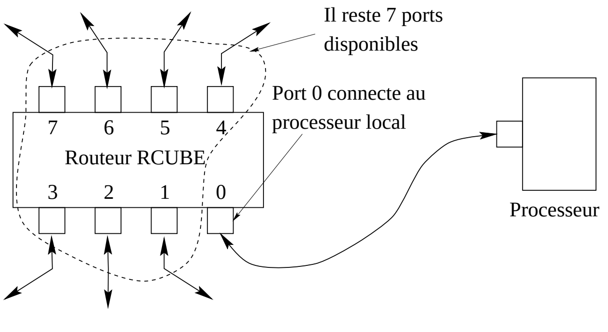


Figure 4.1: Valence 7

l'efficacité du mécanisme de routage adaptatif implémenté par le routeur RCUBE. Pour ces différentes études, nous avons été amenés à réaliser un outil spécifique: le simulateur MILE [11] [12] [13].

4.2 Environnement de simulation

4.2.1 Présentation

L'environnement de simulation MILE est dédié à l'évaluation des performances des réseaux d'interconnexion. C'est en fait un simulateur cycle par cycle d'automates communiquants interconnectés entre eux suivant une topologie donnée. La topologie du réseau est quelconque et peut être décrite de façon procédurale. Les automates sont décrits en langage **C**. MILE permet la simulation du comportement global d'une topologie et d'en extraire les mesures caractéristiques.

4.2.2 Architecture

L'architecture de MILE se compose de trois parties:

- Bibliothèque de composants définis par l'utilisateur. Chaque composant représente un noeud du réseau tel que processeur ou routeur. Un composant est défini par:
  - ses paramètres et attributs ainsi que son interface et sa représentation graphique(icône).
  - les procédures **C** qui décrivent son comportement.
- Un éditeur graphique EMILE: c'est un outil convivial pour la description de réseaux de petite taille et l'édition de leurs paramètres. Le format de fichier supporté est un sous-ensemble du langage **NDL**(Network Description Language) de SGS-THOMSON.
- Un simulateur SMILE: c'est un noyau de simulation séquentiel. Il existe sous deux formes différentes. La première utilise une interface graphique conviviale qui permet de dérouler les simulations dans un mode interactif. Un mode batch est également disponible.

4.2.3 Composants

Un composant est destiné à un réseau d'interconnexion en vue d'une simulation. Le concept retenu ici est similaire à celui des bibliothèques de composants dans les outils de simulation logique.

La spécification d'un composant est constituée de :

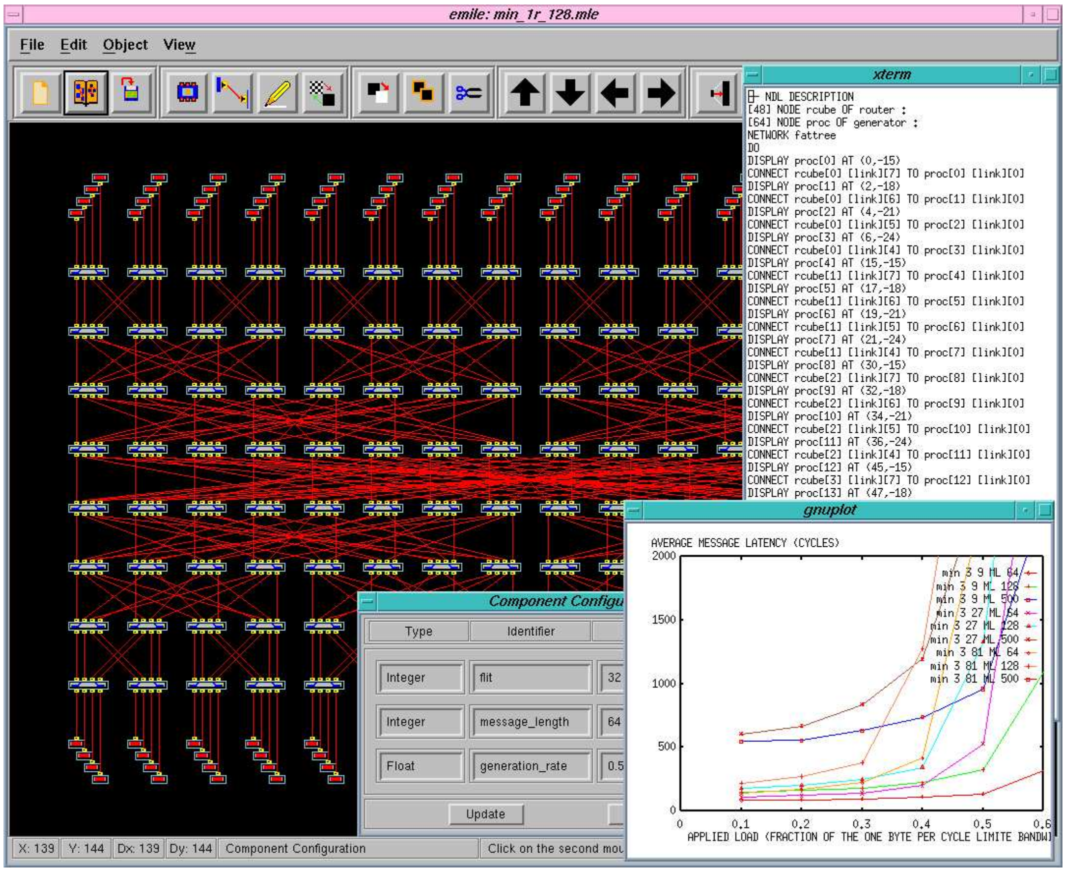


Figure 4.2: Vue générale de l'environnement MILE

- **Déclaration des attributs** consistant en:

  Interface : définit les ports d'entrée/sortie.

  Zone de configuration noeud : définit un groupe d'attributs pouvant être initialisés pour chaque instance d'un même composant: par exemple, deux routeurs distincts peuvent avoir des tables de routage différentes.

  Zone de configuration composant : définit un groupe d'attributs de configuration au niveau composant. Ces attributs sont principalement représentés par des paramètres structurels.

  Sondes : définit l'ensemble des sondes de mesure disponibles.

- **Description comportementale:** le comportement d'un composant est décrit par trois procédures: une procédure d'initialisation (le constructeur), une procédure d'exécution qui est appelée à chaque cycle pour chaque noeud du réseau et une procédure de terminaison (le destructeur).

MILE autorise la définition de sondes (*probes*) comme une partie du composant. Les sondes sont utilisées dans la description comportementale du composant pour faire des calculs coopératifs dans le but de réaliser des mesures. Il existe deux types de sondes: partagées ou privées. Une sonde partagée peut concerner plusieurs noeuds tandis qu'une sonde privée ne peut être associée qu'a une seule instance de composant.

On peut par exemple compter le nombre de paquets entrant par un port particulier d'un noeud particulier (sonde privée). On peut également compter le nombre total de paquets ayant atteint leur destination (sonde partagée). Les sondes "privées" sont implémentées comme des variables locales à un noeud et les sondes "partagées" sont implémentées comme des variables globales. Toutes les sondes doivent être décrites explicitement dans le modèle comportemental des noeuds instanciés dans le réseau (ce qui complique la description des modèles). Lors de la simulation, il est possible de n'activer que les sondes utiles aux mesures qu'on cherche à effectuer.

4.3 Modèles utilisés

Nous avons décrit deux modèles de composants un pour le routeur RCUBE et un modèle simple de processeur. On considère qu'un lien permet de transmettre un caractère de 9 bits à chaque cycle. Un caractère de 9 bits est représenté au niveau du simulateur MILE par deux octets (dans MILE l'unité d'allocation est l'octet). Le processeur est un émetteur-récepteur qui génère et consomme des messages. Il a été construit en conformité avec la méthode suivie dans [14].

4.3.1 Modèle du routeur

La figure 4.3 présente le fichier de définition des ressources du composant RCUBE. La section *interface* définit l'interface externe du composant et donne des indications utiles à sa représentation graphique. Cette section décrit les ports de communication. L'ensemble forme l'interface de communication du composant. Le premier paramètre dans une ligne correspond à la direction du port. Dans notre cas il a pour valeur *bidirectional* indiquant que le port est bidirectionnel. Un port peut être en entrée (*input*) ou en sortie (*output*). Le second paramètre correspond à la face cardinale sur laquelle est placé le port dans la représentation graphique du composant. Suivent l'identificateur du port et sa largeur en octets. L'identificateur est une chaine de caractères simple. On peut définir un identificateur comme un élément faisant partie d'un vecteur.

Les attributs sous la section *node configuration* sont ceux destinés à contenir les informations de configuration pour chaque instance de RCUBE.

- *header length* tableau déterminant pour chaque lien d'entrée la taille de l'en-tête du message (1 ou 2 octets).

- *header deletion* tableau spécifiant pour chaque port de sortie s'il est en mode de suppression d'en-tête ou pas (*header deletion mode*).

```
component router
begin

  interface
    bidirectional north link[0,3][2];
    bidirectional south link[4,7][2];
  end

  node configuration
   bit  header_length[8],
        header_deletion[8];
   int  intervals[8][8],
        lists[8][8][8];
  end

  component configuration
   int in_fifo_depth=64,
       outfifo_depth=5,
       flit=32;
  end

  probes
  end
end
```

Figure 4.3: Fichier de déclaration des ressources de RCUBE (.mmd)

- *intervals* A chaque port d'entrée est associée une table d'intervalles à 8 entrées. Chaque élément représente une borne d'un intervalle. La table doit être triée de manière croissante.

- *lists* Ce tableau réalise une correspondance entre chaque intervalle et une liste de liens de sortie possibles associés à cet intervalle. C'est un tableau à trois dimensions de 8x8x8 éléments correspondants à 8 ports de sortie pour chacun des 8 intervalles pour chacun des 8 ports d'entrée.

Les attributs de la section *component configuration* ont une portée globale et concernent toutes les instances de RCUBE. Le paramètre *in_fifo_depth* fixe la taille du tampon FIFO d'entrée de tous les ports de RCUBE. Le paramètre *outfifo_depth* fixe la taille du tampon de sortie. Le paramètre *flit* fixe la taille du crédit de caractères représentant la quantité de base des transactions du contrôle de flux dans RCUBE (voir 3.3.1).

4.3.2 Modèle de processeur

Le modèle du processeur est un émetteur-récepteur qui génère et consomme des messages. Il accepte deux paramètres principaux: la taille du message généré et la fréquence de génération ou charge (fixant le volume du trafic). Le modèle du processeur implémente un trafic uniforme aléatoire Les adresses sont générées de manière équiprobable, aléatoire et uniforme. La déclaration complète des attributs du modèle du processeur est donnée dans la figure 4.4. Elle contient les paramètres suivants utilisés de la génération de charge:

```
component generator
 begin
    interface
         bidirectional north link[2];
    end
    node configuration
       unsigned label,
                range;
    end
    component configuration
         int    flit=32;
         int   message_length=64;
         float generation_rate=0.6;  -- Load
    end
    probes
      shared long    max_delay;
      shared float   mean_delay;
      shared long    generated_messages;
      shared long    received_messages;
      shared float   throughput;
      shared float    bandwidth;
      shared long    side_links;
   end
 end
```

Figure 4.4: Fichier de définition des ressources du processeur

- *label* c'est l'adresse identifiant chaque processeur.
- *range* Contient le nombre maximal d'adresses existantes. Ce paramètre définit l'intervalle de génération aléatoire des adresses. Par exemple, dans un réseau de N processeurs, *range* est égal à N.

Les autres paramètres déterminent les paramètres de génération de charge:

- *flit* a la même définition que pour le routeur RCUBE.
- *message_length* fixe la taille des messages en caractères.
- *generation_rate* ou taux de génération qui est une fraction de la charge maximale possible. La charge maximale corresponds à l'émission d'un caractère à chaque cycle.

Le routeur RCUBE implémente un mécanisme de contrôle de flux de bas niveau garantissant une transmission sans pertes, même en cas de congestion du réseau: pour chaque lien physique entre deux routeurs, le noeud récepteur peut bloquer l'émission des caractères par le noeud émetteur qui est à l'autre extrémité du lien. Ce mécanisme de contrôle de flux peut se rétropropager à travers le réseau jusqu'au processeur qui est la source d'un message, si toutes les FIFO distribuées dans les routeurs se remplissent. Le contrôle de flux doit donc être pris en compte par les processeurs: la charge définie par le paramètre *generation_rate* est injectée dans une FIFO de profondeur "illimitée" qui injecte elle-même les messages sur le réseau dés que celui-ci le permet.

Le générateur inclut également des sondes pour la récupération de données. Les plus importantes sont: la latence moyenne (*mean_delay*) et la bande passante effective. La bande passante effective (*throughput*) est la bande passante en sortie du réseau. C'est à dire la bande passante correspondant à tous les messages qui ont totalement traversés le réseau. Elle est également exprimée en fraction de la bande passante maximale. Avant la saturation du réseau, elle est sensiblement égale à la charge injectée dans le réseau. Le **point saturation** se définit donc comme la valeur de la charge en entrée du réseau à partir de laquelle la bande passante effective n'est plus égale à la charge. Le reste des paramètres corresponds à:

- *max_delay*: latence maximale.
- *generated_messages*: nombre de messages générés lors d'une simulation.
- *received_messages*: nombre de messages reçus lors d'une simulation.
- *bandwidth*: bande passante à l'entrée du réseau. Il corresponds aux messages effectivement injectés dans le réseau, alors que la charge correspond aux messages réellement générés et mis dans la FIFO de sortie du processeur. Avant saturation, ce paramètre corresponds sensiblement à la charge. A partir de la saturation, le réseau entre dans une situation de congestion.

4.4 Étiquetage de différentes topologies de réseau

Plusieurs topologies peuvent être retenues pour constituer le réseau d'interconnexion d'une machine MPC. Il existe, dans la littérature, une multitude de topologies. Il est utile de distinguer entre les topologies directes et les topologies indirectes. Dans un réseau direct, il y a un processeur attaché à chaque routeur alors que dans un réseau indirect il y a quelques routeurs auxquels n'est attaché aucun processeur.

4.4.1 Switch simple

Le réseau switch simple ou crossbar 8×8 est réalisé avec un seul RCUBE auquel sont connectés 8 processeurs. Les processeurs sont numérotés de 0 à 7. L'étiquetage de RCUBE est simple: Toutes les tables de routage sont identiques et tous les intervalles ont une seule adresse. Les intervalles vont de 0 à 7. A chaque intervalle est associé un seul port de sortie: le port de livraison du processeur correspondant à l'adresse représentée par l'un des intervalles.

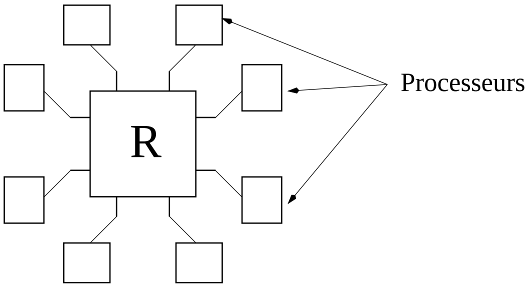


Figure 4.5: Crossbar

4.4.2 Réseau complètement connecté

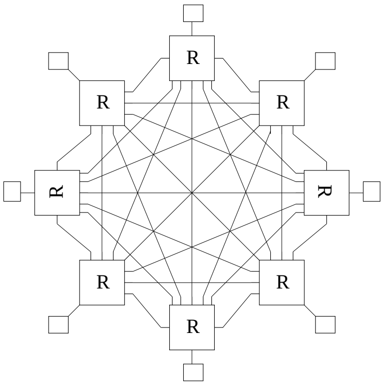


Figure 4.6: Réseau de 8 RCUBEs complètement connectés

Le réseau complètement connecté réalisé avec 8 RCUBEs se construit en connectant chacun des 8 routeurs aux 7 autres à travers un seul lien. Chaque RCUBE comporte un processeur. Les 7 ports restant au niveau de chaque RCUBE sont utilisés pour le connecter avec les autres RCUBEs. L'étiquetage des RCUBEs est déterministe et identique à celui du réseau précédent. La seule différence est que le routage s'effectue en deux étapes au lieu d'une. Le paquet partant de l'émetteur traverse d'abord le routeur associé. Il est ensuite acheminé vers le routeur associé au récepteur. Il est enfin livré à ce dernier.

4.4.3 Grille 2D

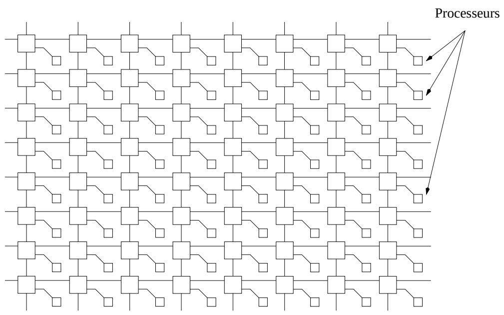


Figure 4.7: Grille 2D

Considérons une grille à deux dimensions de n fois n éléments. Nous supposons en premier un étiquetage classique des noeuds utilisant un couple (x, y) tel que $x, y \in [0, n-1]$. Pour une plus grande clarté, nous considérons que le noeud $(0, 0)$ est situé au coin inférieur gauche de la grille.

Routage déterministe

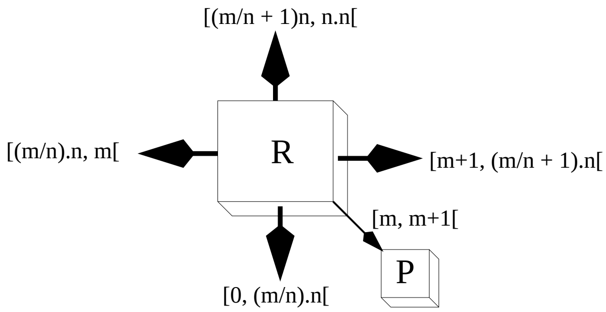


Figure 4.8: Étiquetage déterministe de la grille 2D

L'étiquetage par intervalles pour un routage déterministe est présenté à la figure 4.8. Il est défini pour chaque noeud (x, y) tel que $m = (y \times n) + x$ constitue l'adresse du processeur local. Il est évident que si un port de sortie n'est pas connecté (c'est le cas des noeuds situés sur les bords) alors les intervalles correspondants sont vides. Considérons un message en train d'être routé dans le réseau et supposons que (x_c, y_c) est l'adresse du noeud courant, Comme on peut le voir ci-dessous, cet étiquetage déterministe définit une stratégie de routage où il existe un seul changement de direction pour n'importe quelle paire de noeuds.

- $y_d < y_c$ (route vers le nord)
- $y_d > y_c$ (route vers le sud)
- $y_d = y_c$ and $x_d = x_c$ (route vers le processeur local)
- $y_d = y_c$ and $x_d < x_c$ (route vers l'est)
- $y_d = y_c$ and $x_d > x_c$ (route vers l'ouest)

Routage adaptatif

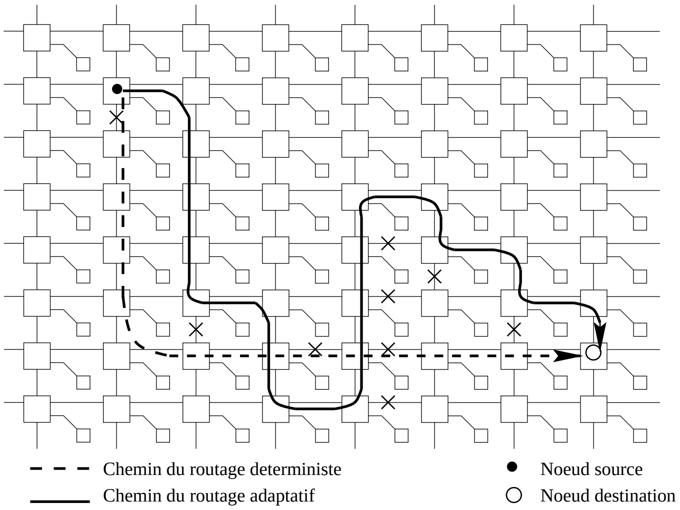


Figure 4.9: Principe du *turn model*

Pour exploiter les capacités de routage adaptatif de RCUBE, on s'inspire du *turn model* de [15]. Cette approche consiste essentiellement à autoriser certains changements de direction supplémentaires de façon à contourner un lien indisponible ou en panne (voir figure 4.9). La difficulté consiste à éviter les phénomènes d'interblocages en interdisant certains types de changement de direction. Trois algorithmes basés sur le *turn model* ont été proposés par [15]. Le tableau de la figure 4.10 présente ces trois algorithmes. Le port de sortie privilégié est celui donné par l'algorithme déterministe. Le port de sortie possible est choisi parmi un ensemble de ports de sortie suivant une certaine priorité. La plus haute est 1 et la plus basse est 4. Les cases vides montrent que le port de sortie correspondant n'est pas une sortie possible pour le port d'entrée.

Port d'entree	Port de Sortie Preferentiel	Port de Sortie														
		Algorithm A1					Algorithm A2					Algorithm A3				
		E	O	N	S	Chgt. de dir.	E	O	N	S	Chgt. de dir.	E	O	N	S	Chgt. de dir.
Injection	Est	1		2			1					1		2		
	Ouest		1	2				1	3	2			1		2	
	Nord			1				3	1	2			2	1		
	Sud	3	4	2	1			2		1			2		1	
Est	Est															
	Ouest		1					1	3	2			1#*			
	Nord							3	1	2			2#	1#		
	Sud		2		1			2		1			2#		1#	
Ouest	Est	1					1					1				
	Ouest															
	Nord											2		1		
	Sud	2			1											
Nord	Est	1					1					1				
	Ouest		1					1	2				1		2	
	Nord							1		2			2		1	
	Sud	2	3		1			2		1			2		1	
Sud	Est	1		2			1					1				
	Ouest		1	2				1					1*			
	Nord			1				2	1			2		1		
	Sud	2	3		1			1					1			
Resume																

#: Non present si venant du second port (Est2 ou Ouest2)
*: Correspond au second port (Est2 ou Ouest2)
#*: Seulementle second port de sortie est present si venant du second port d'entree..

Figure 4.10: Algorithmes adaptatifs

4.4.4 Grille composée

La grille à 2 dimensions a un diamètre important par construction. Pour réduire ce diamètre, nous proposons une structure construite par composition de la grille et d'un réseau complètement connecté à 4 éléments (Figure 4.11). D'après Zerrouk [16], le routage dans un réseau composé se fait comme suit:

- Si le destinataire est connecté à la source par le réseau complètement connecté local, le routage est direct.
- Si le destinataire est distant, le routage se décompose en 2 étapes:
  - Emprunter la grille à laquelle appartient le noeud source. Effectuer un routage dans la grille jusqu'a atteindre le groupe auquel appartient le destinataire.
  - Effectuer un routage direct dans le réseau complètement connecté local.

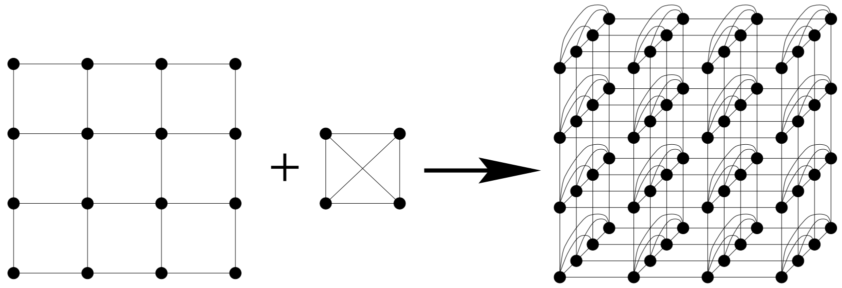


Figure 4.11: Grille composée

Pour le routage dans la grille, on peut implémenter n'importe quel algorithme développé pour une grille simple. La seule transformation à effectuer est le pas de progression lors du calcul des intervalles qui ne sera plus d'une unité mais de la taille d'un groupe qui est 4.

La grille composée constitue un bon compromis puisqu'elle possède la régularité de la grille 2D tout en exploitant au mieux la valence 7 des noeuds de la machine MPC.

4.4.5 Les réseaux multi-étages

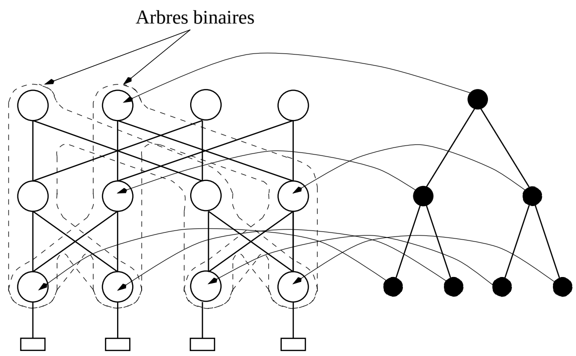


Figure 4.12: Les réseaux LCAN: une superposition d'arbres

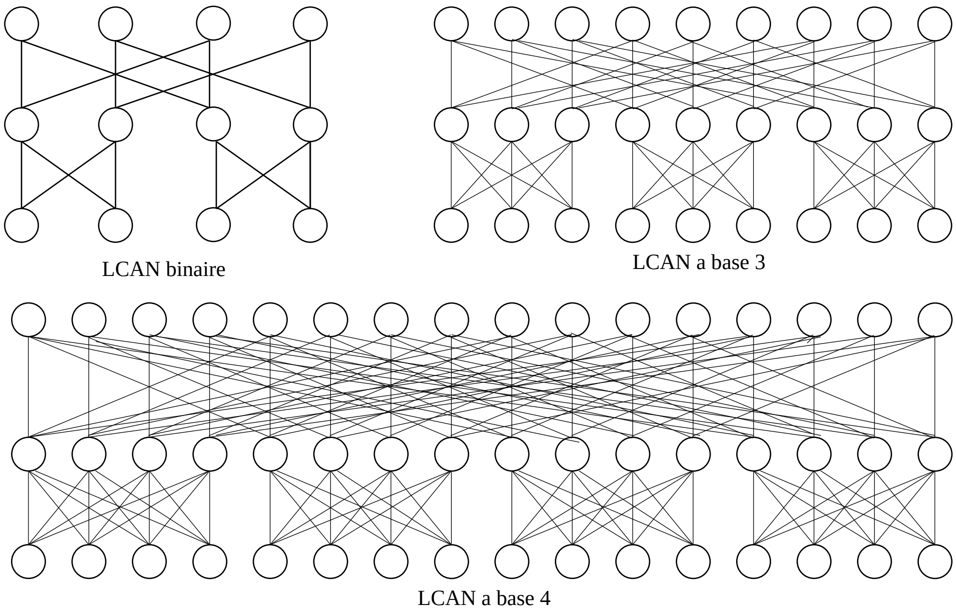


Figure 4.13: Plusieurs variantes de LCANs

Les réseaux multi-étages MIN (*Multistage Interconnection Networks*) sont des réseaux où l'interconnexion entre les processeurs est réalisée à travers plusieurs étages de routeurs. Ici, nous considérons les réseaux pliés également appelés LCAN (*Least Common Ancestor Networks*) [17] [18] [19] [20] La différence entre un MIN et un LCAN est que les liens de routage sont unidirectionnels dans le premier et bidirectionnels dans le second. Nous notons que les structures MIN les plus traditionnelles peuvent être pliées pour obtenir des structures LCAN.

