Architectures parallèles à connectique programmable : reconfiguration et routage

Philippe Waille

THESE

présentée par

WAILLE Philippe

pour obtenir le titre de

DOCTEUR de
l'INSTITUT NATIONAL POLYTECHNIQUE
de GRENOBLE

(Arrêté ministériel du 23 Novembre 1988)

ARCHITECTURES PARALLELES A CONNECTIQUE PROGRAMMABLE : RECONFIGURATION ET ROUTAGE

Date de soutenance : 11 Septembre 1991

Composition du jury :

Président	Guy	**MAZARE**
Rapporteurs	Bernard	**TOURSEL**
	Gérard	**MICHEL**
Examinateurs	Jean-Paul	**SANSONNET**
	Traian	**MUNTEAN**

Thèse préparée au Laboratoire de Génie Informatique

Je tiens à remercier :

- Monsieur Guy Mazaré, Professeur à l'Institut National Polytechnique de Grenoble, qui me fait l'honneur de présider le jury,

- Monsieur Bernard Toursel, Professeur à l'Université de Lille, et Monsieur Gérard Michel, Directeur Technique à la société APTOR, dont les commentaires m'ont permis d'améliorer ce manuscrit,

- Monsieur Jean-Paul Sansonnet, Professeur à l'Université de Paris Sud, pour l'intérêt qu'il a porté à mon travail en acceptant de participer à ce jury,

- Monsieur Traian Muntean, Directeur de recherches à l'Institut National Polytechnique de Grenoble, sans qui cette thèse n'aurait pas vu le jour. Je veux lui exprimer ma profonde gratitude pour les idées dont il m'a fait part et les encouragements qu'il m'a prodigués tout au long de mon ravail de recherche.

Je dois aussi beaucoup aux très pertinentes suggestions de Messieurs Jacques Briat et Paul Amblard, Maîtres de Conférences à l'Université Joseph Fourier. Qu'ils trouvent ici l'expression de ma reconnaissance pour ce travail aussi ingrat qu'efficace de critique du manuscrit.

Je remercie enfin mes collègues des équipes "SyMPa" et "FLoP" qui m'ont aidé et encouragé, ainsi que Monsieur Jacques Eudes, dont j'ai récupéré l'environnement d'édition de documents.

Cette thèse est dédiée à Christelle.

Quelle curieuse coïncidence ... !

Chapitre 1

INTRODUCTION

1.1 Parallélisme et performances

L'architecture séquentielle de Von Neumann s'est imposée comme l'architecture universelle des machines informatiques d'hier et d'aujourd'hui.

Sa remise en cause au profit d'architectures parallèles répond à trois motivations différentes (non mutuellement exclusives).

- Les systèmes informatiques distribués doivent leur existence à la nature intrinsèquement parallèle des applications auxquels ils sont destinés. Citons à titre d'exemple le domaine de l'automatique industrielle. Il s'agit alors de coordonner l'activité simultanée de plusieurs machines dans une même chaîne de fabrication. Citons également les systèmes de réservation dans les transports.

  L'exemple le plus frappant reste toutefois celui des réseaux informatiques eux-mêmes. Ceux-ci ont fait l'objet d'une recherche intensive pendant plusieurs années avant de tendre vers des standards bien établis.

- Les systèmes soumis à des contraintes de fiabilité et de continuité de service mettent en oeuvre des architectures parallèles redondantes dans un but de tolérance aux pannes. Un élément en panne est alors détecté et neutralisé par des techniques de confinement (déconnexion, réorganisation de la machine) ou de vote majoritaire.

- Le parallélisme est enfin une alternative à l'amélioration de la technologie, susceptible de reculer les limites de taille, de temps et de coût de résolution des problèmes à traiter.

Nous ne considérerons que la dernière de ces motivations ; le recours à des architectures parallèlles permettant d'accroître la puissance des machines par multiplication des unités de calcul.

Les machines parallèles sont donc caractérisées par la présence de plusieurs opérateurs ou processeurs travaillant de concert pour en augmenter la puissance de calcul.

Les principaux critères de classification des architectures parallèles sont la nature des processeurs (degré d'autonomie, complexité et puissancex), leur nombre et la façon dont ils sont connectés entre eux et/ou à la mémoire.

Dans [1], Trew et Wilson dressent un répertoire des principaux constructeurs de machines parallèles et de leurs produits.

Nous excluerons les machines dites à flot de données dans lesquelles l'ordre d'exécution des instructions n'est pas une séquence définie explicitement par le programmeur, mais dépend au contraire de la disponibilité des arguments des opérations à exécuter.

Sont également éliminées les machines dites synchrones, dans lesquelles tous les opérateurs exécutent les mêmes instructions au même moment ; il s'agit essentiellement des machines vectorielles de type "pipeline" et/ou SIMD.

Le supercalculateur vectoriel pipeline le plus connu est le Cray-1 ([1], chapitre 7.1, page 240). Un bon exemple de machine SIMD est fourni par le DAP ([1], chapitre 2.1, pages 14 à 24).

Nous n'étudierons ici que des machines dites MIMD (Multiple Instruction Multiple Data) dont les processeurs :

- constituent autant de machines de Von Neumann autonomes,
- exécutent séquentiellement chacun sa propre suite d'instructions et échangent de l'information avec les autres,
- dont la synchronisation doit être spécifiée explicitement, d'où le qualificatif d'asynchrone également donné à ces machines par opposition aux machines précédentes.

1.2 Absence de mémoire commune

Les machines à mémoire commune préservent le modèle de programmation de l'architecture de Von Neumann, le partage de mémoire restant l'unique primitive de communication et de synchronisation.

Leur point critique réside justement dans l'accès concurent à la mémoire commune par les différents processeurs, qui se traduit par des conflits d'accès entre les processeurs et de fortes contraintes sur le réseau d'interconnexion processeurs-mémoire.

C'