La construction et l'étiquetage d'un LCAN est directement dérivée d'un arbre (voir figure 4.12). Dans l'exemple présenté dans la figure 4.12, l'arbre sur lequel se base la construction est un arbre binaire. Cependant, il est possible de construire d'autres LCAN en se basant sur des arbres d'arité plus élevée (voir figure 4.13).

Comme exemple, nous considérons la structure MIN en papillon (*butterfly*) bien connue dans la littérature [21] [22] [23] [24] [25] qui est basée sur un arbre binaire. Sa version LCAN est présentée en figure 4.14. Plusieurs structures peuvent être dérivées du LCAN butterfly juste en effectuant des permutations sur la position des liaisons connectant les routeurs à des étages successifs. Il est montré que ces permutations n'ont aucun effet sur l'étiquetage qui reste inchangé. Considérant que le nombre de routeurs à chaque étage est 2^n, le nombre d'étages est de $n+1$ et le nombre de processeurs est de $k.2^n$, où k est le nombre de ports situés en bas du premier étage. Pour le cas de RCUBE $k = 4$ au maximum.

L'étiquetage d'un LCAN butterfly d'ordre n tel que le nombre de processeurs p est égal à $k.(2^n - 1)$ se fait comme suit (% désigne le modulo et la division est entière) :

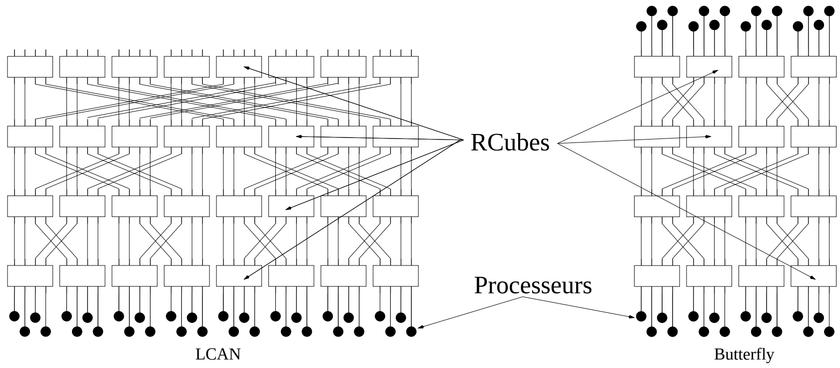


Figure 4.14: Butterfly et LCAN dérivé réalisés avec des RCUBEs

- Pour chaque routeur (j, i) de la j^{eme} ligne et de la i^{eme} colonne:
  - Grouper les ports du haut et les étiqueter avec $[(2^j.k.((i/2^j)+1))\%p, (2^j.k(i/2^j))\%p[$
  - Grouper les ports du bas gauches et les étiqueter avec $[(2^j.k.(i/2^j))\%p, (2^j.k(2(i/2^j)+1))\%p[$
  - Grouper les ports du bas droits et les étiqueter avec $[(2^j.k.(2(i/2^j)+1))\%p, (2^j.k((i/2^j)+1))\%p[$
- Étiqueter les ports terminaux (routeurs du premier étage vers les processeurs correspondants) de gauche à droite et de 0 à $k.2^n - 1$

Routage déterministe

Les réseaux LCAN, comme les réseaux MIN en général, sont adaptatifs par construction puisqu'ils fournissent plusieurs chemins pour aller d'un point à un autre du fait que ces réseaux ne sont qu'une superposition d'arbres.

Sur ces réseaux, l'algorithme déterministe est dérivé de l'algorithme adaptatif en divisant les intervalles correspondants aux ports groupés par le nombre de ces ports. De cette manière, a chaque port faisant partie d'un groupe sera assigné un intervalle distinct. L'union de ces intervalles donne l'intervalle initial auquel étaient associés tous les ports du groupe. Pour RCUBE et le réseau présenté en figure 4.14, l'intervalle des ports du haut sera divisé en 4 intervalles de taille égale. Les intervalles des ports de gauche et de droite seront chacun divisés en 2 intervalles de taille égale.

Variantes à valence 7

D'après la figure 4.14, le LCAN n'est pas conforme aux spécifications de la machine MPC car on a 4 processeurs par routeur sur la rangée du bas. Une manière de le rendre

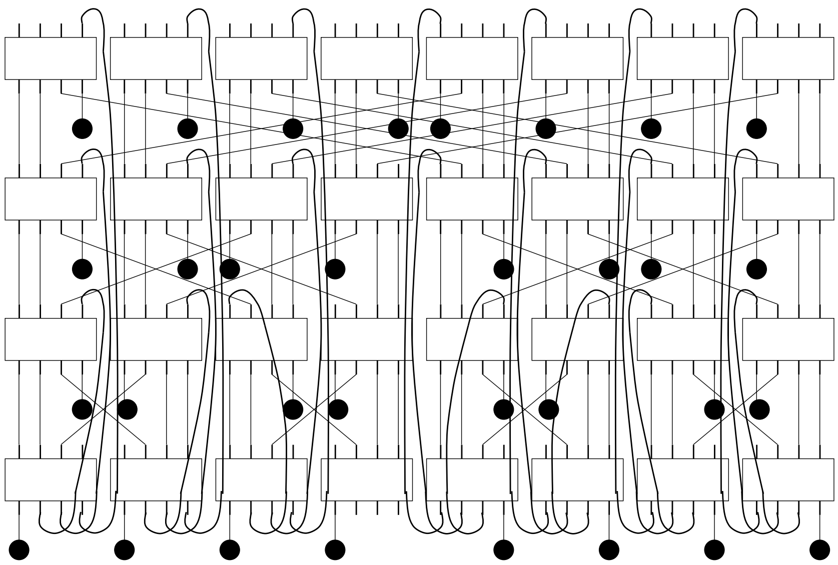


Figure 4.15: Variante du LCAN Butterfly à valence 7

conforme est de ne connecter qu'un seul processeur par routeur à la base. Le réseau obtenu requiert un grand nombre de routeurs par processeur. Par exemple, pour 32 processeurs, le nombre de routeurs est $2^{5+1} \times 32 = 192$, ce qui est très coûteux. Il est donc intéressant d'essayer de réduire ce nombre en gardant la structure de base du LCAN.

Un LCAN basé sur un arbre binaire requiert au minimum 4 ports par routeur (2 en haut et 2 en bas). Considérons, un couple processeur/routeur qui constitue la brique de base de la machine MPC. Nous pouvons construire un LCAN en utilisant 4 ports des 7 ports disponibles. Reste alors 3 ports inoccupés. 2 de ces 3 ports sont utilisés pour doubler la bande passante des liens verticaux (vers le haut et vers le bas). Le port restant est connecté au routeur de la base dans la même colonne. Cette structure permet de garder le schéma de routage inchangé. C'est ce port qui met directement en contact le processeur local à un routeur au routeur de la base qui permet de simuler la présence de 4 processeurs par routeur sur la rangée du bas et de garder le routage inchangé. Ce réseau est présenté en figure 4.15. Le nombre de processeurs par colonne est égal au nombre d'étages puisque chaque routeur comporte un processeur. Ainsi, puisqu'au niveau du routeur de la base, on ne peut connecter au maximum que 4 processeurs, l'extension du réseau au delà de 32 processeurs pose problème. Nous avons choisi d'étendre le réseau avec des étages de routeurs dépourvus de processeurs.

4.4.6 l'hypercube

Un hypercube [26] [27] [28] de dimension D est constitué de 2^D noeuds. Chacun possède D liens correspondant aux dimensions 0 à $D-1$.

Pour tout noeud v atteint à partir d'un noeud u, à travers le lien correspondant à la

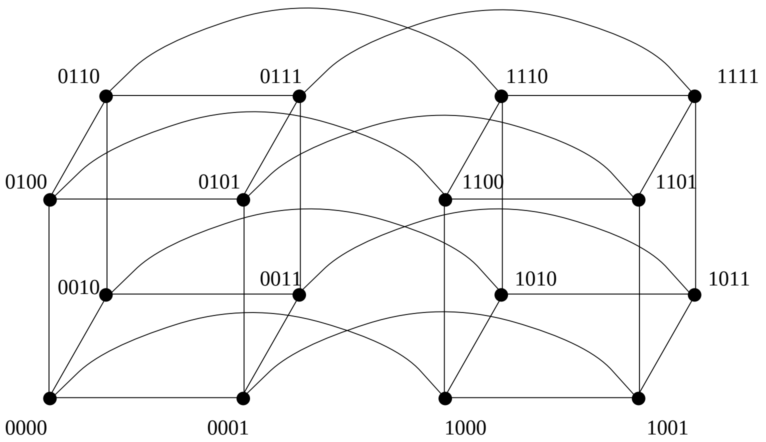


Figure 4.16: Hypercube à 4 dimensions

dimension d, faire:

a Étiqueter v avec l'adresse de u dont seulement le d^{ieme} bit est mis à 1.

b Atteindre tous les noeuds adjacents à v dont les liens qui les lient à lui correspondent à des dimensions supérieures à d.

Il suffit ensuite de choisir un noeud au hasard, qu'on étiquettera avec l'adresse 0.

Par construction, les adresses de toute paire de noeuds adjacents diffèrent de seulement un bit. Une stratégie de routage est alors simple à définir. Pour aller d'un noeud x vers un noeud y, il suffit de:

- Répéter ce qui suit jusqu'a ce que l'adresse du noeud courant c (au début $c = x$) soit identique avec celle du noeud destination y.
- Déterminer les différences de bits entre l'adresse de c et celle de y.
- Considérer le rang des bits différents. Les classer par ordre décroissant. Nous nommons par r_p le rang du premier bit différent.
- Emprunter le lien correspondant à la dimension de r_p.

Dans un réseau de RCUBEs, on associe un processeur à chaque routeur,

le nombre maximal de noeuds est donc de 2^{8-1}. L'étiquetage d'un hypercube de degré D se fait de la manière suivante:

Soit $a_c = b_{D-1}b_{D-2}....b_0$ l'adresse courante d'un processeur. Le routeur associé comprend D ports de sortie en plus du port local. On associe un intervalle à chaque dimension. La borne inférieure de l'intervalle associé à une dimension d est calculée de la manière suivante:

- Si $b_d = 0$ alors atteindre à travers le lien associé à d tout le groupe de 2^d éléments codé comme suit: $b_{D-1}...b_{d+1}10...0$

- Si $b_d = 1$ alors atteindre à travers le lien associé à d tout le groupe de 2^d éléments codé comme suit: $b_{D-1}...b_{d+1}00...0$

La mise à 0 des bits de poids faible est nécessaire pour atteindre tous les éléments adjacents du groupe (en commençant par celui qui a la plus petite adresse). Une fois toutes les bornes calculées, elles sont triées suivant un ordre croissant. La liste d'intervalles comprends également a_c. A chacune de ces bornes on associe le port de sortie correspondant à la dimension.

Routage adaptatif

Pour des réseau de taille inférieure à 128, les ports disponibles ne sont pas tous utilisés. Par exemple, pour un hypercube de 32 éléments, seuls 6 ports sont utilisés par RCUBE, le reste est non connecté. Pour une utilisation plus efficace de cette ressource, nous avons choisi de doubler la bande passante sur certaines dimensions ce qui produit un schéma de routage adaptatif.

4.4.7 le réseau *banyan-hypercube*

Le réseau dit *banyan-hypercube* [29] [30] [31] est une combinaison entre un réseau multi-étages et un hypercube. Dans un réseau multi-étages basé sur l'arbre binaire, chaque routeur utilise au minimum 4 ports de communication (2 en haut et 2 en bas). Dans une telle topologie, le reste des ports est disponible pour réaliser une interconnexion supplémentaire quelconque. Une possibilité est la connexion des routeurs appartenant au même étage par une structure d'hypercube. Dans cette topologie, chaque routeur comporte un processeur. Un paquet est routé à travers le réseau multi-étages tant qu'il

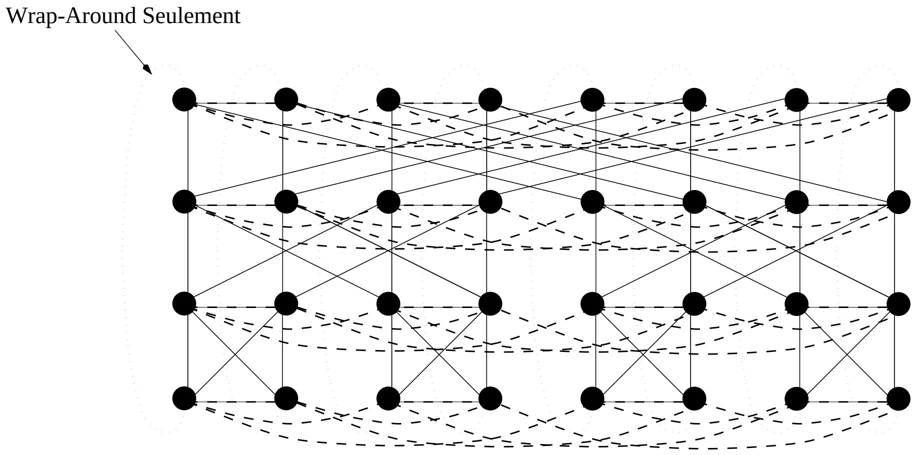


Figure 4.17: Réseaux *banyan-hypercube* et *wrap-around banyan-hypercube*

n'a pas atteint un routeur faisant partie du même étage que sa destination. Quand celui-ci a atteint l'étage correspondant, il est routé à travers l'hypercube local.

Le nombre de ports utilisé pour réaliser le réseau multi-étages est égal à 4. Ajoutant le port local connectant le processeur au routeur, il ne reste plus que 3 ports permettant de construire un hypercube local de, au plus, $2^3 = 8$ noeuds. Cette taille est la taille maximale que peut avoir un étage. Ainsi, la réalisation de ce réseau avec des circuits RCUBE ne peut s'effectuer au delà de 32 noeuds.

A chaque noeud on associe un couple (i, j). i corresponds à l'abscisse du noeud et j son ordonnée dans la grille composée par tous les noeuds quand les liens ne sont pas considérés. L'étiquetage de RCUBE se fait comme suit:

- Étiqueter le processeur avec l'intervalle: $[i + j.E, i + 1 + j.E[$ (E corresponds à la taille d'un étage).
- Étiqueter les ports du haut avec: $[(j + 1).E$, 0[.
- Étiqueter les ports du bas avec $[0, j.E[$.
- Étiqueter les autres ports en appliquant les règles de l'hypercube.

L'étage du haut et l'étage du bas comportent un groupe de ports inutilisés. Ces ports peuvent être connectés sur pour former un anneau au niveau de chaque colonne. Le réseau ainsi obtenu s'appelle *wrap-around banyan-hypercube* ou WABH.

4.4.8 Évaluation des performances et détermination des meilleurs réseaux

Nous avons effectué les simulations sur différents réseaux pour évaluer leur performances. Nous avons fait varier deux paramètres: la charge appliquée et la taille des messages générés. Nous avons fait varier la charge de 10 à 90 % de la bande passante maximale avec un pas de 10 choisi trois tailles de messages pour effectuer les simulations: 64, 128 et 500 caractères. Pour chaque taille, nous avons fait varier la charge du minimum au maximum. Ainsi, le total des simulations pour chaque réseau est de 27 simulations. Celà correspond aux 9 valeurs de la charge multipliées par les 3 tailles de messages.

Plusieurs mesures ont été effectuées à chaque simulation. Nous avons mesuré la latence moyenne, la bande passante effective, la bande passante en entrée, le nombre de messages générés et le nombre de messages reçus. Nous avons choisi de ne considérer que la latence moyenne et la bande passante effective car ce sont les paramètres les plus significatifs. Un autre paramètre significatif est la latence maximale. Cependant, ce paramètre n'est pas régulier et sa valeur peut changer considérablement d'une simulation à une autre. Nous ne l'avons donc pas considéré.

Nous avons simulé plusieurs réseaux de différentes tailles. Les réseaux que nous allons présenter ont donné des résultats comparables. Ces réseaux sont (les notations choisies pour désigner chaque réseau seront utilisées dans tout le reste du chapitre):

- BH: *Banyan-Hypercube*.
- GC: Grille composée.
- GrilleD: Grille 2D à algorithme déterministe.
- GrilleA: Grille 2D à algorithme adaptatif.
- Hyp: Hypercube déterministe.
- HypA: Hypercube adaptatif.
- LCAN2P: LCAN binaire à un seul processeur par routeur à l'étage de base (pour le conformer aux spécifications de MPC).
- LCAN2T: LCAN binaire transformé pour être de valence 7 (un processeur par routeur connectés colonne par colonne au routeur de la base).
- RCC: Réseau complètement connecté.
- Switch: Crossbar réalisé avec un seul RCUBE.
- WABH: *Wrap-Around Banyan-Hypercube*.

Les résultats sont présentés sous forme de tableaux donnant la latence moyenne en fonction de la charge appliquée. Dans ces tableaux figure également la valeur du point de saturation et les paramètres liés au coût (nombre de processeurs, etc...). Les réseaux sont regroupés par taille. A chaque taille de réseau correspond un tableau présentant les résultats de tous les réseaux de cette taille pour les 3 tailles de messages (64, 128 et 500 caractères). Nous avons choisi 5 tailles de réseaux: 8, 16, 32, 64 et 128 processeurs. Cette manière de présenter a pour buts:

- Déterminer le réseau qui possède la meilleure latence.
- Déterminer le réseau qui possède la meilleure bande passante.
- Permettre au concepteur de déterminer le meilleur réseau pour une taille de message donnée et une charge donnée.
- Introduire le facteur coût directement dans le processus de choix.

Dans tous les tableaux présentés ci-après, les tailles de messages sont exprimés en caractères, la latence moyenne en cycles, la bande passante effective en fraction de la bande passante maximale et les paramètres du coût en unités.

Évaluation du coût et de l'extensibilité

Réseau	Nombre de processeurs	Nombre de routeurs	Connectivité	Diamètre	Incrément minimal	Taille maximale
GC	n	n	8	$\sqrt{n}+1$	$2\sqrt{n}$	∞
Grille	n	n	5	$2\sqrt{n}-1$	$\sqrt{n}$	∞
LCAN2P	n	$n(logn+1)$	8	$2logn+1$	n	∞
LCAN2T	n	$\frac{n}{4}(log(\frac{n}{4})+1)$	8	$2log(\frac{n}{4})+1$	n	∞
Hyp	n	n	8	$logn$	n	128
BH	n	n	8	$2logk$	k	32
WABH	n	n	8	$\frac{3}{2}logk$	k	32
$n = k(logk+1)$ pour les réseaux BH et WABH						

Table 4.1: Coût et extensibilité

Le tableau 4.1 présente les paramètres du coût et de l'extensibilité pour chaque réseau. Ce tableau a pour but de donner les formules de calcul du coût d'un réseau donné à une taille donnée. Il fournit également des informations sur l'extensibilité (si le réseau est extensible) tels que l'incrément minimal nécessaire pour étendre le réseau ou la taille maximale réalisable avec des RCUBEs.

Le nombre de processeurs est n. Pour les réseaux LCAN le nombre de processeurs par routeur au premier étage est p. Un autre paramètre, k, est introduit pour exprimer plus facilement l'incrément minimal des réseaux BH et WABH. Tous les paramètres sont exprimés en unités de processeurs ou de routeurs. L'incrément minimal et la taille maximale sont exprimés en nombre de processeurs.

4.4.9 Réseaux à 8 processeurs

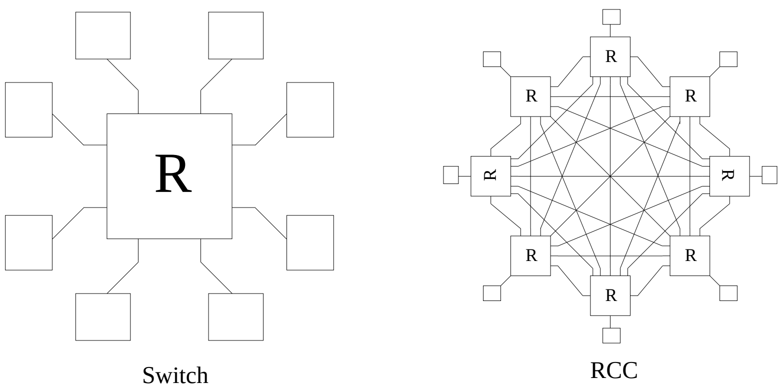


Figure 4.18: Les deux réseaux crossbars réalisés avec RCUBE

D'après le tableau 4.2, le réseau Switch présente la latence la plus faible pour des valeurs faibles de la charge. Ce résultat s'explique par le fait qu'un simple RCUBE a une latence minimale par rapport à plusieurs dans un contexte ou les conflits sur les tampons sont minimes à cause de la faible charge appliquée.

Si nous considérons que le RCC est un crossbar, la structure de mémorisation est différente par rapport à celle du switch contenant un seul routeur. Le réseau RCC permet de router un deuxième message bloqué derrière le premier message dans le réseau Switch.

La performance du réseau LCAN2P par rapport au RCC à partir d'une certaine charge et jusqu'a la saturation est due à la supériorité en bande passante de celui-ci et à l'adaptativité du routage.

Dans la phase intermédiaire, le réseau RCC obtient une latence moyenne inférieure à celle du réseau LCAN2P. La cause est qu'en absence

		Coût		Point	Latence en fonction de la charge					
TM	Réseaux	NP	NR	Sat.	0.1	0.2	0.3	0.4	0.5	0.6
64	Switch	8	1	0.55	**77.78**	**83.79**	**94.30**	115.56	198.79	sat
	RCC	8	8	0.73	87.20	91.11	96.96	**105.65**	**115.15**	**143.73**
	MIN2P	8	32	0.78	121.06	115.84	129.84	141.16	150.21	183.67
128	Switch	8	1	0.55	**145.48**	**155.14**	**176.72**	220.14	371.75	sat
	RCC	8	8	0.62	154.13	162.77	180.72	**208.44**	**250.72**	930.27
	MIN2P	8	32	0.73	187.59	196.85	206.97	224.36	256.90	**299.57**
500	Switch	8	1	0.56	**532.32**	606.34	627.58	729.04	1203.79	sat
	RCC	8	8	0.60	547.37	574.95	**598.38**	**725.31**	989.90	1971.97
	MIN2P	8	32	0.64	572.95	631.37	645.40	755.40	**905.82**	**1251.66**

Table 4.2: Latence en fonction de la charge pour des réseaux à 8 processeurs

(relative) de conflits qui tendent à ralentir les messages la distance moyenne du réseau RCC est plus courte que celle du réseau LCAN2P. Ce réseau est celui qui dispose de la plus grande bande passante effective. Ce résultat est atteint à cause du grand nombre de routeurs qu'il met en oeuvre. Cependant ce faible gain en bande passante ne justifie pas l'utilisation de 32 routeurs.

Le réseau qui offre de très bonnes performances couplées à un coût avantageux est le réseau RCC. De plus ce réseau est à valence 7 donc directement réalisable sans transformation aucune. C'est ce dernier que nous recommandons pour cette taille de réseau.

Réseaux à 16 processeurs

Les résultats des simulations sont présentés dans la Table 4.3 Les réseaux intéressants sont le GC et l'hypercube aussi bien adaptatif (HypA) que déterministe (Hyp) qui

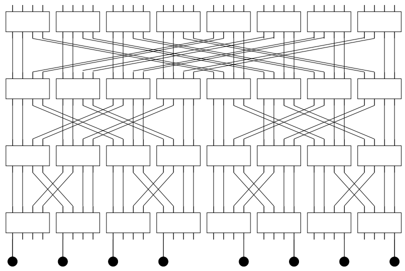


Figure 4.19: Le réseau LCAN2P

présentent des résultats comparables. Pour des tailles de messages supérieures à 64 et pour de fortes charges le réseau LCAN2P obtient la plus faible latence.

Nous remarquons qu'il n'y a aucun réseau qui se distingue nettement des autres. Nous ne pouvons de ce fait recommander un réseau déterminé car la considération de plusieurs critères de choix aboutit à des résultats différents.

Si l'extensibilité et la latence sont considérées ensemble, le réseau indiqué est le GC car l'hypercube n'est pas extensible au delà de 128 processeurs. Ce dernier peut toutefois être choisi puisque l'extensibilité ne rentre finalement en ligne de compte qu'au delà de 128 processeurs. L'hypercube adaptatif (HypA) est donc celui que nous recommandons même si pour des faibles charges le réseau GC obtient des résultats légèrement meilleurs.

C'est le réseau LCAN2P qui obtient la plus grande bande passante. Cependant, le réseau HypA possède également une large bande passante.

TM	Réseaux	Coût		Point	Latence en fonction de la charge					
		NP	NR	Sat.	0.1	0.2	0.3	0.4	0.5	0.6
64	GC	16	16	0.66	96.03	101.54	109.83	119.86	141.50	214.47
	LCAN2P	16	80	0.78	137.51	145.27	147.13	161.63	174.66	197.98
	Hyp	16	16	0.63	99.26	106.27	113.77	126.75	153.98	322.30
	HypA	16	16	0.72	98.14	103.57	109.52	120.02	**130.72**	**164.56**
128	GC	16	16	0.58	163.49	175.66	193.10	226.54	303.76	sat
	LCAN2P	16	80	0.68	205.77	219.78	229.37	250.36	269.29	**344.46**
	Hyp	16	16	0.58	166.57	179.79	201.43	232.68	343.68	sat
	HypA	16	16	0.66	165.46	175.77	192.02	**210.33**	**252.80**	373.88
500	GC	16	16	0.55	570.40	**597.45**	682.68	832.94	1436.23	sat
	LCAN2P	16	80	0.57	606.32	642.23	679.82	**818.32**	**1017.79**	sat
	Hyp	16	16	0.53	564.56	621.56	729.52	862.58	1677.65	sat
	HypA	16	16	0.57	**561.71**	608.72	**671.20**	854.75	1146.28	sat

Table 4.3: Latence en fonction de la charge pour des réseaux à 16 processurs

Réseaux à 32 processeurs

Les résultats des simulations sont présentés dans la Table 4.4. D'après cette table, nous remarquons que les réseaux qui obtiennent les latences les plus faibles sont: Hyp, HypA, GC, BH, WABH et LCAN2P.

TM	Réseaux	Coût		Point	Latence en fonction de la charge					
		NP	NR	Sat.	0.1	0.2	0.3	0.4	0.5	0.6
64	GC	32	32	0.58	105.24	112.57	124.02	144.83	194.46	sat
	LCAN2P	32	192	0.76	158.53	162.07	172.78	180.64	194.63	**219.65**
	LCAN2T	32	32	0.41	119.09	133.19	164.65	744.64	sat	sat
	Hyp	32	32	0.62	104.65	112.40	122.45	137.02	168.25	370.59
	HypA	32	32	0.67	**104.29**	110.65	**119.35**	**131.41**	**154.10**	226.23
	BH	32	32	0.65	106.62	112.55	123.68	137.68	163.73	315.76
	WABH	32	32	0.65	104.49	**110.08**	119.50	132.72	162.34	256.39
128	GC	32	32	0.52	173.11	192.08	221.71	285.33	965.13	sat
	LCAN2P	32	192	0.61	223.91	237.58	254.12	278.71	**342.12**	**962.63**
	LCAN2T	32	32	0.38	190.68	217.77	293.68	sat	sat	sat
	Hyp	32	32	0.56	174.08	187.90	209.71	256.36	475.92	sat
	HypA	32	32	0.58	172.63	**185.03**	**206.01**	251.61	349.17	sat
	BH	32	32	0.57	175.62	188.92	207.84	248.22	371.38	sat
	WABH	32	32	0.57	173.19	186.47	207.28	**247.45**	379.92	sat
500	GC	32	32	0.47	580.35	644.09	802.68	1283.37	sat	sat
	LCAN2P	32	192	0.64	572.95	631.37	**645.40**	**755.40**	**905.82**	**1251.66**
	LCAN2T	32	32	0.37	595.99	764.33	1172.63	sat	sat	sat
	Hyp	32	32	0.47	581.57	637.41	840.29	1011.95	sat	sat
	HypA	32	32	0.53	581.93	623.20	707.68	955.42	2141.69	sat
	BH	32	32	0.53	572.51	632.72	738.74	944.42	1953.26	sat
	WABH	32	32	0.51	572.66	622.71	738.10	991.29	1815.84	sat

Table 4.4: Latence en fonction de la charge pour des réseaux à 32 processeurs

Le réseau GC obtient des performances intéressantes et se classe mieux que le LCAN2P pour des charges inférieures à 0.5 et des tailles de message inférieures à 500 caractères. Ces résultats sont souvent proches de ceux de l'hypercube.

Le réseau LCAN2P possède encore une fois la plus grande bande passante. De ce fait, il atteint son point de saturation plus tard que les autres réseaux et par conséquent obtient les meilleures latences pour les charges importantes.

Par contre, le réseau LCAN2T, qui est l'autre variante d'un LCAN, atteint très tôt son point de saturation.

Le réseau WABH, qui est une variante améliorée du réseau BH, a toujours de meilleures performances que le réseau BH. Ce dernier n'est par conséquent plus considéré.

Nous avons choisi de recommander le réseau HypA car c'est celui qui réalise les meilleurs compromis. C'est un réseau extensible jusqu'a 128 processeurs (hypercube). Le réseau WABH qui obtient de bonnes performances, se place souvent juste derrière le réseau HypA. Cependant, ce réseau (WABH) ne peut être choisi car on ne peut le réaliser que pour cette taille particulière. Le réseau GC, à cause de son extensibilité est également recommandable. Son seul inconvénient est qu'il se sature plus tôt que l'hypercube.

Réseaux à 64 processeurs

La table 4.5 présente les résultats de simulation pour les réseaux à 64 processeurs. Les réseaux Hyp et HypA se distinguent de tous les autres par la plus faible latence et la plus forte bande passante. Le réseau GC obtient également des résultats satisfaisants.

Par contre, les deux réseaux GrilleD et GrilleA se saturent très tôt et présentent des latences relativement élevées,

Nous recommandons donc l'utilisation des réseaux Hyp ou HypA. Pour une taille de 64 processeurs, le réseau le plus avantageux est le HypA aussi bien pour la latence que pour la bande passante.

		Coût		Point	Latence en fonction de la charge				
TM	Réseaux	NP	NR	Sat.	0.1	0.2	0.3	0.4	0.5
64	GrilleD	64	64	0.33	141.98	161.70	235.32	sat	sat
	GrilleA	64	64	0.34	136.13	160.01	219.22	sat	sat
	GC	64	64	0.51	114.58	125.49	142.09	179.15	602.82
	LCAN2T	64	80	0.33	133.07	152.49	263.23	sat	sat
	Hyp	64	64	0.61	**110.06**	**117.36**	128.45	146.09	188.00
	HypA	64	64	0.62	110.19	117.39	**128.11**	**143.87**	**179.26**
128	GrilleD	64	64	0.32	218.59	268.90	550.22	sat	sat
	GrilleA	64	64	0.34	208.22	245.01	465.10	sat	sat
	GC	64	64	0.44	185.86	210.35	258.15	533.33	sat
	LCAN2T	64	80	0.30	206.03	243.44	877.38	sat	sat
	Hyp	64	64	0.53	**179.62**	196.59	**220.32**	279.59	555.42
	HypA	64	64	0.54	179.85	**194.61**	223.05	**278.18**	**511.42**
500	GrilleD	64	64	0.26	648.76	859.04	sat	sat	sat
	GrilleA	64	64	0.26	616.00	779.45	sat	sat	sat
	GC	64	64	0.38	599.27	719.39	1024.35	sat	sat
	LCAN2T	64	80	0.31	629.62	879.87	2524.07	sat	sat
	Hyp	64	64	0.46	593.66	665.05	793.14	1272.83	sat
	HypA	64	64	0.46	**587.29**	649.38	802.30	1111.48	sat

Table 4.5: Latence en fonction de la charge pour des réseaux à 64 processurs

Réseaux à 128 processeurs

TM	Réseaux	Coût		Point	Latence en fonction de la charge				
		NP	NR	Sat.	0.1	0.2	0.3	0.4	0.5
64	GC	128	128	0.32	134.92	162.85	294.59	sat	sat
	LCAN2T	128	192	0.27	150.65	181.06	sat	sat	sat
	Hyp	128	128	0.59	**115.68**	**124.40**	**136.49**	**156.76**	**209.29**
128	GC	128	128	0.28	215.37	289.04	sat	sat	sat
	LCAN2T	128	1.5	0.26	227.47	297.98	sat	sat	sat
	Hyp	128	128	0.52	**185.64**	**204.26**	**234.95**	**296.97**	991.97
500	GC	128	128	0.26	671.28	1209.35	sat	sat	sat
	LCAN2T	128	192	0.24	675.49	1026.95	sat	sat	sat
	Hyp	128	128	0.45	**594.51**	**667.91**	848.13	**1416.33**	sat

Table 4.6: Latence en fonction de la charge pour des réseaux à 128 processurs

Pour des réseaux de 128 processeurs, l'hypercube (Hyp) obtient incontestablement les meilleurs résultats. Il se démarque nettement des autres réseaux. Les réseaux GC et LCAN2T se saturent très vite et n'atteignent pas des niveaux de performances comparables à l'hypercube.

Ainsi tous les critères de choix (meilleures latence et bande passante et meilleur coût) convergent vers l'hypercube. Nous recommandons donc son utilisation pour cette taille de réseau.

Simulation d'une grille composée à 256 processeurs

TM	Réseaux	Coût		Point	Lat. en f. de la charge	
		NP	NR	Sat.	0.1	0.2
64	GC	128	128	0.28	154.26	202.17
128	GC	"	"	0.25	239.51	390.88
500	GC	"	"	0.21	735.24	2265.51

Table 4.7: Latence en fonction de la charge d'une grille composée à 256 processurs

Nous avons simulé un réseau GC à 256 processeurs. Les résultats sont présentés dans le tableau 4.7. Ce réseau se sature très tôt (pour une charge autour de 0.25). En dépit de ces résultats, ces simulations prouvent qu'un tel réseau à grand nombre de processeurs est utilisable et arrive à acheminer des messages avec des délais appréciables pour des charges acceptables.

En Résumé

Au vu des résultats des simulations, il apparaît que la topologie dominante est incontestablement l'hypercube (plus souvent dans sa version adaptative).

Pour les réseaux à 8 processeurs, c'est le réseau RCC qui apparaît comme le plus avantageux. Pour toutes les autres tailles, l'hypercube a une position dominante.

Le réseau LCAN2P obtient d'excellents résultats. Malgré celà, il n'est pas considéré à cause du coût qu'il induit. La plus grande taille simulée de ce réseau (32 processeurs) requiert déjà 192 routeurs. Sa version transformée pour être conforme à la machine MPC (valence 7) n'atteint pas les niveaux de performance escomptés.

Cependant, tous ces résultats, obtenus avec un modèle de génération uniforme de messages, ne donnent qu'une indication générale sur le niveau de performances. Des résultats différents peuvent être obtenus pour des modèles de génération reproduisant le comportement d'applications spécifiques. Un réseau, comme la grille composée GC par exemple, pourrait être parfaitement adapté à une application possédant des communications locales.

4.5 Quantification du gain de l'adaptativité

La fragmentation des messages en paquets est une caractéristique de l'interface PCI. Dans un réseau déterministe (non-adaptatif), puisque le chemin entre deux noeuds est unique, la précédence entre les paquets est garantie. Dans un réseau adaptatif elle ne l'est plus. Le protocole logiciel qui gère les échanges change considérablement suivant le type de réseau dans lequel on se trouve. Dans le cas de réseaux adaptatifs, celui-ci devient plus complexe. Il est donc utile de quantifier le gain en performances qu'apporte l'adaptativité pour fournir un élément d'appréciation précis au concepteur.

Réseaux	**Charge**						Moyenne	Saturation		Gain
	0.1	0.2	0.3	0.4	0.5	0.6		Det.	Ad.	(%)
LCAN2 (32)	3.39	5.56	7.48	16.79	24.55	sat	**15.12**	0.57	0.58	0.01
LCAN2 (64)	3.32	6.77	11.34	17.51	44.58	sat	**22.92**	0.53	0.55	0.02
LCAN2 (128)	3.65	8.03	13.46	21.89	65.48	sat	**31.67**	0.50	0.54	0.04
Hyp (16)	1.13	2.54	3.73	5.30	15,10	48.94	**19.41**	0.63	0.72	0.09
Hyp (32)	0.3	1.54	2.53	4.1	13.75	38.95	**15.70**	0.62	0.67	0.05
Hyp (64)	-0.1	-0.02	0.2	1.5	4.63	28.4	**9.52**	0.61	0.62	0.01

Table 4.8: Gain de l'adaptativité en % de la latence (TM=64)

Les résultats des simulations que nous avons effectué sur les différentes topologies présentées précédemment montrent que la latence moyenne maximale est au maximum

Réseaux	Charge					Moyenne	Saturation		Gain
	0.1	0.2	0.3	0.4	0.5		Det.	Ad.	(%)
LCAN2 (32)	1.66	1.66	0.25	6.01	17.22	**7.72**	0.58	0.57	-0.01
LCAN2 (64)	1.03	2.75	2.93	6.32	11.04	**6.31**	0.55	0.54	-0.01
LCAN2 (128)	1.78	3.03	4.36	7.00	44.71	**18.04**	0.50	0.51	0.01
Hyp (16)	0.66	2.23	4.67	9.60	26.30	**12.60**	0.58	0.66	0.08
Hyp (32)	0.83	1.52	1.76	1.85	26.63	**9.98**	0.56	0.58	0.02
Hyp (64)	-0.12	1.00	-1.23	0.50	7.38	**2.47**	0.53	0.54	0.01

Table 4.9: Gain de l'adaptativité en % de la latence (TM=128)

Réseaux	Charge				Moyenne	Saturation		Gain
	0.1	0.2	0.3	0.4		Det.	Ad.	(%)
LCAN2 (32)	2.59	4.20	2.44	10.68	**6.10**	0.48	0.52	0.01
LCAN2 (64)	4.30	2.21	5.84	42.04	**19.01**	0.47	0.47	0.01
LCAN2 (128)	3.74	3.96	12.59	7.79	**7.68**	0.43	0.44	0.01
Hyp (16)	0.50	2.07	7.99	-0.11	**2.81**	0.57	0.58	0.01
Hyp (32)	-0.06	2.22	15.78	5.58	**7.40**	0.47	0.53	0.06
Hyp (64)	1.07	2.35	-1.15	12.67	**5.3**	0.46	0.46	0

Table 4.10: Gain de l'adaptativité en % de la latence (TM=500)

de l'ordre de 2000 cycles. Celà représente 24 us (pour une taille de messages de 500 caractères et un cycle de 12 ns). Au niveau application, le temps de latence du logiciel est de l'ordre de 100 ns (corresponds à l'ordre de grandeur du temps d'execution d'une routine système). Un gain même de 100 % sur la latence n'a que peu d'effets sur les performances. Par contre, la bande passante constitue plus souvent une donnée critique. Certaines applications génèrent des charges élevées. Le recours à l'adaptativité peut constituer une solution qui éviterait de changer la topologie ou de réaliser une extension matérielle coûteuse.

Les tableaux 4.8, 4.9 et 4.10 présentent le gain en latence et en bande passante de l'adaptativité pour deux réseaux: le LCAN binaire et l'hypercube. Nous avons évalué ce paramètre pour trois tailles pour chacun des deux réseaux. Les trois tableaux correspondent aux trois tailles des messages.

Le gain en bande passante est peu important et se situe souvent aux environs de 0.01. La plus grande valeur est 0.09. Elle correspond à la différence entre la bande passante effective de l'hypercube adaptatif et déterministe pour un hypercube de 16 processeurs et une taille de messages de 64 caractères. Toutes les autres valeurs importantes ont été obtenues avec l'hypercube. En effet, ceci est dû que l'hypercube adaptatif a été construit non pas en modifiant l'algorithme de routage mais en doublant la bande passante sur certains liens. Ce résultat est donc nécessairement dû à une augmentation topologique de la bande passante.

Pour la latence, nous avons effectué plusieurs calculs sur les valeurs obtenues à partir des simulation, pour faire ressortir des résultats comparables. Les valeurs, représentées dans les différents tableaux sont exprimées en pourcentages du gain absolu en latence. Chaque valeur est calculée en divisant la différence entre les latences du réseau déterministe et du réseau adaptatif par la latence du réseau déterministe. La moyenne présente dans chacun de ces trois tableaux est une moyenne pondérée. Les différentes valeurs de la charge sont prises comme coefficients de pondération. Nous avons choisi de calculer une moyenne pondérée pour faire ressortir les gains correspondant aux charges les plus importantes. La cause de celà est que l'adaptativité devient avantageuse seulement dans le cas où la fréquence des conflits sur les ports de sortie se multiplie. Ce comportement est typique d'un réseau auquel on applique une charge relativement importante. Pour des charges faibles, c'est la topologie qui domine ce comportement (c'est à dire que la latence est seulement déterminée par le diamètre du réseau) car les conflits sont rares. La taille des messages est également importante. En

Réseaux	Moyenne
Grille 2D	**7.33**
LCAN2 (32)	**5.78**
LCAN2 (64)	**17.02**
LCAN2 (128)	**11.81**
Hyp (16)	**6.15**
Hyp (32)	**8.64**
Hyp (64)	**5.16**

Table 4.11: Gain de l'adaptativité en % de la latence: Moyenne totale

effet, un gain de 10 % sur un message de 500 caractères corresponds à un gain d'environ 90 % d'un message de 64 caractères. Dans le but de fournir au concepteur du protocole de communication, qui aura a effectuer le choix d'utiliser ou non l'adaptativité, un paramètre unique qui caractérise le gain en latence de l'adaptativité pour un réseau donné, nous proposons de calculer une moyenne globale pour chaque réseau. Cette moyenne est une moyenne pondérée des moyennes déjà calculées et qui figurent dans les trois tableaux correspondant aux trois tailles de messages. Les coefficients de pondération sont les tailles de messages elles-mêmes. Le tableau 4.11 présente pour chaque réseau le gain moyen ainsi calculé.

Ce tableau indique, pour les réseau considérés un gain se situant entre 5 et 20 %. Ce gain n'est pas négligeable. Cependant, puisque le gain en bande passante est faible, nous considérons qu'il est à ce stade inutile de considérer l'adaptativité. Le choix reste toutefois ouvert et le concepteur devra en tenir compte et effectuer les calculs nécessaires en faisant intervenir la taille moyenne des messages. Les résultats devront l'aider à effectuer le choix d'utiliser ou non l'adaptativité.

4.6 Conclusion

Nous avons effectué un grand nombre de simulations dans le but de déterminer le réseau d'interconnexion de la machine MPC. Nous avons, pour cet effet, conçu et développé un environnement complet de simulation dédié à l'évaluation des performances de réseaux d'interconnexion. Nous avons développé deux modèles de composants: un modèle du routeur RCUBE et un modèle de processeur. Utilisant RCUBE, nous avons montré l'étiquetage de différentes topologies. Certaines d'entre-elles sont directement adaptées aux spécifications de la machine MPC. Pour d'autres, nous avons dû effectuer des transformations structurelles pour les rendre conformes. Les résultats de simulation montrent la domination de l'hypercube qui apparaît comme le réseau présentant globalement les meilleures performances. De plus, son comportement est régulier pour toutes les tailles simulées. D'autres réseaux ont obtenu des résultats satisfaisants et offrent plus de possibilités d'extension que l'hypercube. Nous pensons notamment à la grille composée et au LCAN binaire. Ce dernier n'est pas directement conforme à MPC. Sa version transformée n'a pas atteint des niveaux de performances comparables. Il est nécessaire d'expérimenter d'autres manière de le transformer.

La quantification du gain que procure l'adaptativité a donné de faibles valeurs pour la bande passante et des valeurs moyennes se situant entre 5 et 20 % pour la latence. Le faible gain en bande passante est quasiment éliminatoire pour la l'adaptativité puisque c'est ce paramètre qui est le plus significatif actuellement car même un gain de 100 % sur la latence n'aura que peu d'effet sur les performances. Toutefois, l'adaptativité procure des avantages certains pour les applications où la latence est un facteur critique. Ainsi, avant de prendre une quelconque décision quant à son utilisation, il est nécessaire dévaluer le surcoût induit par un protocole de communication plus complexe et de le comparer avec le gain que procure l'adaptativité.

CHAPITRE 5

STRATÉGIES DE BUFFERISATION

5.1 Introduction

On présente dans ce chapitre une étude de différentes stratégies de mémorisation visant à définir une nouvelle architecture de routeur, de façon à obtenir un seuil de saturation plus élevé que celui du routeur RCUBE. Cette étude a été menée en utilisant l'environnement de simulation MILE.

5.2 Etude comparative de deux crossbars réalisés avec RCUBE

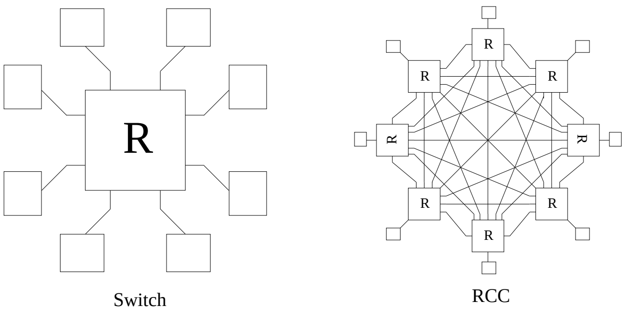


Figure 5.1: Les deux réseaux crossbars réalisés avec RCUBE

Le routeur RCUBE est un crossbar à 8 entrées à mémorisation en entrée. Il est dépourvu de structure de mémorisation centrale. La mémorisation en entrée de type FIFO est souvent la source principale de la limitation des performances [32] [33] [34]. Celle-ci est due au blocage inutile et "abusif" de certains messages qui, parce qu'ils sont en attente d'un port occupé, barrent le passage à ceux qui sont derrière eux.

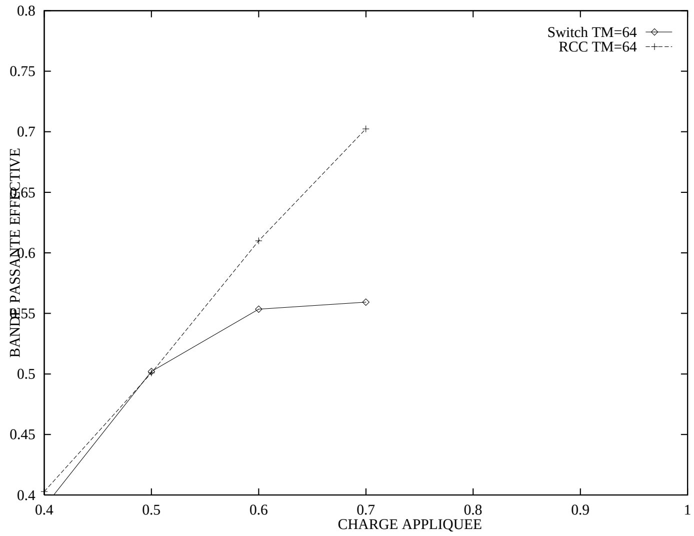


Figure 5.2: Bande passante effective des réseaux Switch et RCC

La figure 5.2 présente des résultats de simulations de deux réseaux réalisés avec RCUBE: le crossbar simple (Switch) et le réseau complètement connecté RCC qui sont deux réseaux à 8 processeurs. Cette figure présente la bande passante effective en fonction de la charge appliquée pour chacun des deux réseaux. Il apparait clairement que le Switch atteint la saturation bien avant le RCC.

En réalité le RCC n'est qu'un crossbar comme le Switch. La bande passante qu'il met à la disposition des processeurs est la même que celle fournie par le Switch car dans les deux cas le routage est déterministe. La seule différence entre les deux est la structure de mémorisation. Modélisons le RCC par un crossbar simple pour bien représenter sa structure de mémorisation. La figure 5.3 montre que ce routeur ainsi modélisé possède une mémorisation en entrée et en sortie. Chaque entrée dispose de son propre crossbar. Celui-ci agit comme un aiguilleur à une seule entrée. Celà fait que l'entrée ne se bloque jamais à cause de l'indisponibilité d'une sortie. Elle ne se bloque que si la FIFO de sortie est déjà pleine et que le port de sortie ne lui est pas attribué. Cette mémorisation en sortie a un impact réel puisqu'elle soulage les messages bloqués derrière d'éventuels messages-barrages en récupérant ces derniers à son niveau. Là, ils peuvent attendre la disponibilité du port de sortie en causant moins de gêne. Cette structure fonctionne de manière optimale pour une taille de messages de 64 caractères puisque c'est également la taille d'une FIFO. Par ailleurs, la taille globale de la FIFO a été doublée. Un message arrivant sur une entrée est directement aiguillé vers la FIFO de sortie correspondant au port qu'il a demandé. Il peut arriver que celui-ci soit déjà alloué. Ce message se bloque alors. Pendant ce temps, le reste du message continue a progresser puisque les 2 FIFOs sont vides et peuvent le contenir. Ce temps de remplissage des FIFOs est doublé par rapport à RCUBE. Il permet au port d'entrée d'être utilisé plus longtemps ce qui augmente la bande passante en entrée. De plus,

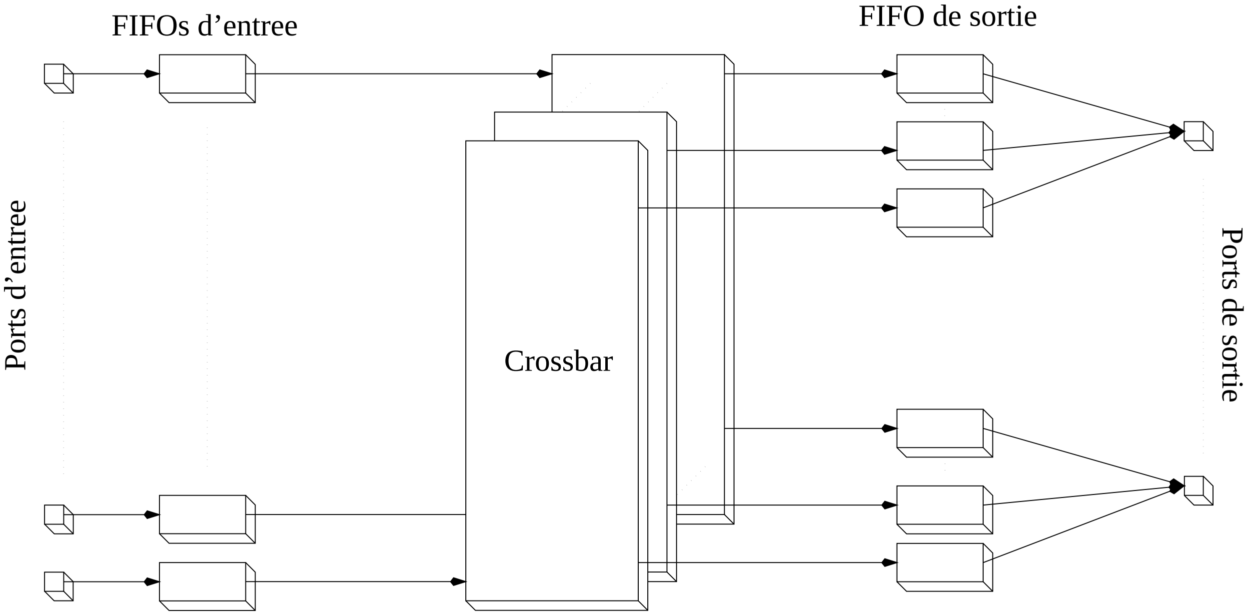


Figure 5.3: Modèle du réseau RCC

quand ce message se verra attribuer le port de sortie, il mettera 64 cycles de moins qu'un message dans RCUBE pour libérer le message qui est derrière lui. Ceci augmente la bande passante en sortie car les messages étant bloqués moins longtemps, les ports de sortie transfèrent de l'information utile plus fréquemment.

Nous venons de voir qu'une modification de la structure de mémorisation a un impact certain sur les performances. Il est alors judicieux de penser à améliorer l'architecture de RCUBE en y incluant une structure de mémorisation centralisée. Il est nécessaire pour celà d'effectuer des expérimentations pour déterminer l'organisation de ce tampon et de ses différents paramètres.

5.3 Stratégies de mémorisation

Dans [35], il fut établi que le débit effectif des réseaux d'interconnexion est limité à une fraction (typiquement 20% à 50%) de la bande passante maximale. Ce résultat est la conséquence du comportement des seuls routeurs puisque le mode de fonctionnement des liens n'a aucun impact sur le débit.

Pour limiter les phénomènes de contention, il faut disposer d'une capacité de mémorisation dans le réseau, c'est à dire dans les routeurs. Quelle que soit le type d'organisation adoptée, elle doit assurer la fonction de contrôle du flux des messages.

5.3.1 Mémorisation en entrée

Le schéma classique de la plupart des routeurs d'aujourd'hui est que les tampons FIFO sont placés au niveau des ports d'entrée. Ce schéma, utilisé par RCUBE, est le plus facile à implémenter et possède plusieurs avantages tel que le support de paquets de longueur variable. Cependant, ce type d'organisation est la cause principale des problèmes de contention.

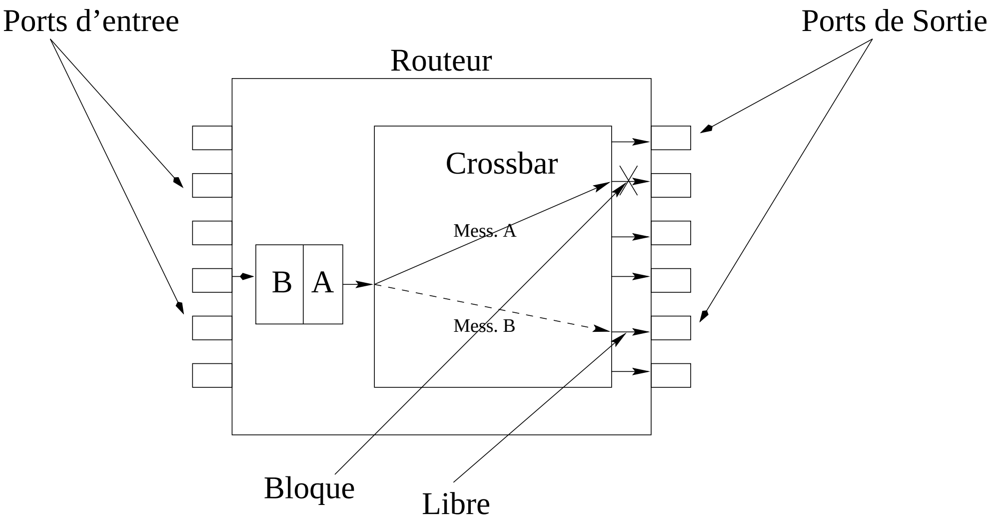


Figure 5.4: Problème de queuing dans un routeur

Dally [32], a donné une description de la cause principale de la faiblesse des tampons FIFOs en entrée. Au niveau d'un tampon FIFO, un message bloqué constitue un obstacle pour ceux qui le suivent (Figure 5.4). Il est clair que le message B est dans un état d'attente tant que le message A n'a pas été servi. Cet état d'attente est superflu puisque la ressource demandée (port de sortie) était initialement disponible.

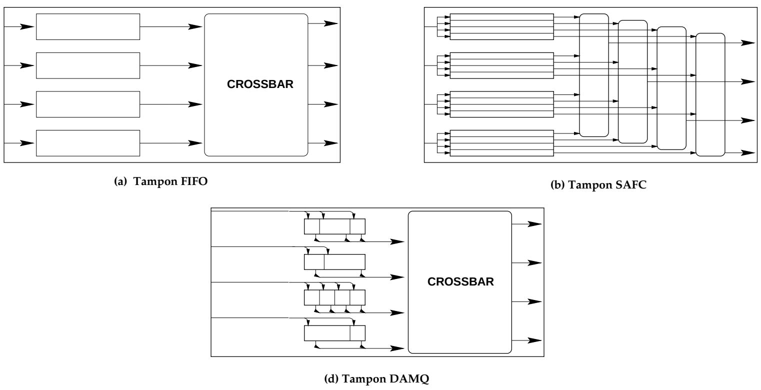


Figure 5.5: Schémas de mémorisation en entrée

Afin d'améliorer la mémorisation en entrée, plusieurs schémas d'implémentation ont été proposés et discutés [33]. Le premier schéma est appelé SAFC (pour *Statically Allocated Fully Connected*). Ce schéma consiste en une multiplication des queues en entrée d'un même port. Les différentes sorties de celles-ci sont connectées à plusieurs bus. Chacun de ces bus réalisant l'interconnexion entre tous les ports d'entrée et un seul port de sortie à raison d'une seule queue connectée par port d'entrée (voir figure 5.5 (b)). Ce schéma présente certains inconvénients: Il implique une faible optimisation de l'utilisation des mémoires FIFO et nécessite un pré-routage des paquets arrivants.

Le dernier schéma proposé est le DAMQ (pour *Dynamically Allocated Multi-Queue*). Ce schéma élimine une partie des inconvénients des schémas précédents notament en améliorant la gestion de la mémoire (Figure 5.5 (d)). Cependant, ce schéma est complexe et n'atteint pas le niveau de performances escomptées [33].

5.3.2 Mémorisation en sortie

Théoriquement, la meilleure solution est de placer les tampons au niveau des ports de sortie. Si on arrive à mettre ensemble tous les messages destinés à un même port alors on évite les phénomènes de contention parasites décrits précédemment. Ce schéma pose

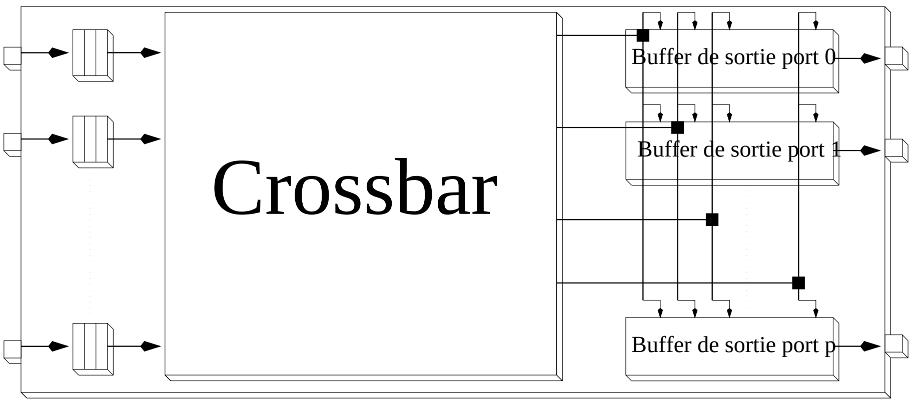
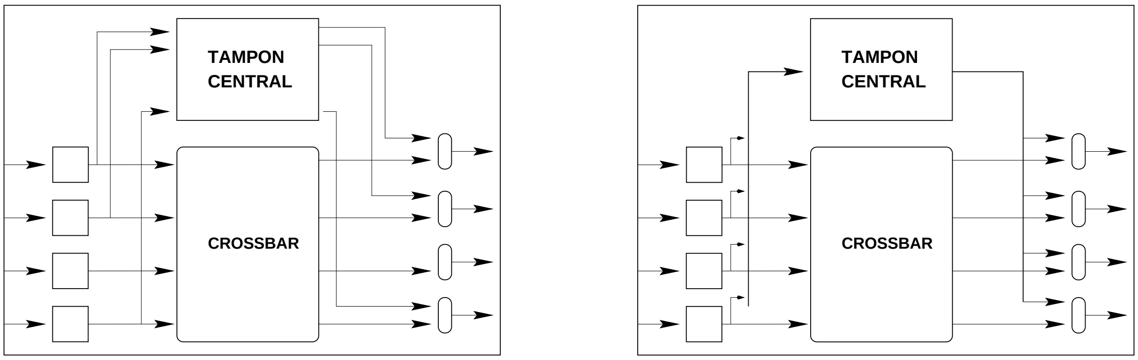


Figure 5.6: Schémas de mémorisation en sortie

cependant deux contraintes majeures au stade de l'implémentation. Le tampon alloué à un port doit avoir autant d'accés en écriture que le nombre d'entrées du crossbar. Cette contrainte se traduit par la conception d'un tampon multi-ports. Celà introduit un problème d'arbitrage au niveau de l'allocation de l'espace. Ces deux fortes contraintes imposent une compléxité accrue au niveau implémentation.

5.3.3 Mémorisation centralisée



Memoire multi-ports   Parallelisation des acces

Figure 5.7: Deux manières d'implémenter un tampon centralisé

Une mémoire centralisée peut faire fonction de tampon de telle façon que les messages bloqués peuvent y séjourner jusqu'a ce qu'ils obtiennent un port de sortie. L'implémentation de ce type de schéma est difficile puisque la réalisation de la mémoire impose d'avoir soit une bande passante élevée soit avec des ports d'accés multiples, soit avec des mots larges. La seconde solution introduit une augmentation de la latence et une compléxité accrue au niveau des ports (parallélisation des mots successifs d'un même port pour être conformes à la largeur du tampon centralisé). Cette solution a cependant été retenue dans plusieurs architectures de routeurs parmi lesquels le Vulcan d'IBM [14].

5.3.4 Résumé

L'organisation des tampons au niveau des routeurs à un impact réel sur les performances d'un réseau d'interconnexion et particulièrement sur le seuil de saturation. Le schéma le plus simple à implémenter est la mémorisation en entrée de type FIFO. Ce schéma est la cause principale de la sous-utilisation de la bande passante dans RCUBE. Le schéma de mémorisation en sortie requiert une implémentation complexe. Le tampon centralisé, dans l'une de ses implémentations est simple à rèaliser mais impose une cadence de transfert très élevée. Dans une autre implémentation, il introduit une compléxité similaire à celle du schéma de mémorisation en sortie.

Dans la majorité des cas, toute amélioration de ce schéma introduit une compléxité au niveau matériel. Cette compléxité est essentiellement dûe à:

- une multiplication des ports d'accés.
- une gestion simultanée de plusieurs requêtes.

Ces deux points décrivent les fonctionnalités d'un crossbar. Dans ce qui suit, nous proposons un schéma de tampon centralisé. Ce schéma particulier est basé sur l'utilisation du crossbar du routeur pour réaliser l'interconnexion entre ce tampon, les ports d'entrée et les ports de sortie.

5.4 Modèle de routeur générique

Un modèle de routeur générique a été défini par B. Zerrouk et moi-même pour expérimenter différentes stratégies de mémorisation.

L'architecture du routeur générique est composée de ports bidirectionnels liés à travers un crossbar et d'un espace mémoire dédié à la mémorisation. Chaque port est une paire constituée d'un port d'entrée et d'un port de sortie. Chaque port d'entrée contient une petite FIFO pour gérer le contrôle de flux suivant la même technique de crédit que RCUBE.

L'espace de mémorisation est organisé en une série de tampons FIFO disposés en parallèle. Leurs sorties sont connectées à des entrées suuplémentaires du crossbar et leurs entrées sont connectées aux sorties de celui-ci de manière à former une structure à rebouclage. La structure du crossbar permet d'interconnecter les sorties des tampons et les ports d'entrée d'un coté aux entrées des tampons et aux ports de sortie d'un autre coté. La connexion de la sortie d'un tampon à l'entrée d'un autre tampon n'a aucun intérêt et est donc interdite (voir figure 5.8). A chaque port d'entrée est associée une

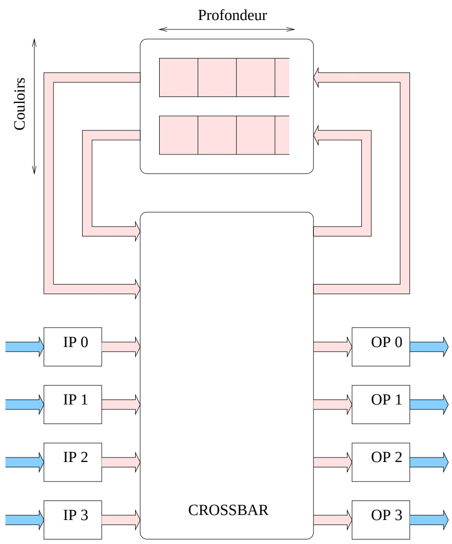


Figure 5.8: Architecture du routeur générique

table de routage complète. Chaque entrée représente une adresse d'un noeud donné du réseau. Elle contient la liste ordonnée des ports de sortie qu'un message, destiné à ce noeud, est autorisé à emprunter. Elle contient également la liste des tampons vers lesquels ce message, dans le cas où les ports de sortie qu'il demande sont indisponibles, peut être dirigé.

Le nombre de tampons et leurs tailles sont paramétrables. La programmabilité de l'allocation des tampons ainsi que la possibilité de changer leur nombre et leur taille sont les facteurs clés qui permettent d'émuler différentes stratégies.

5.5 Détermination du nombre et de la taille des tampons

Le travail effectué ici a été mené en coopération avec la société BULL dans le cadre du projet TWIN. Ce projet vise la conception d'un routeur de nouvelle génération qui sera le successeur de RCUBE.

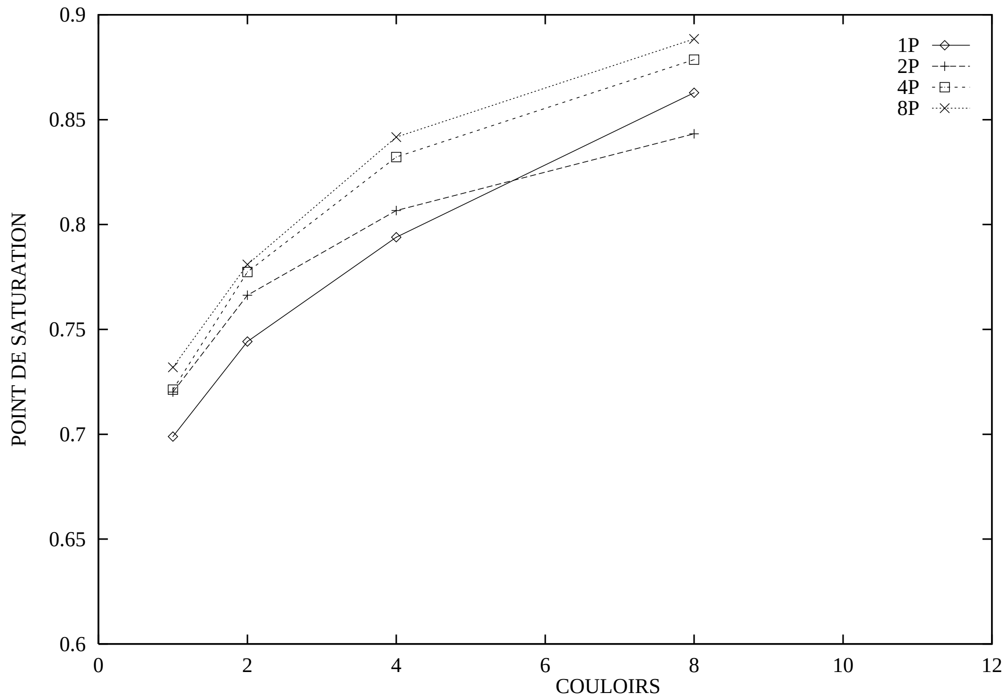


Figure 5.9: Bande passante effective en fonction du nombre de couloirs

Le routeur TWIN est destiné à router des lignes de caches dans un réseau d'interconnexion d'une machine parallèle à mémoire partagée. De ce fait, il possède une taille de messages maximale fixée à 128 caractères. Il implémente la même structure à rebouclage que celle du modèle générique présenté ci-dessus. Chaque port possède une FIFO d'entrée de 64 caractères. Le terme couloir désigne une FIFO appartenant au tampon central. Le terme profondeur désigne la taille de cette FIFO. Cette taille est exprimée en multiples de la taille maximale d'un message. Ainsi, une taille unitaire corresponds à 128 caractères. Les couloirs sont numérotés. Pour déterminer le nombre optimal de couloirs et la profondeur de FIFO optimale, nous avons choisi la stratégie d'allocation des tampons la moins avantageuse qui est la mémorisation en entrée. Celà veut dire qu'un message dont le port de sortie est indisponible est dirigé vers le couloir non plein d'indice le plus petit.

Quatres valeurs ont été retenues pour faire varier le nombre de couloirs et la profondeur. Ces valeurs sont les mêmes pour les deux paramètres: 1,2,4 et 8. Pour chaque valeur du nombre de couloir nous avons effectué 4 simulations correspondant aux 4 tailles de la profondeur. Neuf simulations sont nécessaire pour chaque valeur du nombre de couloir et de la profondeur. Celà corresponds à la variation de la charge de 10 à 90 % de la bande passante maximale. Le modèle de génération des messages est identique à celui utilisé pour l'évaluation des réseaux présentés précédemment. Le réseau simulé est un crossbar composé d'un seul routeur à 4 ports et de 4 processeurs.

La figure 5.9 présente la bande passante effective (point de saturation) en fonction du nombre de couloirs pour les quatre tailles de la profondeur. Les résultats montrent que le comportement du crossbar simple a été considérablement améiloré par l'ajout d'une structure de mémorisation centrale. Toutes les simulations donnent des résultats supérieurs à 70 %. Il en ressort que les meilleures valeurs sont de 8 couloirs à profondeur 8. Cependant, la configuration à 8 couloirs et à profondeur 4 obtient des résultats trés proches tout en étant moitié moins coûteuse.

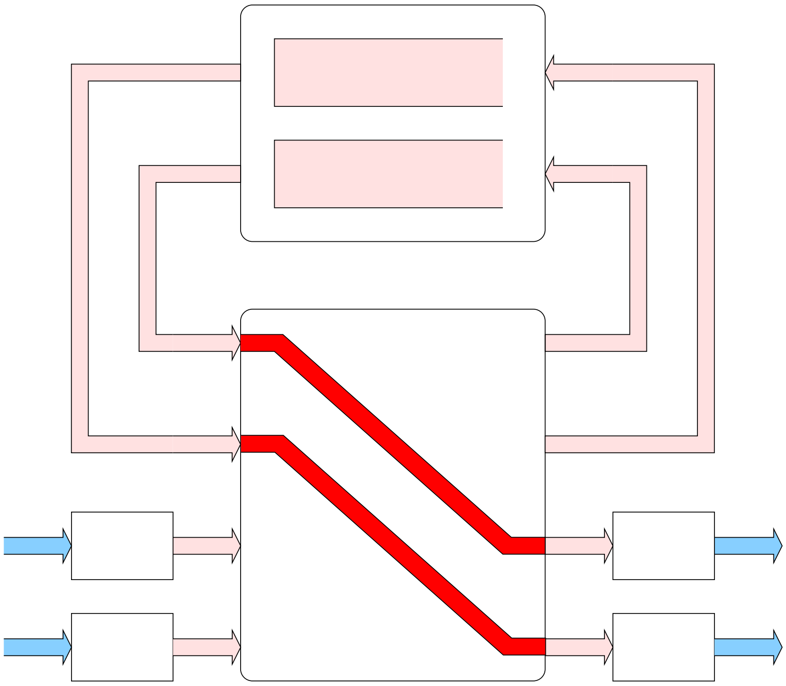


Figure 5.10: Stratégie de mémorisation en sortie

5.6 Détermination de la stratégie d'allocation

Dans le but de déterminer la meilleure stratégie d'allocation, une autre série de simulation a été effectuée. Nous avons vu au paragraphe 5.3 que la mémorisation en sortie est plus performante que la mémorisation en entrée. L'émulation par notre modèle de la mémorisation en sortie se réalise en assignant chaque couloir à un port de sortie distinct. Pour éviter les problèmes d'arbitrage, nous avons choisi un nombre de couloirs égal au nombre de ports du routeur. Si un message arrivant sur l'entrée du routeur n'a pas été satisfait, il est dirigé vers le couloir associée au port de sortie qu'il a demandé (voir figure 5.10). Un avantage de cette émulation est que les messages sont préroutés

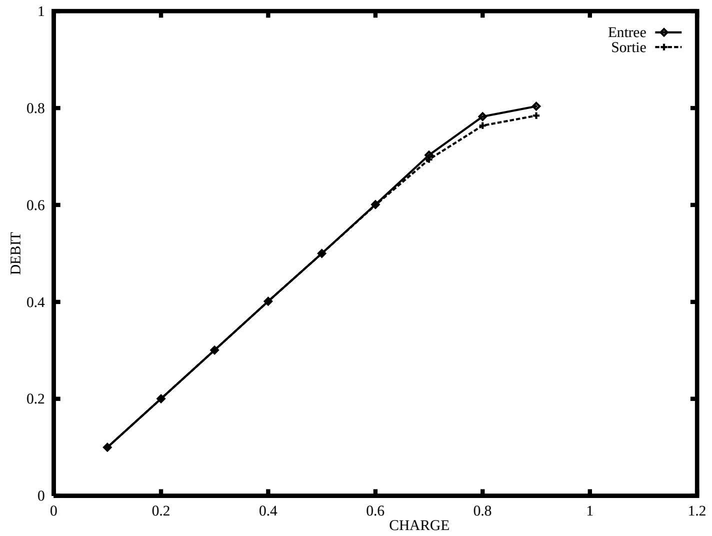


Figure 5.11: Bandes passantes obtenues par les deux stratégies de mémorisation

dès leur arrivée. Les messages contenus dans le même couloir ont toujours la même destination (port de sortie). Le seul inconvénient est qu'un message bloqué à cause d'un couloir plein restera bloqué au niveau de la FIFO d'entrée même s'il existe de la place pour l'accueillir dans un autre couloir.

Nous avons évalué cette stratégie d'allocation qui émule la mémorisation en sortie et l'avons comparée à la stratégie d'allocation qui implémente une mémorisation en entrée. La figure 5.11 présente les bandes passantes des deux réseaux correspondant aux deux stratégies. Cette courbe montre que les débits sont du même ordre de grandeur. La figure 5.12 qui présente les latences, montre par contre une différence nette au profit de la stratégie de mémorisation en sortie. Celà confirme les affirmations rencontrées dans la littérature.

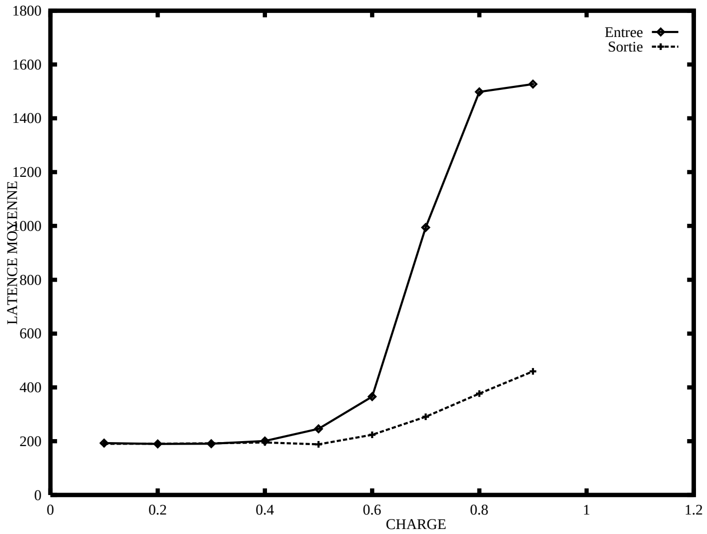


Figure 5.12: Latences obtenues par les deux stratégies de mémorisation

5.7 Conclusion

L'étude des performances de la structure de mémorisation de RCUBE à travers les résultats des deux crossbars (Switch et RCC) a montré le besoin de disposer d'une mémorisation centralisée. Nous avons mené une série de simulations dans le but de déterminer l'architecture de ce tampon centralisé.

L'architecture a été définie en connectant une série de FIFOs au crossbar de manière a former une structure de rebouclage. Un modèle de routeur générique a été développé et utilisé pour expérimenter diverses tailles de FIFOs et diverses stratégies d'allocation. Le résultat obtenu est que la mémorisation en sortie est plus performante que la mémorisation en entrée, confirmant les affirmations de [33].

CHAPITRE 6

CONFIGURATION AUTOMATIQUE DE TOPOLOGIES QUELCONQUES

6.1 Introduction

Dans ce chapitre, nous présentons un algorithme permettant de construire de façon automatique le schéma de routage par intervalle pour un réseau de topologie quelconque, dans le cas où il peut y avoir un nombre quelconque de processeurs attaché à chaque noeud du réseau.

Les machines massivement parallèles reposant sur l'hypothèse d'une topologie régulière sont fortement vulnérables aux pannes: il suffit d'un seul lien défaillant pour rendre le système inutilisable. Heureusement, les réseaux d'interconnexion utilisant des routeurs dynamiques tels que RCUBE supportent des topologies irrégulières: Cette particularité améliore fortement la résistance aux pannes: en cas de défaillance d'un noeud ou d'un lien, il suffit de recalculer les tables de routage correspondant à la topologie "dégradée" résultant de la suppression du noeud ou du lien en panne.

6.2 Algorithme de Van Leeuwen-Tan

Van Leeuwen-Tan [36] ont mis au point un algorithme d'étiquetage de topologies quelconques dans le cas où un et un seul processeur est attaché à chaque noeud du réseau. L'idée directrice de cet algorithme consiste à construire un arbre recouvrant sur la topologie considérée. Cet arbre est construit en utilisant tous les noeuds du réseau et une partie de ses liens. Partant d'un noeud quelconque du réseau, l'arbre est construit récursivement jusqu'à ce que tous les noeuds aient été visités. Les noeuds de l'arbre sont numérotés suivant un ordre croissant. Cet algorithme est constitué par la

```
PROCEDURE Etiquetter(u,v)
/* u et v sont des noeuds du reseau */
/* u est le pere de v dans l'algorithme de recherche recursive */
/* v est le noeud courant */

DEBUT
    l(v) = i ;     /* l(v) represente l'adresse du noeud v */
    i = (i+1) mod N;

    POUR chaque noeud w adjacent a v
      FAIRE
            SI w n'est pas numerote
              ALORS
                    Numeroter le port v -> w avec i
                    Etiquetter(v,w);
              SINON
                    SI w =/= u
                      ALORS
                            Numeroter le port v -> w avec l(w)
                    FINSI
            FINSI
      FINFAIRE
    SI v =/= x
      ALORS
            Numeroter le port v -> u avec i
    FINSI
FIN
```

Figure 6.1: Procédure principale de l'algorithme de Van Leeuwen-Tan

procédure récursive *Étiqueter(u,v)*. Au début du parcours, la variable i est initialisée à 0, et la procédure est appelée par *Étiqueter(x,x)* où x représente un noeud quelconque du réseau choisi au hasard. Cette procédure est présentée en figure 6.1. N représente le nombre total de noeuds du réseau. i est une variable globale qui varie de 0 à $N-1$.

Cet algorithme se déroule comme suit:

1 - Choisir un noeud x quelconque du réseau.

2 - Donner à ce noeud x l'adresse 0.

3 - Étiqueter le premier port de ce noeud avec l'adresse 1.

4 - Donner l'adresse 1 au noeud suivant (de l'autre côté du lien attaché au port 1 du noeud x).

5 - Explorer tous les noeuds attachés à chaque port de ce noeud.

6 - A chaque fois qu'on arrive au niveau d'un noeud celui-ci devient le noeud courant v. On lui attribue l'adresse courante i et on incrémente celle-ci d'une unité (modulo N).

   – Considérer chaque port de v sauf le port qui le lie à u, son père, par lequel v a été atteint.
   – Si ce port est attaché à un noeud déjà visité w, alors étiqueter ce port avec l'adresse de ce noeud déjà visité.
   – Si ce port est attaché à un noeud non-visité, alors, l'étiqueter avec le nouveau i et atteindre ce noeud pour qu'il devienne le nouveau noeud courant.

7 - Si au niveau d'un noeud tous les ports ont été parcourus alors revenir en arrière. En effectuant le parcours arrière, étiqueter chaque port (dans le sens inverse du parcours initial) avec $i + 1$ *modulo* N.

8 - L'algorithme s'arrête dés que l'on arrive au noeud de départ et que tous ses liens ont été parcourus.

Les caractéristiques de l'étiquetage obtenu sont les suivantes (voir figure 6.3)

- Il y a deux sortes de liens: les liens qui font partie de l'arbre recouvrant (au traits pleins sur la figure 6.3.b) et les liens qui ne font pas partie de l'arbre recouvrant (aux traits pointillés sur la figure 6.3.b)

- Tous les noeuds appartenant à un sous-arbre ont des adresses supérieures ou égales à l'adresse de la racine du sous-arbre.

- Pour ce qui concerne les liens appartenant à l'arbre recouvrant, le port de sortie d'un noeud y permettant de "descendre" dans l'arbre (de la racine vers les feuilles) a pour index l'adresse du noeud adjacent à y. Le port de sortie permettant de "remonter" dans l'arbre (des feuilles vers la racine) a un index défini par le maximum de toutes les adresses des noeuds appartenant au sous-arbre correspondant au noeud y (modulo N).

- Pour ce qui concerne les liens n'appartenant pas à l'arbre recouvrant, les ports de sortie ont pour index l'adresse du noeud adjacent. Les liens n'appartenant pas à l'arbre recouvrant sont cependant utilisés pour le routage: ils constituent des ""raccourci"" pour circuler dans l'arbre.

A partir de ces caractéristiques, on peut construire un schéma de routage par intervalle. Si on supprime tous les liens qui n'appartiennent pas à l'arbre recouvrant, la stratégie de routage est évidente et les tables de routage par intervalle sont définies comme suit:

- Soient p_i les M ports d'un routeur quelconque et $L(p_i)$ les index attachés aux ports.

- On classe les M ports de telle sorte que:

$$L(p_0) \leq L(p_1) \leq \ldots \leq L(p_{M-1})$$

- On définit l'intervalle associé au port p_i comme $[L(p_i), L(p_{i+1})[$ (avec $p_M = p_0$).

Analysons maintenant comment l'étiquetage généré permet d'utiliser les ""raccourci"". Soient deux noeud X et Y qui appartiennent chacun à deux branches distinctes d'un sous-arbre de l'arbre recouvrant issu de l'application de l'algorithme de Van Leeuwen-Tan. Soit A le noeud racine de ce sous-arbre. A constitue l'ancêtre commun le plus proche de X et de Y. Ce sous-arbre a été construit comme suit:

- Attribuer une adresse au noeud A.
- Explorer toute la branche contenant le noeud X ainsi que ses ramifications.
- Explorer ensuite toute la branche contenant le noeud Y ainsi que ses ramifications.

Si X et Y étaient liés par un lien "raccourci", le noeud Y aurait été exploré et numéroté juste après avoir atteint le noeud X, ce qui est contraire à l'hypothèse. Par conséquent, on peut affirmer que tout lien "raccourci" relie obligatoirement un noeud à l'un de ses ancêtres (ou respectivement l'un de ses descendants).

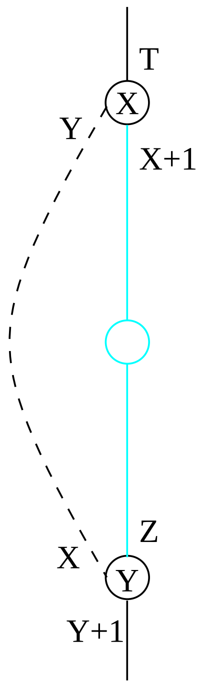


Figure 6.2: Étiquetage de deux noeuds reliés par un lien "raccourci"

La figure 6.2 montre l'étiquetage des noeuds X et Y. Dans cette figure, on considère que X est l'ancêtre de Y. L'adresse de X est donc inférieure à celle de Y. Z représente le maximum de toutes les adresses des descendants du noeud Y. T représente le maximum de toutes les adresses des descendants du noeud X. Le lien "raccourci" est étiqueté avec l'intervalle $[Y, T[$ dans le sens $X \rightarrow Y$ et avec l'intervalle $[X, Y[$ dans le sens $Y \rightarrow X$. Ainsi, le routage suivant ce lien est décrit par ce qui suit:

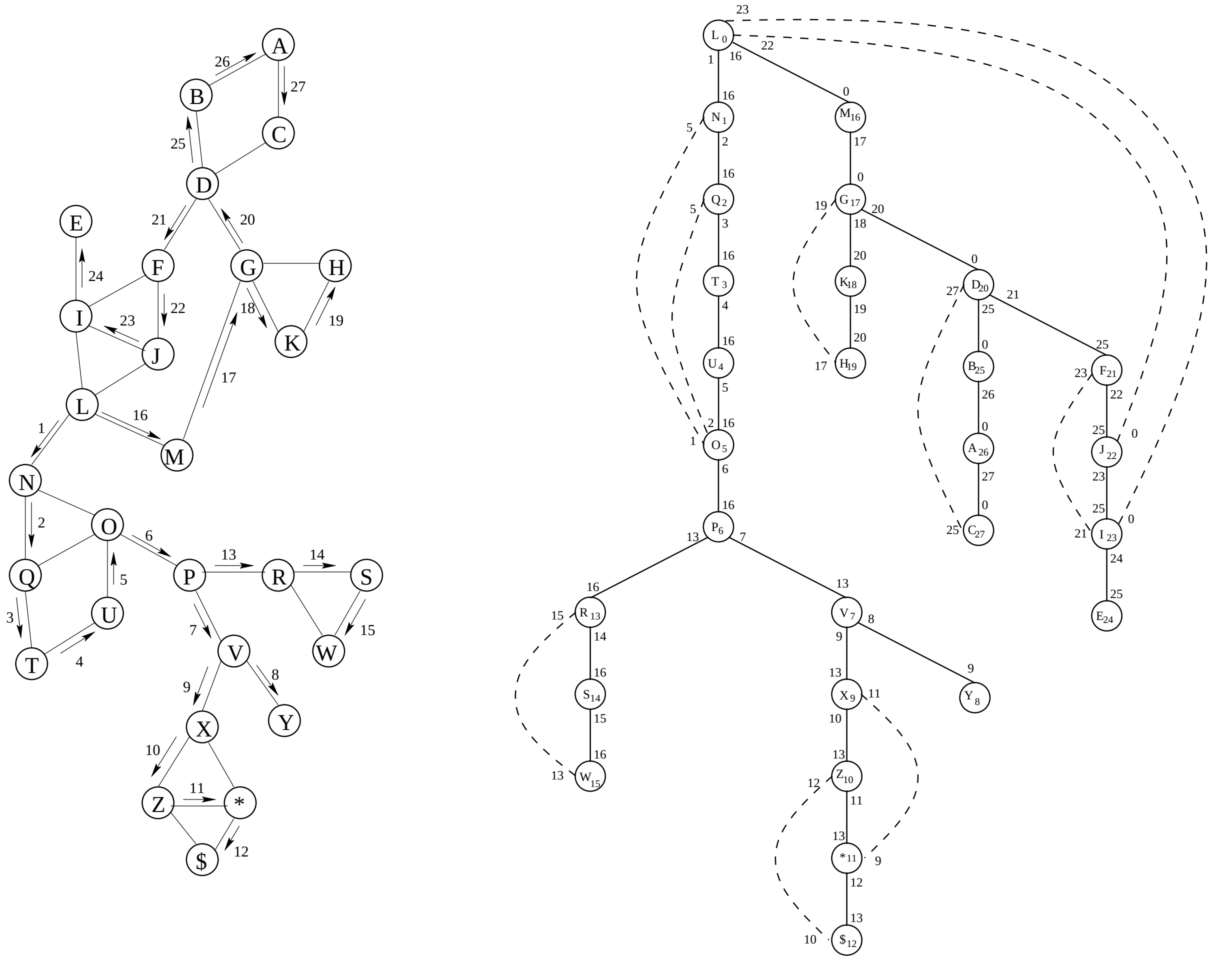
- Dans le sens $X \rightarrow Y$: Le lien raccourci route les messages destinés vers Y et ses descendants et les descendants de X qui ne sont pas des ancêtres de Y (qui ont des adresses comprises entre Z et T).
- Dans le sens $Y \rightarrow X$: Le lien raccourci route les messages destinés vers les ancêtres de Y qui sont des descendants de X.

Le lien "raccourci" entre les noeuds X et Y est donc utilisé en descente pour atteindre des descendants de X et en montée pour atteindre des ancêtres de Y. Ce schéma est conforme au routage dans un arbre. Donc l'utilisation des liens "raccourci" est correcte.

Remarque: Cet algorithme permet que deux ports d'un même noeud possèdent tous deux l'index 0. Supposons en effet qu'un noeud v possède un port vers le noeud racine x. Ce port va être étiqueté avec l'adresse 0. Cependant, v peut ne pas avoir été directement atteint lors du parcours initial depuis le noeud x mais depuis un autre noeud u. En revenant vers u, il se peut qu'a ce moment l'adresse courante i vaut justement $N-1$. La même étiquette 0 va donc être assignée au port $v \rightarrow u$. Ce conflit est résolu en assignant l'adresse du noeud u à ce port.

La figure 6.3 présente un exemple de réseau quelconque étiqueté avec l'algorithme de Van Leeuwen-Tan. Les flèches dans la figure 6.3.a représentent le sens de parcours. Les numéros à l'intérieur des noeuds sont les adresses de ces noeuds. Les numéros sur les flèches représentent l'ordre de parcours. Nous remarquons que cet ordre correspond parfaitement aux adresses des noeuds. L'arbre recouvrant est composé de tous les noeuds du réseau. Les liens en traits pleins sont ceux qui ont été empruntés lors du parcours initial du réseau. Les liens ""raccourci"" ne faisant pas partie de l'arbre sont représentés en pointillés.

L'algorithme qui vient d'être présenté produit une numérotation complète de tous les noeuds du réseau. Ces numéros représentent les adresses des processeurs connectés à ces routeurs. Cependant, on peut construire des réseaux où certains routeurs ne soient connectés à aucun processeur. Dans ce cas, le schéma de routage produit par l'algorithme de Van Leeuwen-Tan génère un espace adressable plus grand que le nombre total de processeurs. Le fonctionnement est correct car les processeurs connaissent leurs adresses respectives. Cependant, il faut absolument éviter d'envoyer un message dont l'adresse correspond à celle d'un routeur sans processeur. Ce message bloquerait le réseau car, arrivé au routeur correspondant à son adresse destination, il ne trouvera aucune sortie et restera bloqué à ce niveau. Ainsi, une erreur (matérielle ou logicielle) sur l'adresse peut provoquer des blocages. Par ailleurs, l'algorithme de Van Leeuwen-Tan ne traite pas le cas où plusieurs processeurs sont connectés à un même noeud de routage. Ainsi, il est nécessaire de le modifier pour qu'il produise un étiquetage linéaire et continu des processeurs seuls.



a) Parcours du reseau

b) Arbre recouvrant

Figure 6.3: Étiquetage d'un réseau et arbre recouvrant obtenu

6.3 Algorithme modifié

On travaille sur un réseau possédant deux types de noeuds: des noeuds processeurs et des noeuds routeurs. Un processeur est lié à un routeur par un lien unique. Un routeur peut donc être connecté à zéro, un ou plusieurs processeurs.

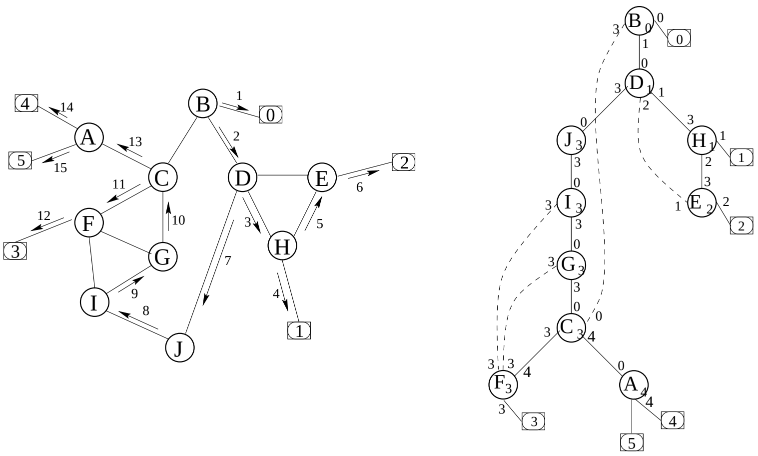
En premier lieu, on modifie la manière d'étiqueter chaque noeud. Les routeurs ne sont pas adressables. Ce sont uniquement les processeurs qui le sont. Quand on arrive à un noeud routeur, on étiquete tous les processeurs rattachés à ce noeud. Les processeurs sont étiquetés de la manière suivante:

- Étiqueter le port routeur → processeur par i

- Le processeur prend l'adresse i

- $i = i + 1$

On assigne cependant à chaque routeur rencontré dans le parcours du réseau une pseudo-adresse égale à l'adresse du prochain processeur à étiqueter. On assigne également un numéro aux routeurs. Ce numéro (que nous notons $Z()$) représente l'ordre de visite de chaque routeur. Cette dernière numérotation est faite en vue d'établir une relation d'ordre stricte entre les routeurs. Cette relation de précédence, qui n'est pas un adressage, permet de résoudre certains conflits comme nous le verrons plus loin. La figure 6.4 présente un réseau comprenant 6 processeurs et 10 routeurs. Appliquant



a) Parcours du reseau b) Arbre recouvrant

Figure 6.4: Application directe de l'algorithme sans numérotation des routeurs

directement l'algorithme de Van Leeuwen-Tan, en partant du noeud B, avec seulement la restriction de ne pas incrémenter la variable globale i pour les routeurs, on obtient une numérotation linéaire des processeurs comme illustré par la figure 6.4.b. Cependant, cette modification génère une erreur d'étiquetage. Les liens reliant les noeuds F et G possèdent la même étiquette des deux cotés. Cet étiquetage incorrect est dû à la non-progression du numéro assigné à des routeurs dépourvus de processeurs visités successivement. Cet étiquetage incorrect concerne seulement les liens "raccourci".

Les liens "raccourci" relient toujours un noeud X à l'un de ses descendants Y. Supposons que X et Y sont dépourvus de processeurs. Supposons également qu'aucun noeud faisant partie du chemin (dans l'arbre recouvrant) qui relie X à Y ne possède de processeur.

- Dans le sens $X \rightarrow Y$, ce lien permet d'atteindre Y et ses descendants et les descendants de X qui ne sont pas des ancêtres de Y. Toutes les adresses de ces noeuds sont supérieures (ou égale) à l'adresse de Y. Comme le premier processeur qu'on atteint se situe après Y, alors l'étiquetage est correct dans cette direction.

- Dans le sens $Y \rightarrow X$, ce lien permet en principe d'atteindre les noeuds qui sont des ancêtres de Y et qui sont en même temps des descendants de X. Cependant, aucun processeur n'est attaché à l'un de ces noeuds. X, Y et les noeuds intermédiaires possèdent la même pseudo-adresse. C'est l'adresse du prochain processeur qu'on devra rencontrer. Ce processeur se trouve après le noeud Y. Rappelons qu'on assigne toujours à un routeur l'adresse du prochain processeur à atteindre. Les deux noeuds X et Y possèderont donc la même pseudo-adresse. Le lien "raccourci" sera donc indexé dans les deux extrémités par le même index qui n'est autre que la pseudo-adresse de chacun des 2 noeuds. Celà crée une boucle. Pour supprimer cette boucle, il suffit d'utiliser le lien "raccourci" pour router les messages destinés non pas vers le processeur qui se situe après Y mais vers le dernier processeur rencontré avant X. Ce processeur a pour adresse la pseudo-adresse de X (ou de Y) decrémentée d'une unité.

```
- inchange -
POUR chaque noeud w adjacent a v
  FAIRE
        SI w n'est pas numerote
          ALORS
                - inchange -
          SINON
                SI w =/= u
                  ALORS
                        SI l(v) = l(w)
                          ALORS
                                SI Z(v) < Z(w)
                                  ALORS
                                        Indexer le port v -> w avec l(w)
                                  SINON
                                        Indexer le port v -> w avec l(w)-1
                                FINSI
                          SINON
                                Indexer le port v-> avec l(w)
                        FINSI
               FINSI
       FINSI
 FINFAIRE
- inchange -
```

Figure 6.5: Deuxième modification de l'algorithme

On implémente celà au niveau de l'algorithme en utilisant le numéro d'ordre pour distinguer deux noeuds de même pseudo-adresse reliés par un lien raccourci. Ce numéro d'ordre indique si le noeud courant est un ancêtre ou un descendant du noeud adjacent. Dans le cas où c'est un ancêtre, l'étiquetage tel qu'il est effectué

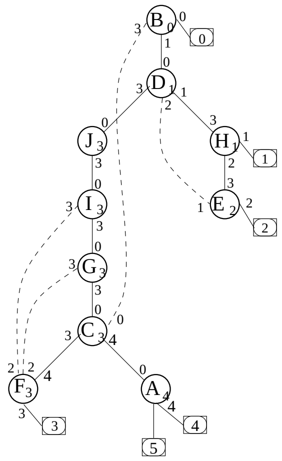
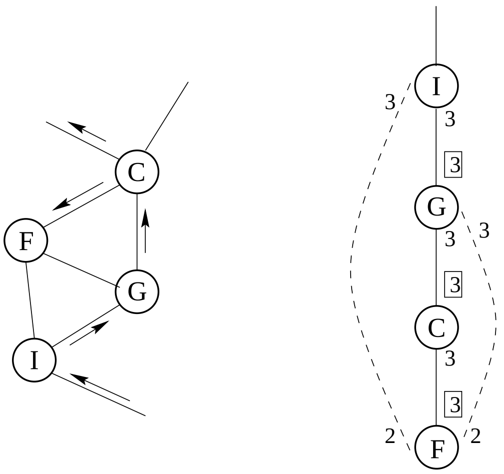


Figure 6.6: Traitement du cas de routeurs de même numéro

par l'algorithme de Van Leeuwen-Tan est conservé. Dans le cas contraire, on étiquete le port en question avec la pseudo-adresse du noeud courant decrémentée d'une unité. La figure 6.5 présente la modification de l'algorithme. La figure 6.6 montre le résultat après introduction de cette modification.



a) Parcours du reseau b) Arbre recouvrant

Figure 6.7: Cas d'une branche contenant des routeurs sans processeurs

Par ailleurs, un autre problème persiste. Cet algorithme génère encore un étiquetage erroné lorsqu'il explore des branches constituées de routeurs auxquels aucun processeur n'est attaché. En effectuant le retour sur une branche de ce type, on indexe les ports de retour (vers le père de chaque noeud courant dans l'arbre recouvrant) avec la même pseudo-adresse contenue dans la variable i qui n'a pas évolué du fait qu'aucun processeur n'a été rencontré (voir figure 6.7).

```
PROCEDURE Etiquetter(u,v)
/* u et v sont des noeuds du reseau */
/* u est le pere de v dans l'algorithme de recherche recursive */
/* v est le noeud courant */

DEBUT
    I(v) = i ;    /* I(v) represente l'adresse du noeud v */
    POUR chaque processeur attache a v
      FAIRE
            Etiquetter le port v->processeur avec i
            Donner l'adresse i au processeur
            i = (i+1) mod N;
      FINFAIRE

    POUR chaque noeud w adjacent a v
      FAIRE
            SI w n'est pas numerote
              ALORS
                    Numeroter le port v -> w avec i
                    Etiquetter(v,w);
              SINON
                    SI w =/= u
                      ALORS
                            SI I(v) = I(w)
                              ALORS
                                    SI Z(v) < Z(w)
                                      ALORS
                                            SI i = I(v)
                                              ALORS interdire le lien dans ce sens
                                              SINON Indexer le port v -> w avec I(w)
                                            FINSI
                                      SINON
                                            Indexer le port v -> w avec I(w)-1
                                    FINSI
                              SINON
                                    Indexer le port v -> w avec I(w)
                            FINSI
                    FINSI
            FINSI
      FINFAIRE
    SI v =/= x
      ALORS
            Numeroter le port v -> u avec i
    FINSI
FIN
```

Figure 6.8: Version finale de l'algorithme

On résout ce problème en interdisant le routage dans cette branche dans le sens de l'exploration de cette branche. En revanche on l'autorise dans le sens inverse (le sens du retour). Celà est nécessaire pour router les messages issus d'éventuels ancêtres reliés à l'un des noeuds de cette branche par un lien "raccourci". Ainsi configurée, cette branche ne fera que remonter tous les messages vers le haut de l'arbre recouvrant. La figure 6.8 présente la version finale de l'algorithme. La figure 6.9 présente l'étiquetage corrigé. L'étiquette -1 sur une direction indique que le routage est interdit suivant cette direction.

Nous avons réalisé et testé un outil logiciel qui implémente l'algorithme que nous venons de décrire. Cet outil accepte en entrée un fichier décrivant la topologie d'un

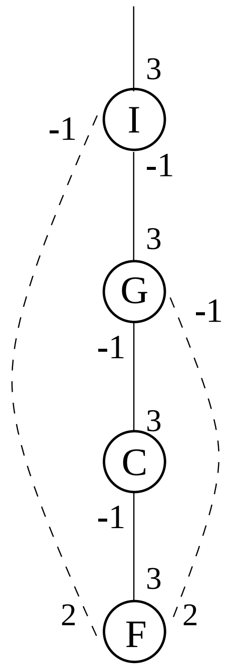


Figure 6.9: Interdiction des liens dans une direction

réseau construit avec des routeurs RCUBE (possédant une valence 8) et des processeurs (sous le format NDL [12] utilisé par l'environnement MILE). En sortie, il produit un nouveau fichier dans lequel il ajoute les tables de routage pour les routeurs du réseau. Ce fichier est directement exploitable par de l'environnement MILE.

Cet algorithme est également utilisé par le logiciel de configuration automatique du réseau HSL de la machine MPC développée au LIP6: les tables de routage calculées pour une topologie particulière (décrite dans un fichier) sont distribuées à tous les processeurs grâce au réseau de contrôle (ethernet) et téléchargées dans les routeurs RCUBE à travers le bus PCI.

6.4 Conclusion

Nous avons présenté dans ce chapitre une amélioration de l'algorithme d'étiquetage automatique de topologies quelconques de Van Leeuwen-Tan. Nous avons modifié cet algorithme dans le but de configurer automatiquement des réseaux utilisant une technique de routage par intervalle où il n'y a pas nécessairement un processeur pour chaque routeur. La particularité de notre algorithme est qu'il produit des adresses consécutives. Ceci permet une détection plus facile des erreurs de transmission lorsqu'elles concernent les en-têtes des messages. Nous avons réalisé et testé un outil logiciel qui implémente cet algorithme.

CHAPITRE 7

VALIDITÉ D'UN SCHÉMA DE ROUTAGE

7.1 Introduction

La théorie que nous allons présenter dans ce chapitre a été développée par B. ZERROUK et moi-même.

- Le but de la théorie est de fournir des outils formels pour vérifier **statiquement** qu'un schéma de routage en routage *wormhole* est valide. C'est à dire qu'un schéma de routage est exact, non bloquant et ne contient pas de boucle infinie. La vérification est statique car elle ne prend en compte que le schéma de routage et la topologie du réseau indépendamment des caractéristiques du trafic.

- Le réseau d'interconnexion est représenté par un graphe fini connexe orienté. Les arcs sont bidirectionnels et sont décomposés pour les besoins de l'étude en deux liens, un dans chaque direction.

- La représentation des schémas de routage est générale de manière à couvrir tous les types de dépendances (dépendance de l'entrée, du noeud, etc..., voir 3.3 à cet effet) et tous les modes de codage.

7.2 État de l'art

Lors du fonctionnement d'un réseau d'interconnexion deux phénomènes peuvent se produire: l'interblocage et la boucle infinie. Le premier survient lorsqu'un certain nombre de messages se retrouve dans l'impossibilité de progresser à cause d'une dépendance circulaire entre ces messages.

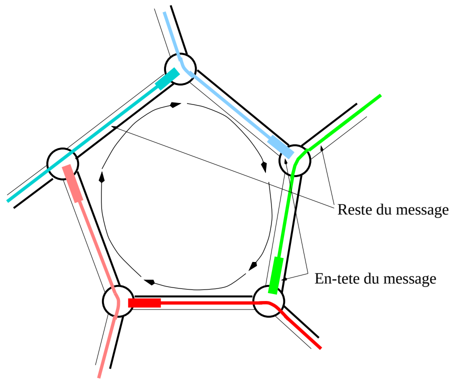


Figure 7.1: Situation de blocage mutuel dans un réseau

Une situation similaire à celle de la figure 7.1 aboutit au gel de la progression des messages impliqués. Le second phénomène arrive lorsqu'un message est indéfiniment routé à travers le réseau sans jamais atteindre sa destination. La cause de ces deux phénomènes vient du schéma de routage lui-même.

Plusieurs études ont été menées afin de résoudre ces deux problèmes. Les méthodes curatives consistent à détecter dynamiquement puis à supprimer l'interblocage par la destruction de certains messages. Ceux-ci devront être reroutés ultérieurement. Ces méthodes imposent de disposer de mécanismes de détection et de destruction de l'interblocage. Celà induit forcément un surcoût matériel et une perte en performances. Les méthodes préventives visent la construction de schémas de routage sains.

7.2.1 Interblocage en routage *store and forward*

En routage *store and forward*, les messages sont transférés en totalité d'un routeur vers un autre à chaque étape du routage. La structure du routeur, pour ce type de technique, implémente généralement un banc de tampons banalisés dans lesquels sont stockés les messages.

L'interblocage arrive lorsque les tampons d'un groupe de routeurs sont pleins et remplis de messages destinés à des routeurs dans le même groupe. La figure 7.2 illustre cette situation par un exemple de trois routeurs.

Les problèmes d'interblocage dans le routage *store and forward* ont été largement étudiés [37] [**?**] [38] [39] [40] [41] [42] [**?**] [43].

Gunther [42] a montre que si le graphe de dépendance des tampons correspondant à l'algorithme de routage ne contient pas de cycle alors il ne peut y avoir interblocage. Le graphe de dépendance des tampons est un graphe orienté dont les sommets représentent les tampons de chaque noeud du réseau. Deux sommets i et j sont reliés par une arête de i vers j si un message peut être routé du tampon associé à i vers celui associé à j.

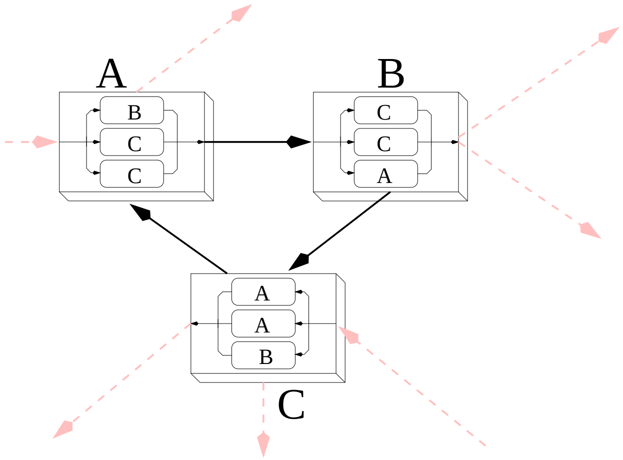


Figure 7.2: Interblocage en routage *store and forward*

Plusieurs algorithmes de routage non-bloquants ont été développés pour les réseaux *store and forward* [42] [44] [38] [39] [41]. Ces algorithmes sont basés sur le principe de structuration du tampon central. Les tampons, dans chaque routeur sont partitionnés en classes. Leur allocation aux messages est restreinte de manière à établir un ordre partiel entre les classes de tampons. Cet ordre permet de réserver explicitement des tampons pour des messages d'une destination donnée de manière à briser toute dépendance circulaire.

Plusieurs techniques de structuration des tampons ont été mises au point. Une solution proposée par Raubold et Haenle [37], s'applique aux topologies régulières pour les algorithmes de routage non-adaptatifs optimaux. Elle consiste à partitionner les tampons en K classes où K est le diamètre du réseau si le routage se fait toujours par le chemin le plus court. L'allocation d'un tampon à un message se fait ensuite selon le nombre de routeurs traversés.

7.2.2 Routage *wormhole*

En routage *wormhole*, un message peut être distribué sur plusieurs noeuds du réseau. A cause de celà les solutions développées pour le routage *store and forward* ne sont plus applicables [8].

Théorie de Dally et Seitz

Dally et Seitz [8] ont constaté que pour la technique de routage *wormhole* ce sont les canaux de communication entre routeurs (liens) et non plus les tampons qui constituent les ressources objets de conflit. Ils en ont conclu que la seule alternative est la restriction du routage lui même. Ils ont montré qu'un algorithme de routage est non-bloquant si le graphe de dépendance des liens ne comporte pas de cycles. Le graphe de dépendance des liens est un graphe orienté dont les sommets sont les canaux de communication entre routeurs. Il existe une dépendance depuis un lien l_i vers un lien l_j s'il existe au moins une adresse pour laquelle l_j est le successeur direct de l_i. On représente cette dépendance dans le graphe de dépendance par une arête du sommet i vers le sommet j.

Cette condition d'absence de cycle, proposée initialement par Dally et Seitz pour les algorithmes de routage déterministes a été étendue aux algorithmes adaptatifs [45] [46] [47] [48] La figure 7.3 présente un exemple d'anneau bidirectionnel. Le

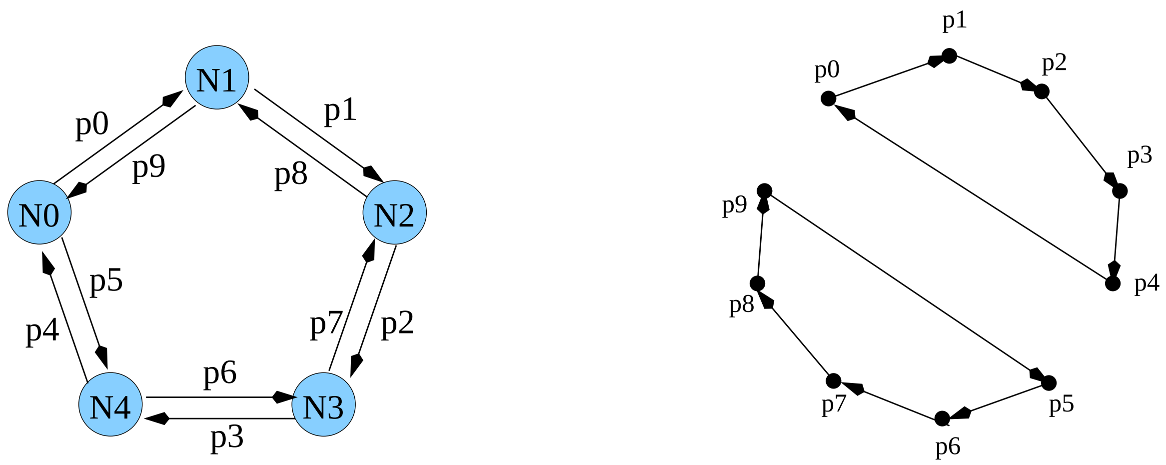


Figure 7.3: Anneau bidirectionnel et son graphe de dépendances des liens

routage, dans cet anneau, utilise le plus court chemin entre deux noeuds et est donné par le tableau suivant:

Destination	Noeud Source				
	N_0	N_1	N_2	N_3	N_4
N_0	-	p_9	p_8, p_9	p_3, p_4	p_4
N_1	p_0	-	p_8	p_7, p_8	p_4, p_0
N_2	p_0, p_1	p_1	-	p_7	p_6, p_7
N_3	p_5, p_6	p_1, p_2	p_2	-	p_6
N_4	p_5	p_9, p_5	p_2, p_3	p_3	-

Le graphe de dépendance présenté en figure 7.3 comporte deux cycles. Pour briser ces deux cycles, il suffit de restreindre le routage. Une manière de briser ces deux cycles est d'interdire la traversée du noeud 0 par des messages dont il n'est pas destinataire. Deux noeuds sont concernés: le noeud 1 quand il envoie un message au noeud 4 et réciproquement. Ainsi, les messages, dans ces deux cas, devront emprunter le chemin en sens inverse plus long. La figure 7.4 montre le nouveau graphe de dépendance dépourvu de cycles.

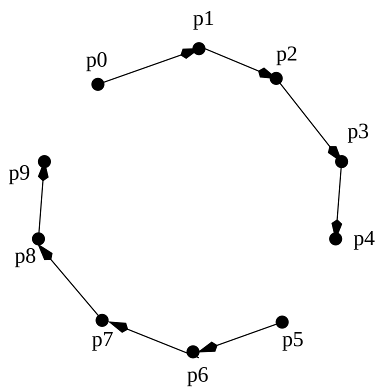


Figure 7.4: Graphe de dépendance après restriction du routage

Un modèle appelé le *turn model* a été dérivé de la théorie de Dally et Seitz par Glass et Ni [15] [49]. Il est destiné à la conception d'algorithmes de routage non-bloquants adaptatifs pour des topologies régulières telle que les grilles. Ce modèle est basé sur l'interdiction de certains changements de direction de façon à éviter d'avoir des cycles. La figure 7.5 montre un exemple d'algorithme développé pour une grille à deux dimensions dans lequel deux changements de direction ont été interdits (en pointillés).

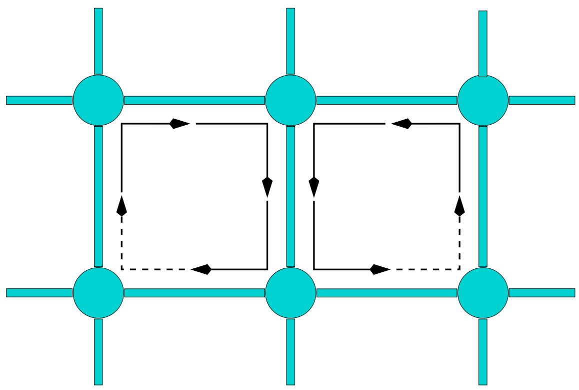


Figure 7.5: Exemple d'application du *turn model*

Pour supprimer les cycles dans le graphe de dépendance des liens, Dally et Seitz [8] ont introduit la notion de canal virtuel. Cette technique permet, de rendre non-bloquant des algorithmes initialement bloquants. En effet, il suffit de multiplexer plusieurs canaux virtuels sur un seul lien physique. Ceci a pour effet d'augmenter le nombre de sommets du graphe de dépendance. Il est alors possible de définir des contraintes sur l'allocation des canaux virtuels de façon à rendre le graphe de dépendances acyclique. De cette manière, l'exemple précédent est résolu simplement par l'ajout de deux canaux virtuels $N_0 \rightarrow N_1$ et $N_0 \rightarrow N_4$. Ainsi, au lieu de restreindre le routage, les messages traversant N_0 emprunteront les nouveaux canaux. La figure 7.6 montre que le graphe de dépendance ne contient plus de cycles.

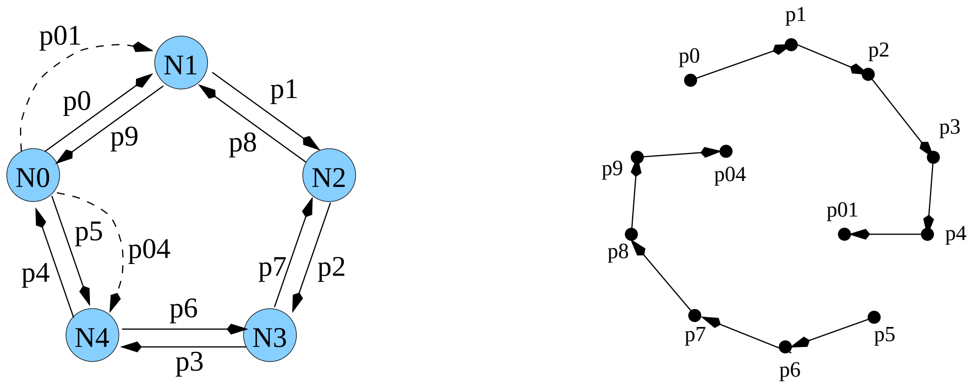


Figure 7.6: Résolution de l'interblocage à l'aide de canaux virtuels

Cette technique a cependant un coût matériel important, puisqu'elle nécessite l'implémentation de plusieurs canaux virtuels indépendants sur le même lien physique.

Première théorie de Duato

En chaque noeud du réseau $\mathcal{R}$, un algorithme de routage adaptatif définit pour chaque destination possible d contenue dans l'entête du message un ensemble de liens de sortie possibles $T(d)$ permettant d'atteindre le noeud destination d. La table de routage $T(d)$ est donc indexée par le numéro du noeud destination d et chaque entrée de la table contient une liste de liens. La réunion des tables $T(d)$ des différents routeurs définit le schéma de routage. Dans l'approche proposée par Duato [50] **?? ??** le graphe de dépendance des liens déduit des tables de routage T peut contenir des cycles (d'autres études ont effectué la même généralisation [51] [52]).

Dans une première approche, Duato [53] [54] définit un sous-réseau connexe $\mathcal{R}_1$ en supprimant certains liens. Cette réduction du nombre de liens disponibles aboutit à un ensemble de tables de routage réduites appelées $T_1(d)$. Ces tables de routage réduites T_1 sont évidemment incluses dans les tables de routage initiales, puisque les liens de sortie permis pour une destination donnée sont un sous-ensemble de l'ensemble défini par le réseau $\mathcal{R}$. Les tables de routage réduites T_1 doivent avoir les deux propriétés suivantes:

- Il y a toujours un chemin entre deux noeuds quelconques du réseau. On dira que le schéma de routage correspondant est **connecté**.
- Le graphe de dépendance des liens engendré par les tables réduites T_1 est sans cycles.

L'idée directrice de la méthode proposée par Duato est que la simple existence de tables de routages réduites T_1 (qui possèdent un graphe de dépendance sans cycles) permet à un message "pris dans un cycle" de trouver un "chemin d'échappement" (même si les tables de routage T effectivement utilisées entraînent des cycles dans le graphe de dépendance).

Cependant, l'absence de cycles dans le graphe de dépendance des liens associé aux tables de routage T_1 ne suffit pas à garantir l'absence d'interblocage. Il faut encore introduire le concept de **dépendance indirecte**, et de **graphe de dépendance étendu**. Ce graphe définit deux types de dépendances au lieu d'une pour le graphe de dépendance de Dally.

- Une dépendance directe existe entre deux liens l_i et l_j appartenant à $\mathcal{R}_1$ si l_j peut être immédiatement emprunté par un message venant de l_i (c'est la dépendance définie par Dally).

- Une dépendance indirecte existe entre deux liens l_i et l_j appartenant à $\mathcal{R}_1$ s'il existe un chemin dans le réseau entre l_i et l_j dont les liens intermédiaires n'appartiennent pas à $\mathcal{R}_1$.

A partir de celà, Duato à énoncé qu'un schéma de routage T est non-bloquant s'il existe un schéma de routage réduit T_1 connecté et ne comportant pas de cycle dans son graphe de dépendance étendu. Cette condition est une condition suffisante pour qu'un schéma de routage soit non-bloquant.

Le fait que $\mathcal{R}_1$ soit connecté et l'absence de cycles dans son graphe de dépendance font qu'un message aura toujours des chemins dans $\mathcal{R}_1$ pour lui permettre d'"échapper" aux dépendances cycliques. Par ailleurs, si un message sort de $\mathcal{R}_1$ en empruntant un lien appartenant à $\mathcal{R} - \mathcal{R}_1$, on est certain qu'il ne provoquera pas d'interblocage. C'est l'absence de cycle dans le graphe de dépendance étendu qui le garantit. En effet, ce message ne peut se retrouver que dans l'un des cas suivants: ou bien il revient dans $\mathcal{R}_1$ (par une dépendance indirecte) ou bien il est routé vers sa destination par un autre chemin dans $\mathcal{R} - \mathcal{R}_1$.

Seconde théorie de Duato

Construire un schéma de routage réduit T_1 en supprimant des liens ne produit qu'une condition suffisante pour qu'un algorithme de routage soit non-bloquant. Duato [9] propose de rendre plus générale cette construction du schéma de routage réduit dans le but d'arriver à une condition nécessaire et suffisante. Dans la première approche de Duato, on associe, dans chaque routeur, à une adresse de destination d, un ensemble de liens de sorties $T(d)$. Dans la seconde approche de Duato, on associe à chaque lien de sortie d'un routeur, un ensemble d'adresses $A(l)$, qui peuvent sortir par ce lien. Le tableau $A(l)$ est indexé par le numéro du lien de sortie, et contient une liste d'adresses de destination. La dimension de ce tableau est égale à la valence du routeur. Les listes d'adresses $A(l)$ ne sont pas nécessairement disjointes si le schéma de routage est adaptatif. La réunion des tables de routage $A(l)$ des différents routeurs définit le schéma de routage. On définit les tables de routage réduites $A_1(l)$ qui sont incluses dans les tables $A(l)$ $(A_1(l) \subseteq A(l) \forall l)$ avec les deux propriétés suivantes:

- Le schéma de routage A_1 est connecté
- Le graphe de dépendance des liens engendré par les tables de routage réduites A_1 est sans cycles.

La principale différence par rapport à la première approche est la suivante: On peut avoir deux destinations d_1 et d_2 telles que: $d_1 \in A_1(l)$ et $d_2 \in A(l) - A_1(l)$. Dans la première approche, un lien était soit disponible soit indisponible pour le schéma de routage réduit T_1. Dans la deuxième approche, la disponibilité d'un lien dans le schéma de routage réduit A_1 dépend de la destination.

L'idée directrice est toujours de fournir des chemins alternatifs permettant aux messages susceptibles de provoquer l'interblocage d'échapper aux cycles de dépendances.

Le nouveau graphe de dépendances étendu comporte 4 types de dépendances: La dépendance directe, la dépendance indirecte, la dépendance **croisée directe** et la dépendance **croisée indirecte**. Considérons deux liens l_a et l_b. On définit comme suit les 4 types de dépendances entres ces deux liens:

- Une dépendance directe existe entre l_a et l_b s'il existe une adresse destination d tel que:
  - $d \in A_1(l_a)$ au niveau du noeud x où l_a est un lien de sortie.
  - $d \in A_1(l_b)$ au niveau du noeud y où l_a est un lien d'entrée et l_b un lien de sortie.
- Une dépendance indirecte existe entre l_a et l_b s'il existe une adresse destination d et un ensemble de liens $l_i, i = 1, k$ tel que:
  - $d \in A_1(l_a)$ au niveau du noeud x où l_a est un lien de sortie.
  - $d \in A(l_1) - A_1(l_1)$ au niveau du noeud x_1 où l_a est un lien d'entrée et l_1 un lien de sortie.
  - $d \in A(l_i) - A_1(l_i)$ pour $i = 2, k$ au niveau du noeud x_i où l_{i-1} est un lien d'entrée et l_i un lien de sortie.
  - $d \in A_1(l_b)$ au niveau du noeud y où l_k est un lien d'entrée et l_b un lien de sortie.
- Une dépendance croisée directe existe entre l_a et l_b s'il existe une adresse destination d tel que:
  - $A_1(l_a)$ est non-vide.
  - $d \in A(l_a) - A_1(l_a)$ au niveau du noeud x où l_a est un lien de sortie.
  - $d \in A_1(l_b)$ au niveau du noeud y où l_a est un lien d'entrée et l_b un lien de sortie.

- Une dépendance croisée indirecte existe entre l_a et l_b s'il existe une adresse destination d et un ensemble de liens $l_i, i = 1, k$ tel que:
  - $A_1(l_a)$ est non-vide.
  - $d \in A(l_a) - A_1(l_a)$ au niveau du noeud x où l_a est un lien de sortie.
  - $d \in A(l_1) - A_1(l_1)$ au niveau du noeud x_1 où l_a est un lien d'entrée et l_1 un lien de sortie.
  - $d \in A(l_i) - A_1(l_i)$ pour $i = 2, k$ au niveau du noeud x_i où l_{i-1} est un lien d'entrée et l_i un lien de sortie.
  - $d \in A_1(l_b)$ au niveau du noeud y où l_k est un lien d'entrée et l_b un lien de sortie.

La notion de dépendance croisée (directe ou indirecte) est similaire à la notion de dépendance simple. La différence est que c'est A_1 qui fournit le lien l_a pour un message d'adresse d dans le cas d'une dépendance simple (directe ou indirecte) entre deux liens l_a et l_b. En revanche, c'est $A - A_1$ qui fournit l_a dans le cas d'une dépendance croisée. On remarque qu'une dépendance croisée, respectivement directe ou indirecte, n'existe entre deux liens l_a et l_b que si aucune dépendance, respectivement directe ou indirecte, n'existe entre ces deux liens.

Pour illustrer l'introduction de ces dépendances considérons l'exemple suivant:

Exemple Soit un anneau unidirectionnel composé de quatre noeuds notés n_i, $i = 0, 1, 2, 3$ et deux liens connectant chaque paire de routeurs adjacents, sauf pour les noeuds n_0 et n_3 qui sont reliés par un seul lien uniquement. Soit B_i, $i = 0, 1, 2, 3$ et H_i, $i = 0, 1, 2$ les liens de sortie des noeuds n_i. L'algorithme de routage est comme suit:

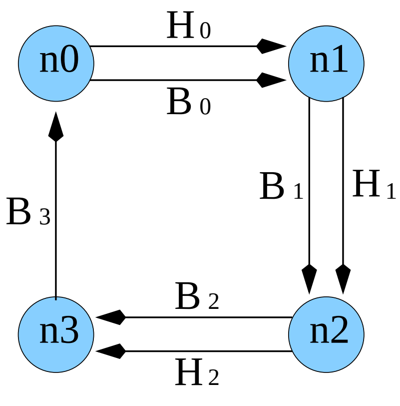


Figure 7.7: Anneau unidirectionnel

- Si le noeud local n_i est égal au noeud destination n_j alors délivrer le message.
- Dans les autres cas, utiliser B_i $\forall j \neq i$ ou H_i $\forall j > i$.

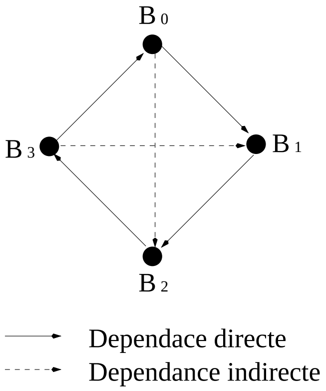


Figure 7.8: Graphes de dépendances étendus obtenus en choisissant les tables de routage réduites à partir de la 1ère théorie

La figure 7.7 montre le réseau. Examinons ce réseau et essayons de construire dessus une configuration d'interblocage. Un message contenu dans le tampon de H_2 est nécessairement destiné à n_3. Donc ses *flits* peuvent avancer. Alors le tampon de H_2 va se vider. De même, si un message est contenu dans le tampon de H_1, il est nécessairement destiné à n_2 ou n_3. Comme le tampon de H_2 est vide, les *flits* vont également avancer et le tampon de H_1 se vide aussi. S'il existe des *flits* contenus dans la queue de H_0, ils sont destinés à n_1, n_2 ou n_3. Même si l'en-tête de ces *flits* est contenue dans le tampon de B_1 ou celui de B_2, ces *flits* peuvent avancer puisque les tampons de H_1 et H_2 sont vides. Le tampon de H_0 se vide alors. Donc, toute configuration d'interblocage ne peut être provoquée que par les liens B_i. Le tampon de B_0 ne peut contenir les *flits* destinés à n_0 car ceux-ci sont localement délivrés au processeur local et ne sont pas réinjectés dans le réseau. Pour toute autre destination (n_1, n_2 ou n_3), ces *flits* peuvent avancer car le tampon de H_1 et le tampon de H_2 sont vides. De cette manière le tampon de B_0 peut être vidé brisant ainsi la dépendance cyclique. Le schéma de routage est donc non-bloquant.

Essayons de prouver que ce schéma de routage est non-bloquant en utilisant la première approche de Duato. Construisons donc le schéma de routage réduit T_1. Ce schéma de routage doit être connecté. Il doit donc obligatoirement contenir B_3, sans ca on ne pourra pas router les messages générés par n_3. B_2 aussi doit faire partie de T_1. Sans celà, n_1 et n_2 ne pourront envoyer de messages à n_0. B_1 doit également faire partie de T_1 car il est nécessaire pour router les messages de n_1 vers n_0. Il n'est donc pas possible de choisir H_1 et H_2 car le choix de l'un d'eux exclut respectivement B_1 et B_2 qui doivent obligatoirement être dans T_1. Nous avons le choix entre B_0 et H_0. Les deux seules combinaisons possibles pour construire T_1 sont donc: (B_0, B_1, B_2, B_3) et (H_0, B_1, B_2, B_3). la figure 7.8 montre les deux graphes de dépendance étendus. pour ces deux configurations différentes du schéma de routage réduit. Tous deux comportent plusieurs cycles. On ne peut donc pas prouver que ce schéma de routage est non-bloquant en utilisant la première théorie de Duato. Cependant, on peut le prouver en

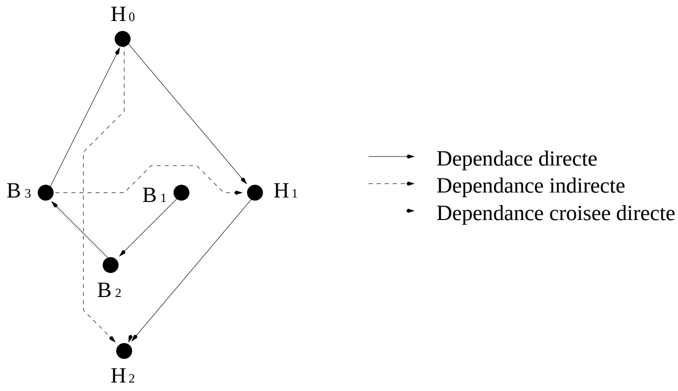


Figure 7.9: Graphe de dépendance étendu obtenu à partir de la 2ème théorie

utilisant la seconde théorie. Considérons un schéma de routage réduit A_1 défini comme suit:

- H_0, H_1, H_2 et B_3 font partie de A_1
- B_0 n'est pas utilisé.
- B_1 et B_2 ne peuvent être utilisés que pour router des messages dont les destinations sont inférieures aux noeud courant.

La figure 7.9 montre le nouveau graphe de dépendance étendu pour A_1. Puisque $A_1(B_1) \bigcap A_1(H_2)$ est vide, alors il n'existe aucune dépendance directe entre B_1 et H_2. Cependant, $n_3 \in A(B_1) - A_1(B_1)$ et $n_3 \in A_1(H_2)$. Donc, une dépendance croisée directe existe entre B_1 et H_2. Comme le graphe présenté à la figure 7.9 est acyclique, le schéma de routage est déclaré non-bloquant.

Théorie de Fleury et Fraigniaud

Fleury et Fraigniaud [10] ont bati une nouvelle théorie à partir des résultats auxquels sont arrivés Dally et Seitz d'une part et Duato d'autre part. Ils ont constaté que la théorie de Duato ne s'appliquait qu'aux schémas de routage dépendant du noeud. Pour étendre le champ d'application de leur théorie, ils ont considéré que le schéma de routage est une fonction à trois variables qui sont: l'en-tête du message, le noeud local et l'état du réseau. Ils expriment par "état du réseau" l'ensemble des informations statiques (algorithme de routage) et dynamiques (disponibilité des ports de sortie). L'idée directrice de la méthode est la même que celle de Duato. Elle autorise l'existence de cycles dans le graphe de dépendances des liens du schéma de routage A. Elle consiste en une restriction des tables de routage dans le but de fournir un ensemble complet de chemins d'échappement aux cycles présents dans le schéma de routage A. On définit les tables de routage réduites A_1 de la même manière que dans la seconde théorie de Duato. Pour garantir que ces tables de routage réduites A_1 fournissent réellement des chemins d'échappement aux cycles, on établit une condition sur ces tables. Cette condition est que ces tables ainsi introduites ne doivent pas provoquer un cycle de type particulier dans un nouveau type de graphe de dépendance, introduit à cet effet, et appelé **graphe de dépendance relatif**. Ce graphe fait appel à un seul type de dépendance appelé **dépendance relative**. Le cycle de type particulier est appelé **cycle atteignable**.

Soit l_i et l_j deux liens de $\mathcal{R}$. Il existe une dépendance relative entre l_i et l_j si:

1 - l_i est un lien de sortie d'un noeud x. $\exists d \in A(l_i)$, tel qu'un message m d'adresse d traverse l_i et traverse éventuellement des liens intermédiaires avant d'aboutir à un noeud y tel que l_j est un lien de sortie de ce noeud y.

2 - Au niveau de ce noeud y, $d \in A_1(l_j)$. Autrement dit l_j est proposé au message m par A_1.

Pour un réseau $\mathcal{R}$, le graphe de dépendance relatif, noté $D(\mathcal{R}, A, A_1)$ est un graphe orienté dont les sommets sont les liens de $\mathcal{R}$. Un arc est établi du sommet associé à l_i vers le sommet associé à l_j s'il existe une dépendance relative de l_i vers l_j.

Soit C_D un cycle de $D(\mathcal{R}, A, A_1)$:

$$C_D = \{l_1 = (x_1, y_1), l_2 = (x_2, y_2), \cdots, l_r = (x_r, y_r), l_{r+1} = l_1, r \geq 1\}$$

Les liens l_i composent le cycle. Chaque lien l_i part du noeud x_i vers le noeud y_i. D'après la définition d'une dépendance relative, ce cycle dans D correspond à un cycle dans $\mathcal{R}$. Ce cycle, qu'on note $C_{\mathcal{R}}$, est de la forme $P_1, P_2, \cdots, P_r$. Chaque élément P_i est un chemin de x_i vers x_{i+1} (éventuellement $P_i = l_i$ si $y_i = x_{i+1}$). Ainsi chaque chemin P_i est composé de:

$$P_i = \{l_{i1} = (z_{i1}, z_{i2}), l_{i2} = (z_{i2}, z_{i3}), \cdots, l_i k_i = (z_{ik_i}, z_{ik_i+1}), k_i \geq 1\}$$

Les noeud $z_{ij}, j = 1, k_i + 1$ constituent les noeuds intermédiaires entre le noeud x_i et le noeud x_{i+1}. Ainsi $l_{i1} = l_i$, $z_{i1} = x_i$, $z_{i2} = y_i$ et $z_{ik_i+1} = x_{i+1}$.

Un cycle C_D est un cycle atteignable dans $D(\mathcal{R}, A, A_1)$, s'il existe un cycle $C_{\mathcal{R}}$ dans $\mathcal{R}$ induit par C_D tel que $C_{\mathcal{R}} = P_0, P_1, \cdots, P_{r-1}$ et r messages $m_i, i = 0, \cdots, r-1$ tel que:

1 - m_i occupe tous les liens de P_i

2 - l'en-tête de m_i, contenant l'adresse destination d_i du message m_i, est contenue dans le tampon de l_{ik_i} dans le noeud x_{i+1}.

3 - $d_i \in A_1(l_{i+1})$.

Un cycle atteignable dans $D(\mathcal{R}, A, A_1)$ correspond à un cycle dans $\mathcal{R}$ ayant une propriété particulière. Cette propriété est que tous les tampons des liens appartenant à ce cycle peuvent être simultanément remplis par des messages à un moment particulier. On fait remarquer que celà ne conduit pas nécessairement vers l'interblocage car il peut exister un chemin d'échappement.

A partir de celà, Fleury et Fraigniaud affirment qu'un schéma de routage défini par les tables de routages A est non-bloquant s'il existe des tables de routage réduites A_1 tel que le graphe de dépendance relatif $D(\mathcal{R}, A, A_1)$ ne contient aucun cycle atteignable.

7.3 Présentation informelle de la théorie

Dans les travaux précédents, Duato [9] et Fleury et Fraigniaud [10] sont arrivés à des conditions nécessaires et suffisantes pour qu'un schéma de routage soit non-bloquant. Leur approche est basée sur l'existence d'un schéma de routage réduit qui a des propriétés particulières. Duato, propose que ce schéma de routage réduit ne provoque pas de cycles dans son graphe de dépendance étendu. Fleury et Fraigniaud affirment que le graphe de dépendance relatif (du schéma de routage initial relativement au schéma de routage réduit) ne doit contenir aucun cycle atteignable.

Bien qu'elles définissent des conditions nécessaires et suffisantes pour qu'un schéma de routage soit non-bloquant, ces théories ne sont pas pratiques. L'exemple que nous avons à la fin de la seconde théorie de Duato illustre le fait qu'il est difficile de trouver le "bon" schéma de routage réduit. Notons que l'exemple en question ne contient que 4 noeuds. Rappelons que notre objectif principal est la vérification automatique de la validité d'une fonction de routage. La construction empirique de ce schéma de routage réduit en essayant toutes les combinaisons est impraticable. Prenons comme exemple un réseau à n noeuds. Supposons que chaque routeur possède p ports et que le schéma de routage dépend du port d'entrée. Supposons également que le schéma de routage fournit en chaque noeud au moins 2 ports de sortie pour une destination donnée d. La construction du schéma de routage réduit au niveau d'un seul routeur requiert $(2^n)^p$ combinaisons. Par ailleurs, une fois le schéma de routage réduit sélectionné, il s'agit de construire son graphe de dépendance. Sa construction met en jeu un grand nombre de tests et de vérifications à cause des dépendances indirectes et relatives qui font intervenir des liens non-adjacents. Ainsi, n'importe quelle paire de liens dans le réseau peut faire l'objet d'une dépendance.

Si la définition d'une condition nécessaire et suffisante constitue un résultat indéniable, on ne peut malheureusement en dériver des méthodes de vérification automatique efficaces.

La théorie que nous présentons a pour point de départ la cause fondamentale de l'occurence de l'interblocage: **le cycle de dépendance**. Cependant, tout cycle ne conduit pas forcément à l'interblocage. Le cycle constitue donc une condition nécessaire et non suffisante de l'occurence de l'interblocage. Fleury et Fraigniaud ont introduit un concept intéressant: **le cycle atteignable**. Un cycle atteignable doit nécessairement être lié au moins à un chemin de routage du réseau. Nous introduisons donc la notion de **chemin de routage vrai** qui est une suite de liens successifs qui route un message ayant pour origine un processeur. De là, nous affirmons qu'un cycle lié à des processeurs par des chemins de routage vrais peut être atteint par des messages générés par ceux-ci. Ces messages risquent de provoquer un interblocage. Un cycle qui n'est lié avec aucun chemin de routage vrai ne peut être la cause d'un interblocage.

La théorie que nous présentons a pour but de caractériser les causes statiques du malfonctionnement d'un algorithme de routage. Ainsi, le cas de messages n'arrivant jamais à leur destination constitue un autre cas de malfonctionnement. Deux causes principales peuvent induire cette situation. D'une part, le schéma de routage peut être inexact. D'autre part, il se peut qu'un message de destination donnée rentre dans un chemin de routage en boucle. Il circulera alors indéfiniment et n'arrivera jamais à destination.

Nous montrerons qu'il existe des relations entre les causes des trois cas de malfonctionnement cités précédemment. Ainsi, nous montrons qu'un schéma de routage est inexact s'il contient au moins un chemin de routage vrai en boucle. Nous montrons également que si un schéma de routage est non-bloquant, il est nécessairement exact. Nous montrons également que la boucle infinie constitue un cas particulier d'une situation d'interblocage.

7.4 Présentation formelle de la théorie

7.4.1 Considérations préliminaires

Un réseau quelconque est modélisé par un graphe simple orienté connexe. Le graphe est formé de noeuds qui sont des routeurs et des processeurs. On désigne par P l'ensemble des processeurs. Chaque processeur est connecté à un routeur par un lien bidirectionnel local. Il peut cependant exister des routeurs non connectés à un processeur. Les processeurs sont identifiés par des étiquettes distinctes prises dans un ensemble d'entiers croissants $E = \{0, ...|P| - 1\}$. E est appelé l'espace d'adressage. Les processeurs échangent des messages à travers le réseau. Une table de routage existe au niveau de chaque routeur. Cette table est indexée par le port d'entrée l par lequel le message est arrivé et par l'adresse destination d, elle retourne une liste de ports de sortie que ce message a la possibilité d'emprunter. Cette liste est notée $W(l, d)$. Autrement dit, la décision de routage dépend non seulement de l'adresse de destination mais également du port d'entrée. Ce cas est plus général que celui envisagé par Duato. La réunion de toutes les tables $W(l, d)$ de tous les routeurs définit le schéma de routage.

Certaines hypothèses et conventions liées au déroulement de l'opération de routage sont requises afin d'asseoir toute réflexion ultérieure sur des bases ne pouvant conduire vers des contradictions ou des cas particuliers:

- Le routage utilise la technique *wormhole*.
- Au niveau de chaque routeur, chaque port d'entrée possède un tampon.
- Les messages sont de longueur finie quelconque. Un message de **taille unitaire** occupe un seul tampon d'un routeur.
- **Livraison:** Chaque processeur consomme tout message lui arrivant à travers le lien **terminal**.
- **Injection** Chaque processeur peut injecter un message vers tout autre processeur. Il le fait à travers le lien d'**injection** vers le réseau.

- **Convention 1:** Au niveau de chaque routeur auquel est connecté un processeur local d'adresse d, la liste $W(l,d)$ ne doit contenir que le port terminal l_d (port d'acheminement vers le processeur local) pour tout lien l. Pour toute autre adresse $d' \neq d$, $W(l,d')$ ne doit pas contenir l_d.

- **Convention 2:** Pour chaque port d'entrée, si la liste $W(l,d)$ correspondant à une adresse d est vide, alors le message est détruit. Nous admettons, par convention, que détruire un message est équivalent à le router à travers un chemin en boucle. Cette convention sera utilisée dans la démonstration 1.

- **Convention 3:** Nous supposons que l'algorithme de choix d'un port de sortie dans l'ensemble $W(l,d)$ est quelconque.

7.4.2 Exactitude d'un schéma de routage

Tout schéma de routage établit une relation entre des ports d'entrée et des ports de sortie d'un même routeur. La topologie de son coté établit une relation physique entre

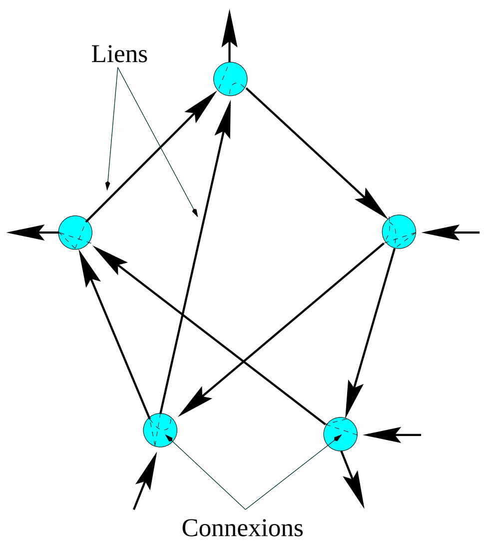


Figure 7.10: Représentation du réseau sous forme d'un graphe orienté

des ports de sortie et des ports d'entrée appartenant à des routeurs ou des processeurs différents. Cette relation est représentée par un lien d'un port de sortie d'un routeur vers un port d'entrée d'un autre routeur.

Un lien est un liaison orientée qui a deux extrémités. L'extrémité d'origine est toujours un port de sortie et l'extrémité terminale est toujours un port d'entrée. Nous utilisons ainsi une seule entité pour représenter deux ports appartenant à des routeurs différents. Nous appelons connexion la relation interne entre un port de sortie et un port d'entrée d'un même routeur (voir figure 7.10).

Définition 1 *Soit l et l' deux liens respectivement en entrée et en sortie d'un noeud s dans un réseau quelconque. l et l' est une connexion; $l \prec l'$, si $\exists b \in E : l' \in W(l,b)$. Si nous notons ll' l'ensemble des destinations qui peuvent établir une connexion entre l et l':*

$$ll' = \{b : b \in E \; and \; l' \in W(l,b)\}$$

Une connexion $l \prec l'$ est induite par une adresse a; $l \overset{a}{\prec} l'$, si $a \in ll'$.

Définition 2 *Le graphe de dépendance D des liens est défini de la façon suivante:*

- *Les sommets du graphe sont les liens l.*
- *Il y a un arc orienté de l vers l' si $ll' \neq \emptyset$.*

Définition 3 *Soit $R = \{l_1, ..., l_n\}$ une séquence de liens successifs dans un réseau G, R est un chemin de routage si et seulement si $\mathcal{R} = \bigcap_{i=1}^{n-1} l_i l_{i+1}$ est non vide. l_1 est noté $First(R)$ et l_n est noté $Last(R)$ Un chemin de routage R route un message d'adresse a (ou, simplement, route a) si $a \in \mathcal{R}$. (On remarque qu'un chemin de routage définit un chemin dans le graphe de dépendance. En revanche, un chemin quelconque dans le graphe de dépendance ne constitue pas nécessairement un chemin de routage).*

- *R est un chemin de routage vrai si l_1 est un lien d'injection.*
- *R est un chemin de routage en boucle si $l_1 = l_n$.*
- *Un chemin de routage vrai est en boucle s'il contient une boucle.*

Définition 4 *Un schéma de routage est exact si tout message de longueur unitaire et seul dans le réseau atteint sa destination en traversant un nombre fini de liens.*

Cette définition signifie qu'un message envoyé par n'importe quel noeud source dans un réseau donné atteint sa destination en un nombre fini d'étapes.

Lemme 1 *Un schéma de routage d'un réseau d'interconnexion est exact si et seulement si il ne contient pas un chemin de routage vrai en boucle.*

Démonstration Démontrons l'implication dans un sens puis celle dans l'autre sens. Soit un schéma de routage S.

- Si un chemin de routage vrai en boucle existe, alors il existe une adresses a et un processeur source x tel qu'un message de longueur unitaire et d'adresse a envoyé par x peut atteindre le chemin de routage en boucle et y tourner indéfiniment. Ainsi, il n'est jamais acheminé à sa destination finale.
- Supposons qu'un schéma de routage est inexact. Alors, il existe au moins un noeud x tel qu'un message de taille unitaire et de destination d envoyé seul dans le réseau du noeud x n'atteint jamais sa destination. Puisque le message

est seul dans le réseau, il ne peut être pris dans un interblocage avec d'autres messages. Puisqu'il est de longueur unitaire, il ne peut pas être bloqué par lui-même. D'après la convention de livraison, aucun message ne se bloque au niveau du lien terminal. D'après la convention 1, aucun lien terminal autre que celui du processeur associé n'a la même adresse que celui-ci. Donc aucun message ne se trompe de destination. Les deux causes de non-acheminement sont les suivants:

1 **Absence de lien de sortie associé a d au niveau d'un routeur:** Si au niveau d'un routeur $W(l,d)$ est vide, alors, d'après la convention 2, le message est détruit. Cette destruction équivaut par convention à un routage sur un chemin en boucle.

2 **Absence de lien terminal sur le chemin:** Au niveau de tous les routeurs traversés, il existe toujours au moins un lien associé à d, mais ce lien n'est jamais un lien terminal. Désignons par $\{l_1, ..., l_n\}$ cette séquence de liens traversés en partant de x. Cette séquence est un chemin de routage vrai en boucle.

7.4.3 Configurations d'interblocage

Tout interblocage est causé par un cycle de dépendance entre certains liens du réseau. Cependant, tout cycle ne génère pas forcément une situation d'interblocage. Nous avons donc classé les cycles en deux types.

Configuration faiblement bloquante

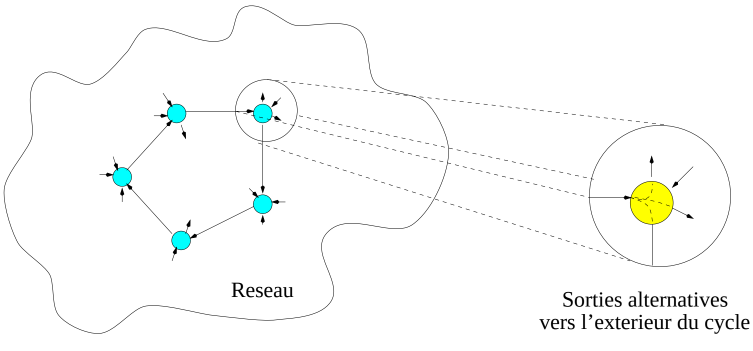


Figure 7.11: Configuration d'interblocage faible

Définition 5 *Soit $\mathcal{C}_w = \{l_0...l_{k-1}\}$ une séquence de liens de successifs. C_w est une configuration faiblement bloquante si et seulement si $l_i \prec l_{(i+1)[k]}$ pour $i = 0, \cdots, k-1$, et chacun des liens concernés appartient à un vrai chemin de routage.*

Deux remarques importantes:

- Il n'existe aucune restriction sur les adresses qui induisent les connexions entre les différents liens composant le cycle. Il se peut qu'un lien ait pour chaque adresse qui induit sa connexion avec son successeur dans le cycle, plusieurs liens de sortie possibles (adaptativité). Ces autres liens peuvent permettre aux messages pris dans le cycle d'échapper à l'interblocage.

- Pour que le cycle puisse produire un interblocage, il faut que les messages puissent y pénétrer. Pour celà, il est nécessaire que chacun des liens du cycle appartienne à un chemin de routage vrai. Un lien qu'aucun message ne traverse (du fait qu'il n'appartienne à aucun chemin de routage vrai) brise le cycle. Il se peut également qu'un lien fasse partie d'un chemin de routage vrai, mais que l'adresse induisant sa connexion avec son successeur dans le cycle ne soit pas une destination de ce chemin. Ce cas a le même effet que le précédent: le cycle est rompu et aucun interblocage ne peut se produire.

Ainsi, par définition, toute configuration faiblement bloquante ne conduit pas nécessairement vers un interblocage.

Nous retrouvons une définition possible d'un chemin de routage en boucle.

Corollaire 1 *Une configuration faiblement bloquante $\mathcal{C}_w = \{l_0 ... l_{k-1}\}$ tel que:*

$$\exists a, l_i \overset{a}{\prec} l_{i+1[k]} \forall i = 0, k-1$$

est un chemin de routage R en boucle qui route les messages d'adresse a.
Réciproquement, tout chemin de routage en boucle est une configuration faiblement bloquante.

Configuration fortement bloquante

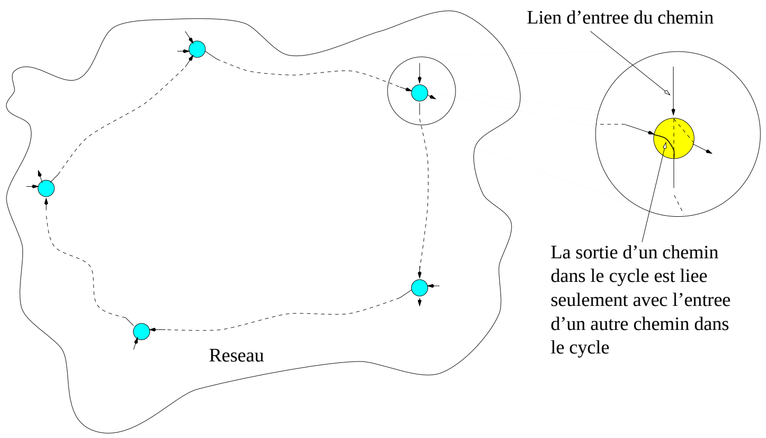


Figure 7.12: Configuration d'interblocage fort

Considérons un ensemble de chemins de routage disjoints R_i (c'est à dire ne possédant aucun lien en commun 2 à 2) tel que:

- Le premier lien *First(R_i)* de chaque chemin R_i n'est pas un lien d'injection.
- Le dernier lien *Last(R_i)* de chaque chemin R_i n'est pas un lien terminal.
- Pour chaque chemin de routage R_i, il existe au moins une adresse destination d_i qui a les propriétés suivantes:
  - R_i constitue un chemin de routage pour tout message d'adresse d_i.
  - Les liens successeurs de *Last(R_i)* induits par cette adresse d_i sont obligatoirement les premiers liens (*First(R_j)*, *First(R_k)*,$\cdots$) d'autres chemins de routage appartenant à ce même ensemble de chemins.

Les liens de cet ensemble de chemins constituent un cycle. Cependant, les conditions supplémentaires que nous avons défini ci-dessus font qu'il a des propriétés particulières.

Définition 6 *Soit $\mathcal{C}_s$ un ensemble de chemins de routage disjoints tel que $Last(R_i)$, pour tout R_i dans $\mathcal{C}_s$, n'est pas un lien d'acheminement. $\mathcal{C}_s$ est une configuration fortement bloquante si et seulement si pour chaque chemin de routage $R_i \in \mathcal{C}_s$, il existe une adresse $d_i \in \mathcal{R}$, telle que si $Last(R_i) \overset{d_i}{\prec} l$, alors l est le premier lien d'un chemin de routage R' qui appartient à $\mathcal{C}_s$.*

Une configuration fortement bloquante est vraie si chacun des chemins impliqués constitue une sous-séquence d'un chemin de routage vrai.

Une configuration faiblement bloquante définit le cycle d'une manière générale. Une configuration fortement bloquante réunit toutes les conditions pour qu'un interblocage se produit:

- Un chemin de routage R_i faisant partie d'une configuration fortement bloquante route tout message d'adresse d_i. Donc, chaque connexion d'un lien de ce chemin avec son successeur est induite par d_i. Si R_i est une sous-séquence d'un chemin de routage vrai, alors il est possible qu'un message d'adresse d_i occupe simultanément tous les liens de R_i.
- Tout successeur du dernier lien de R_i pour l'adresse d_i est obligatoirement le premier lien d'un autre chemin de routage R_j de la configuration fortement bloquante. Celà supprime toute possibilité aux messages de trouver des chemins permettant d'échapper au cycle.

C'est pour ces raisons que nous avons qualifiée cette configuration de configuration **fortement** bloquante.

Définition 7 *Un schéma de routage est fortement libre de tout interblocage si et seulement si il ne contient aucune configuration faiblement bloquante.*

Lemme 2 *Chaque configuration fortement bloquante contient une configuration faiblement bloquante.*

Démonstration Soit $\mathcal{C}_s$, une configuration fortement bloquante. Par définition:

- Pour tout chemin de routage $R_j \in \mathcal{C}_s$,

$$\exists a_j, a_j \in R_j \text{ tel que } Last(R_j) \overset{a_j}{\prec} First(R_{j+1[|\mathcal{C}_s|]}), j = 0, |\mathcal{C}_s| - 1$$

- Chaque chemin R_j est de la forme: $l_{ji} \overset{a_j}{\prec} l_{j(i+1)}$ où l'indice i repère les liens dans le chemin R_j.

On en déduit que la configuration $\mathcal{C}_u$ formée par les liens successifs suivants:

$$l_{00} \overset{a_0}{\prec} l_{01} \overset{a_0}{\prec} \cdots \prec l_{0k_0} \overset{a_0}{\prec} l_{10} \overset{a_1}{\prec} \cdots \prec l_{1k_1} \overset{a_1}{\prec} \cdots \overset{a_{N-1}}{\prec} l_{N-1k_{N-1}} \overset{a_{N-1}}{\prec} l_{00}$$

$\mathcal{C}_u$ est, par définition, une configuration faiblement bloquante.

Dans ce qui suit, nous allons examiner la relation qui existe entre une configuration faiblement bloquante, une configuration fortement bloquante et l'exactitude d'un schéma de routage.

Corollaire 2 *Un schéma de routage qui n'est pas exact est faiblement bloquant. Un schéma de routage fortement libre de tout interblocage est nécessairement exact.*

Démonstration Un schéma de routage non-exact contient nécessairement un chemin de routage en boucle. Puisque celui-ci est un cas particulier d'une configuration faiblement bloquante, alors le schéma de routage est faiblement bloquant.

Un schéma de routage fortement libre de tout interblocage ne contient aucune configuration faiblement bloquante. Ainsi, ce schéma de routage ne contient donc aucun chemin de routage en boucle car celui-ci n'est qu'un cas particulier d'une configuration faiblement bloquante. Il ne contient donc aucun chemin de routage vrai en boucle. Ce schéma de routage est donc exact.

Définition 8 *Un schéma de routage est faiblement libre de tout interblocage ssi il ne contient aucune configuration fortement bloquante.*

Corollaire 3 *Si un schéma de routage est fortement libre de tout interblocage alors il est également faiblement libre de tout interblocage.*

Démonstration Un schéma de routage fortement libre de tout interblocage n'admet aucune configuration faiblement bloquante. D'après le lemme 2, chaque configuration fortement bloquante contient une configuration faiblement bloquante. Alors, il ne peut contenir aucune configuration forte d'interblocage. Il est donc faiblement libre de tout interblocage.

Lemme 3 *Pour un schéma de routage déterministe la définition d'une configuration faiblement bloquante est équivalente à celle d'une configuration fortement bloquante.*

Démonstration Pour un schéma de routage déterministe, la définition d'une configuration d'interblocage faible devient: Si l_i et $l_{i+1} \in \mathcal{C}_w$, $l_i \overset{a}{\prec} l_{i+1}$ et $l_i \in R_j$ et $l_{i+1} \in R_k$, R_j et R_k deux chemins de routage quelconques, alors $a \in R_j$ nécessairement, car le routage est déterministe. Ceci est équivalent à une configuration fortement bloquante.

Ce lemme reproduit exactement le théorème de Dally et Seitz [8] concernant les schémas de routage déterministes: tout cycle détecté et atteignable provoque nécessairement un interblocage.

7.4.4 Conclusion

La théorie que nous venons de présenter se base sur l'étude des cycles de dépendance dans le schéma de routage. Seuls les cycles appartenant à des chemins de routage vrais sont considérés. Partant d'un cycle de dépendance défini de manière générale, on définit deux types de configurations d'interblocage. Ces deux configurations regroupent l'ensemble des cas qui causent un interblocage dans un schéma de routage. Le premier type de configuration d'interblocage est défini en ne considérant que les dépendances entre les liens composant le cycle. On ne considère que les dépendance entre un lien et son successeur dans le cycle. Cette manière de définir cette configuration permet de supposer que l'adaptativité du schéma de routage fournit éventuellement des chemins alternatifs. Cette configuration, que nous avons qualifiée de faiblement bloquante ne conduit donc pas nécessairement vers un interblocage. On définit une configuration fortement bloquante en établissant des conditions sur les liens qui la composent pour qu'elle conduise effectivement vers un interblocage. De là on définit trois types de schémas de routage:

- Schéma de routage sain: n'admet aucune configuration faiblement bloquante vraie.
- Schéma de routage faiblement bloquant: admet au moins une configuration faiblement bloquante mais n'admet aucune configuration fortement bloquante.
- Schéma de routage bloquant: admet au moins une configuration fortement bloquante.

7.5 Principe de la vérification automatique

Nous cherchons à vérifier automatiquement la validité (exactitude et absence d'interblocage) d'un schéma de routage. Nous proposons une méthode qui s'appuie sur la théorie que nous avons présentée.

Voici les cas dans lesquels un schéma de routage peut se trouver:

- Un schéma de routage est non-exact. Il n'est donc pas valide.

- Un schéma de routage est exact. Dans ce cas, la présence ou l'absence en son sein d'une configuration bloquante détermine sa validité.
  - Ce schéma de routage est non-bloquant. Donc il est valide.
  - Ce schéma de routage est faiblement bloquant. Le schéma de routage peut être valide comme il peut ne pas l'être. Celà dépend de la présence ou non au niveau des liens de la configuration d'interblocage de chemins d'échappement.
  - Ce schéma de routage est fortement bloquant, il n'est donc pas valide.

On commence par vérifier l'exactitude du schéma de routage. Si celui-ci est exact, on continue en vérifiant l'interblocage.

7.5.1 Déroulement de la vérification automatique

Pour effectuer la vérification on construit le graphe de dépendance des liens à partir du schéma de routage et de la topologie. Pendant cette construction, on vérifie que:

- Au niveau de chaque routeur connecté à un lien d'injection, le schéma de routage fournit bien un port de sortie pour chaque adresse dans le réseau.
- Seul le lien terminal possède l'adresse du processeur auquel il est connecté.

Ensuite, on détermine tous les chemins de routage entre chaque paire de processeurs pris 2 à 2. Lors de cette construction des chemins de routage, on vérifie l'exactitude du schéma de routage en vérifiant que chaque chemin de routage ne contient pas de boucle. Si le schéma de routage est exact, nous pouvons poursuivre la vérification en recherchant les cycles de dépendance.

Dans un premier temps, il faut détecter tous les cycles. L'algorithme de détection de cycles se base également sur un parcours récursif du graphe. cependant, cette fois on ne considère que les informations de connexion seules sans se préoccuper des adresses qui les induisent. A chaque fois qu'un cycle est détecté, on dresse la liste ordonnée des indices des liens composant ce cycle.

A ce niveau, on effectue deux vérifications:

- On vérifie d'abord si le cycle est atteignable en déterminant si chaque lien du cycle appartient à un chemin de routage vrai.
- On vérifie ensuite si l'ensemble d'adresses par lequel un lien est connecté à son successeur dans le cycle appartient aussi au chemin de routage contenant ce lien. On opère cette vérification pour tous les liens du cycle.

Si l'une de ces vérification aboutit à un résultat négatif (on trouve un lien non-conforme aux considérations suscitées) alors le cycle n'est pas considéré. Autrement. il est retenu comme étant une configuration d'interblocage. Dans ce cas, on cherche à déterminer si c'est une configuration fortement bloquante en examinant les ensembles d'adresses induisant les connexions entre les différents chemins de routage auxquels appartiennent les liens. Si ce cycle est une configuration fortement bloquante vraie, la vérification s'arrête et le schéma de routage est déclaré non-valide. Dans le cas contraire, on continue à rechercher d'autres cycles.

A la fin de cette étape de détection des cycles, on se trouve nécessairement dans l'un des cas suivants:

- Aucun cycle n'a été trouvé auquel cas le schéma de routage est valide.
- Aucun cycle trouvé ne peut constituer une configuration d'interblocage. Dans ce cas également, le schéma de routage est déclaré valide.
- Il existe une ou plusieurs configurations faiblement bloquantes. Dans ce cas, on essaie de trouver pour chacune de ces configurations des chemins d'échappement qui permettent aux messages d'échapper aux cycles. On effectue celà en considérant pour chaque connexion tous les liens alternatifs offerts pour un ensemble d'adresses donné. En empruntant ces liens, on essaie de trouver des chemins composés de liens n'appartenant à aucun cycle jusqu'aux liens terminaux. Si à une étape donnée du parcours on s'aperçoit que tous les successeurs d'un lien font partie d'un cycle, la progression s'arrête et on considère qu'aucun chemin alternatif n'a été trouvé pour ce lien. A la fin de cette opération, si on trouve des chemins alternatifs pour tous les liens de tous les cycles, le schéma de routage est déclaré valide. Dans le cas contraire, on signale celà sans pouvoir se prononcer quant à l'occurence véritable de l'interblocage.

Ainsi, la vérification s'arrête en fournissant dans la majorité des cas une réponse claire sur la validité du schéma de routage. Notons que cette méthode s'appuie uniquement sur les données statiques que sont la topologie et le schéma de routage.

7.5.2 Implémentation

L'outil implémentant cette méthode de vérification accepte en entrée une description du réseau d'interconnexion qui comprend la topologie et le schéma de routage. Cet outil effectue d'abord une transformation de la description d'entrée pour obtenir le graphe de dépendance des liens. Ce graphe ne fait intervenir que les dépendances directes c'est à dire les connexions $l \prec l'$. Pour arriver à celà, l'outil dresse une liste de tous les ports d'entrée et une liste de tous les ports de sortie. A partir de ces deux listes, il construit la liste des liens. On spécifie ensuite pour chaque lien tous ses successeurs en précisant les adresses destinations qui induisent chaque connexion. La structure de donnée représentant le lien est composée de (voir figure 7.13):

- Un champs identificateur.

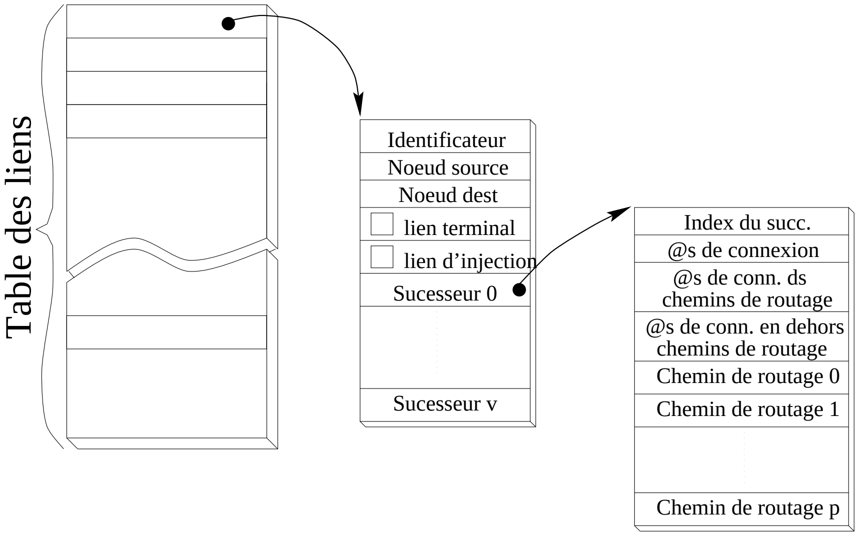


Figure 7.13: Structure de donnée

- Deux champs identifiant respectivement les noeuds source et destination du lien (topologie).

- Deux indicateurs (drapeaux) indiquant si ce lien est un lien d'injection ou un lien terminal.

- Une table des successeurs de ce lien. La dimension de cette table correspond à la valence des routeurs dans le réseau. Chaque élément de cette table pointe vers une structure de données composée de:

  – Index du successeur dans la liste globale des liens.

  – Pointeur vers l'ensemble d'adresses induisant cette connexion. Il s'agit en fait des tables $W(l, d)$.

  – Pointeur vers le sous-ensemble d'adresses appartenant à un chemin de routage vrai.

  – Pointeur vers le sous-ensemble d'adresses n'appartenant à aucun chemin de routage.

  – Liste d'indexs de tous les chemins de routage auxquels appartient ce lien.

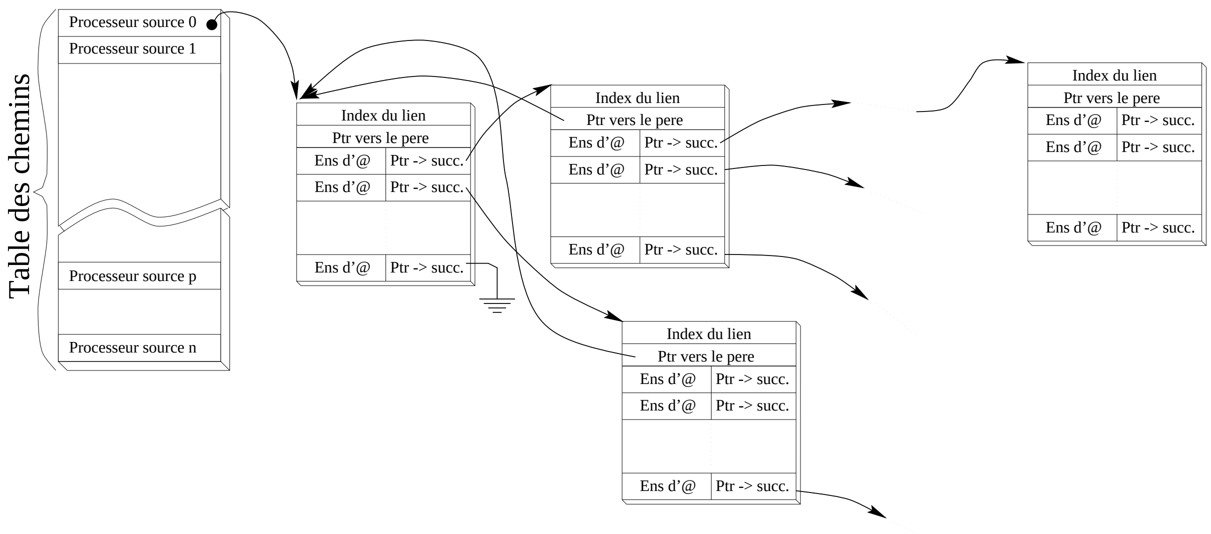


Figure 7.14: Structure de données représentant les chemins de routage

La construction des chemins de routage se fait en parcourant le réseau à partir de chaque processeur. Le but est de construire tous les chemins possibles reliant tout processeur à tout autre processeur. Les chemins construits sont représentés par la structure de données présentée en figure 7.14. Cette structure est une série de listes chainées. Chacune de ces listes est associée à un processeur. Cette liste chainée, qui est un graphe orienté acyclique (DAG) (voir figure 7.15), représente tous les chemins de routage possibles issus de ce processeur. Un élément de cette liste représente un lien du chemin. Il se décompose en:

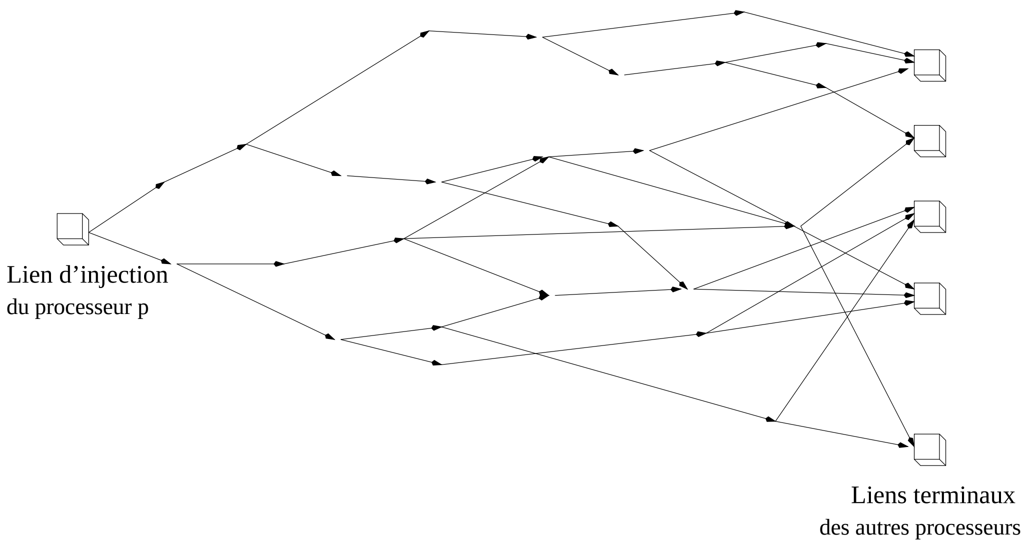


Figure 7.15: Ensemble des chemins de routage issus d'un processeur *p*

- un index du lien dans la table des liens représentée par la structure de données principale.
- un pointeur vers l'élément précédent (chainage inverse).
- une sous-table qui contient les informations des liens successeurs de ce lien dans le chemin. Chaque entrée de cette sous-table est composée de:
  - L'ensemble d'adresses des processeurs atteignables à travers ce lien successeur.
  - Le pointeur vers l'élément de la liste chainée représentant ce lien successeur.

L'association d'un ensemble d'adresses à un chemin permet d'éviter les redondances dans la représentation des liens. En effet, si chaque chemin entre 2 processeurs était représenté individuellement, un même lien faisant partie de plusieurs chemins sera représenté plusieurs fois.

```
Elt_Chemin *Construire_Chemin( l, E)
/* l represente un lien */
/* E represente l'ensemble courant d'@s */

DEBUT
     Elt = Allouer(Elt_Chemin) /* Alloue un element
                                  de type Elt_Chemin */
     i = 0
     POUR tout lien li successeur de l
       FAIRE
            SI li lien terminal
              ALORS
                     Elt.Adresse[i] = pi    /* @ du processeur attache */
                     Elt.Suivant[i] = NIL
              SINON
                     SI   E ∩ A(li) # Ø
                       ALORS
                             Elt.Adresse[i] =   E ∩ A(li)
                             Elt.Suivant[i] = Construire_Chemin( l1,  E ∩ A(li))
                     FINSI
            FINSI
            i = i + 1
       FINFAIRE
     Retourner(Elt)
FIN
```

Figure 7.16: Algorithme de construction de chemins de routage

L'algorithme de construction est un algorithme récursif. On applique cet algorithme successivement à chacun des processeurs du réseau. Cet algorithme est décrit en figure 7.16. A partir d'un processeur d'adresse p, cet algorithme effectue un parcours du graphe en recherchant les liens terminaux de tous les autres processeurs. Il construit progressivement la liste représentant l'ensemble des chemins en y ajoutant chaque lien atteint. L'ensemble d'adresses associé est représenté par l'intersection de l'ensemble d'adresses associées courant et l'ensemble d'adresses associées à ce lien dans le graphe initial. L'ensemble d'adresses courant est l'ensemble d'adresses associées au lien précédent du lien atteint. Cet ensemble représente l'ensemble des processeurs pouvant être atteints à travers ce lien par des messages **issus** du processeur p.

La détection des cycle se fait en utilisant un algorithme similaire au précédent. Cependant, dans ce cas, les adresses qui induisent les connexions entre deux liens ne sont pas considérées. Seule l'existence d'une connexion est effective. On considère cependant qu'un lien n'est jamais connecté à son double dans la direction opposée (pour les liens bidirectionnels). L'algorithme est présenté en figure 7.17.

Cet algorithme permet de détecter les cycles en utilisant une pile. Cette pile contient à tout moment l'ensemble des liens déjà parcourus. L'existence, dans la pile, du lien suivant du dernier lien empilé signifie qu'il existe un cycle. Ce cycle est composé par les liens de la pile situés entre la première occurence de ce lien existant dans la pile et le dernier lien de la pile.

```
PROCEDURE Detecter_Cycle(l)
/* l represente un lien */

DEBUT
      Empiler(l)
      POUR chaque li connecte a l
        FAIRE
              SI li existe dans la pile
                ALORS
                      /* Cycle detecte */
                      Nouveau_Cycle = liens compris entre li et l
                SINON
                      SI   li n'est pas un lien terminal
                        ALORS
                              Detecter_Cycle(li)
                      FINSI
              FINSI

        FINFAIRE
      Depiler(l)
FIN
```

Figure 7.17: Algorithme de détection des cycles

Après la détection du cycle, on effectue son analyse pour déterminer si c'est un cycle atteignable (appartient à des chemins de routage) et déterminer son type (configuration faiblement ou fortement bloquante). On s'assure d'abord que chaque lien du cycle appartient à un chemin de routage. Ensuite, on détecte tous les chemins de routage auxquels appartient au moins un lien du cycle. Avec ce sous-ensemble de chemins, on essaie de construire une configuration fortement bloquante: Chaque lien peut appartenir à plusieurs chemins de routage en même temps. En choisissant à chaque fois un des chemins pour chaque lien, on effectue toutes les combinaisons possibles dans le but de trouver une combinaison qui soit conforme aux caractéristiques d'une configuration fortement bloquante.

7.6 Conclusion

Dans ce chapitre, une théorie complète de la vérification de la validité d'un schéma de routage a été développée. Cette théorie repose sur la définition de deux types de configurations qu'un schéma de routage peut contenir et qui peuvent mener vers une situation d'interblocage et/ou de boucle infinie. Nous avons montré qu'il existe une relation entre l'interblocage, la boucle infinie et l'exactitude.

Sur la base de cette théorie, nous avons proposé une méthode de vérification algorithmique. Cette méthode reprend la notion de connexion qui fut la base sur laquelle nous avons construit notre théorie de vérification. Nous avons présenté un algorithme de vérification opérant sur un schéma de routage représenté selon ce modèle et utilisant des méthodes de parcours simples. Notre méthode est capable de détecter tous les cas d'interblocage et de boucle infinie. Cependant, il reste le cas d'une configuration faiblement bloquante qui nécessite une étude théorique plus approfondie. Cette étude a pour but de fournir les bases théorique de recherche des chemins alternatifs dans de tels cas. Nous n'avons pas pu réaliser l'outil implémentant cette méthode par manque de temps. Un travail complémentaire est nécessaire pour réaliser l'outil logiciel.

CHAPITRE 8

CONCLUSION

Tout au long de cette thèse, notre objectif était de fournir des outils qui permettent de construire efficacement des réseaux d'interconnexion pour des machines parallèles dont les noeuds sont des stations de travail en général et pour la machine Multi-PC en particulier. Nous avons traité trois point essentiels:

Nous avons commencé par déterminer les topologies et schémas de routage optimaux pour la machine Multi-PC. Dans ce but, nous avons conçu et développé un environnement complet de simulation dédié à l'évaluation des performances de réseaux d'interconnexion: MILE. Nous avons défini, pour différentes topologies régulières une méthode de construction des tables de routage. Utilisant MILE, nous avons effectué un grand nombre de simulations sur ces topologies en faisant varier la charge appliquée au réseau et la longueur des messages. Les résultats de simulation montrent la domination de l'hypercube qui apparaît comme le réseau présentant globalement les meilleures performances. De plus, son comportement est régulier pour toutes les tailles simulées. D'autres réseaux ont obtenu des résultats satisfaisants et offrent plus de possibilités d'extension que l'hypercube. Nous pensons notamment à la grille composée et au LCAN binaire.

L'analyse du gain que procure l'adaptativité montre que ce gain est négligeable pour la bande passante et se situe entre 5 et 20 % pour la latence. C'est malheureusement la bande passante qui est le paramètre le plus significatif actuellement car même un gain de 100 % sur la latence n'aura que peu d'effet sur les performances. Toutefois, l'adaptativité procure des avantages certains pour les applications où la latence est un facteur critique. Avant de prendre une quelconque décision quant à son utilisation, il est nécessaire d'évaluer le surcoût lié au fait que l'ordre d'arrivée des messages n'est plus garanti.

Nous avons étudié les performances de la structure de mémorisation du routeur RCUBE à travers les résultats de deux crossbars réalisés différemment utilisant des RCUBEs. Les résultats de cette étude ont montré l'intérêt d'améliorer le routeur RCUBE en y ajoutant une structure de mémorisation centralisée. Nous avons défini l'architecture de cette structure en connectant une série de FIFOs au crossbar de manière a former une structure de rebouclage. Nous avons développé et utilisé un modèle de routeur générique pour expérimenter diverses tailles de FIFOs et diverses stratégies d'allocation. Le résultat obtenu est que la mémorisation en sortie est plus performante que la mémorisation en entrée, confirmant les affirmations de [33].

Nous avons effectué une amélioration de l'algorithme d'étiquetage automatique de topologies quelconques de Van Leeuwen-Tan. Nous avons modifié cet algorithme dans le but de configurer automatiquement des réseaux utilisant une technique de routage par intervalle où il n'y a pas nécessairement un processeur pour chaque routeur. La particularité de notre algorithme est qu'il produit des adresses consécutives facilitant la détection des erreurs de transmission lorsqu'elles concernent les en-têtes des messages. Nous avons réalisé et testé un outil logiciel qui implémente cet algorithme.

Nous avons développé une théorie complète de la vérification de la validité d'un schéma de routage. Cette théorie repose sur la définition de deux types de configurations qu'un schéma de routage peut contenir et qui peuvent mener vers une situation d'interblocage et/ou de boucle infinie.

Nous avons proposé une méthode basée sur cette théorie pour vérifier automatiquement la validité d'un schéma de routage et nous présentons un algorithme qui prouve que cette méthode peut être effectivement mise en oeuvre. Par manque de temps, nous n'avons pas effectué de réalisation logicielle.

Le travail qui reste à faire se décompose en trois axes:

- Expérimenter des variantes de la structure de mémorisation proposée qui supportent des tailles de messages variables.
- Étudier la reconfiguration optimale de topologies régulières tronquées à cause de la panne de quelques routeurs et/ou de quelques liens.
- Réaliser l'outil logiciel implémentant la méthode proposée dans le chapitre 7 en effectuant une optimisation des algorithmes utilisés.

Bibliographie

[1] B. ZERROUK and al. Rcube: A gigabit serial links low latency adaptive router. In *Hot Interconnects Symposium*, Août 1996.

[2] B. ZERROUK and al. The hic serial link technology and associated router. In *Real Time Magazine*, 1996.

[3] N. SANTORO and R. KHATIB. Labeling and implicit routing in networks. In *The Computer Journal*, volume 28, pages 5–8, 1985.

[4] P. KERMANI and L. KLEINROCK. Virtual-cut-through: A new computer communication switching technique. In *Computer Networks*, volume 3, pages 267–286, Septembre 1979.

[5] W. J. DALLY and C. L. SEITZ. The torus routing chip. In *Journal of Distributed Computing*, volume 1, pages 187–196, Avril 1986.

[6] A. COFFLER, M. COMBES, J.C. LEBIHAN, R. MARBOT, and R. NESAMZADEH. Integration of multiple bidirectional point-to-point serial links in the gigabits per second range. In *Hot Interconnects Symposium*, 1993.

[7] F. WAJSB RT. Sp cifications du composant pciddc. 1997.

[8] W. J. DALLY and C. L. SEITZ. Deadlock-free message routing in multiprocessor interconnection networks. In *IEEE Transactions on Computers*, volume C-36, No.5, pages 547–553, Mai 1987.

[9] J. DUATO. A necessary and sufficient condition for dealdlock-free adaptive routing in wormhole networks. In *IEEE Transactions on Parallel and Distributed Systems*, volume 6 No 10, pages 1055–1067, 1995.

[10] E. FLEURY and P. FRAIGNIAUD. Deadlocks in adaptive wormhole routing. In *Rapport de recherche, ENS de Lyon*, Décembre 1994.

[11] B. ZERROUK, A. BOUHRAOUA, and A. GREINER. Definig a new component for the mile simulator. In *Esprit-OMI/MACRAMÉ project report*, Mars 1995.

[12] B. ZERROUK and A. BOUHRAOUA. Mile software architecture overview and principles. In *Esprit-OMI/MACRAMÉ project report*, Mai 1996.

[13] B. ZERROUK and A. BOUHRAOUA. Mile: An open environment for interconnection networks performance evaluation. In *Proc. Parallel and Distributed Computer Systems PDCS'96*.

[14] C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. M. Bender, D. G. Grice, P. H. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker. The sp2 high-performance switch. In *IBM SYSTEMS JOURNAL*, volume 34, No 2, pages 185–204, 1995.

[15] C. J. GLASS and L. M. NI. The turn model for adaptive routing. In *Proceedings of 19th Annu. Int. Symposium on Computer Architecture*, Mai 1992.

[16] B. ZERROUK, J. M. BLIN, and A. GREINER. Encapsulating networks and routing. In *Proceedings of the 8th International Parallel Processing Symposium*, pages 546–553, Avril 1994.

[17] I. D. SCHERSON and A. S. YOUSSEF. Interconnection networks for high performance parallel computers. In *IEEE Tutorial, IEEE Computer Society, 1994, Second Edition*, 1994.

[18] I. D. SCHERSON and C.-K. CHIEN. Self-routing lowest common ancestor networks. In *Proceedings of the IEEE 1992 Symposium on the Frontiers of Massively Parallel Computation*, pages 513–514, 1992.

[19] I. D. SCHERSON and C.-K. CHIEN. Least common ancestor networks. In *International Parallel Processing Symposium*, pages 507–513, Avril 1993.

[20] I. D. SCHERSON and C.-K. CHIEN. Least common ancestor networks. In *Journal of VLSI Design, Special Issue on Interconnection Networks*, volume 2, No.4, pages 353–364, 1995.

[21] D. P. AGRAWAL. Graph theorical analysis and design of multistage interconnection networks. In *IEEE Trans. on Computers*, volume C-32, pages 636–648, Juillet 1983.

[22] D. P. AGRAWAL and J.-S. LEU. Dynamic accessibility testing and path length optimization of multistage interconnection networks. In *IEEE Trans. on Computers*, volume C-34, pages 255–266, Mars 1985.

[23] C. L. WU and T. Y. FENG. On a class of multistage interconnection networks. In *IEEE Trans. on Computers*, volume C-28, pages 649–702, Août 1980.

[24] H. J. SIEGEL and S. SMITH. Study of multistage interconnection networks. In *Proc. of the Fifth Annual Symp. Comput. Architecture*, pages 223–229, Avril 1978.

[25] A. S. YOUSSEF and B. ARDEN. Equivalence between functionnality and topology of log n-stage banyan networks. In *IEEE Trans. on Computers*, volume 39, No.6, pages 829–832, Juin 1990.

[26] S. ABRAHAM and K. PADMANABHAN. Performance of the direct binary n-cube network for multiprocessors. In *Proc. 1986 Int. Conf Parallel Processing*.

[27] G. D. STAMOULIS P. TSENG D. P. BERTSEKAS, C. OZVEREN and J. N. TSITSIKLIS. Optimal communication algorithms for hypercubes. In *Journal of Parallel Distrib. Comput.*, volume 11, pages 263–275, 1991.

[28] Y. CHANG and J. SIMON. Continuous routing and batch routing on the hypercube. In *Proc. 5th ACM Symp. Principles Distrib. Comput.*, pages 272–281, 1988.

[29] A. YOUSSEF and B. NAHARANI. Banyan-hypercube networks. In *IEEE Transactions Parallel and Distributed Systems*, pages 160–169, Avril 1990.

[30] A. BELLACHIA and A. YOUSSEF. Traffic analysis of hypercubes and banyan-hypercubes. In *Proc. 4th Symp. on the Frontier of Massively Parallel Computation*, Octobre 1992.

[31] A. BELLACHIA and A. YOUSSEF. Generalized wrap-around banyan-hypercube networks. In *Technical Report. Mathematics and Computer Science Division, School of Science and Engineering, Al-Akhawayn University in Ifrane, Morroco*, 1996.

[32] W. J. DALLY. Virtual-channel flow control. In *IEEE Transactions Parallel and Distributed Systems*, volume 3, No.2, pages 194–205, Mars 1992.

[33] Y. TAMIR and G. L. FRAZIER. Dynamically-allocated multi-queue buffers for vlsi communications switches. In *IEEE Trans. on Computers*, volume 41, pages 725–737, Juin 1992.

[34] A. AGRAWAL. Limits on interconnection network performance. In *IEEE Transactions Parallel and Distributed Systems*, volume 2, No.4, pages 398–412, Octobre 1991.

[35] W. J. DALLY. Performance analysis of k-ary n-cube interconnection networks. In *IEEE Transactions on Computers*, volume 39, Juin 1990.

[36] J. VAN LEEUWEN and R. B. TAN. Interval routing. In *The Computer Journal*, volume 30, pages 298–307, 1987.

[37] E. RAUBOLD and J. HAENLE. A method of deadlock-free resource allocation and flow control in packets networks. In *ICCC'76 Toronto*, page 483, Août 1976.

[38] S. TOUEG and J. ULLMAN. Deadlock-free packet-switched networks. In *Proceedings of the 11th ACM Symposium on Theory of Computing*, pages 89–98, 1979.

[39] S. TOUEG. Deadlock- and livelock-free packet-switching networks. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, pages 94–99, 1980.

[40] M. GERLA and L. KLEINROCK. Flow control: A comparative survey. In *IEEE Trans. on Communications*, volume COM-28, page 553, Avril 1980.

[41] D. GELERTNER. A dag-based algorithm for prevention of store-and-forward deadlocks in packets networks. In *IEEE transactions on Computing*, volume C-30, pages 709–715, Octobre 1981.

[42] K. D. GUNTHER. Prevention of deadlocks in packet-switched data transport systems. In *IEEE Transactions on Communications*, volume COM-29,No. 4, pages 512–524, Avril 1981.

[43] I. S. GOPAL. Prevention of store-and-forward deadlock in computer networks. In *IEEE Transactions on Communications*, volume COM-33,No. 12, pages 1258–1264, Décembre 1985.

[44] P. M. MERLIN and P. J. SCHWEITZER. Deadlock avoidance in store-and-forward networks. In *IEEE Transactions on Communications*, volume COM-28,No. 3, pages 345–354, Mars 1981.

[45] W. CHOW, H. MADAN, J. PETERSON, D. GRUNWALD, and D. A. REED. Hyperswitch network for the hypercube computer. In *Proceedings of 15th Annu. Int. Symposium on Computer Architecture*, 1988.

[46] C. K. KIM and D. A. REED. Adaptive packet routing in a hypercube. In *Proceedings of 3rd Conf. Hypercube Concurrent Computer Applications*, Janvier 1988.

[47] S. RAGUPATHY, M. R. LEUTZE, and S. R. Schach. Message routing schemes in a hypercube machine. In *Proceedings of 3rd Conf. Hypercube Concurrent Computer Applications*, Janvier 1988.

[48] C. R. JESSHOPE, P. R. MILLER, and J. T. YANTCHEV. High performance communications in processor networks. In *Proceedings of 16th Annu. Int. Symposium on Computer Architecture*, 1989.

[49] C. J. GLASS and L. M. NI. Maximally fully adaptive routing in 2d meshes. In *Proceedings of Int'l Conf. Parallel Processing*, Août 1992.

[50] J. DUATO. On the design of deadlock-free adaptive routing algorithms for multicomputers: design methodologies. In *Proc. Parallel Architectures Languages Europe*, Juin 1991.

[51] W. J. DALLY and H. AOKI. Deadlock-free adaptive routing in multicomputer networks using virtual channels. In *IEEE Transactions on Parallel and Distributed Systems*, volume 4 No 4, pages 466–475, 1993.

[52] S. BORKAR, R. COHN, G. COX, S. GLEASON, T. GROSS, H. T. KUNG, M. LAM, B. MOORE, C. PETERSON, J. PIEPER, L. RANKIN, P. S. TSENG, J. SUTTON, J. URBANSKI, and J. WEBB. iwarp: An integrated solution to high-speed parallel computing. In *Proc. Supercomputing'88*, Novembre 1988.

[53] J. DUATO. A new theory of dealdlock-free adaptive routing in wormhole networks. In *IEEE Transactions on Parallel and Distributed Systems*, volume 4 No 12, pages 1320–1331, 1993.

[54] J. DUATO. A theory to increase the effective redundancy in wormhole networks. In *Parallel Processing Letters*, volume 4 No 1 and 2, pages 125–138, 1994.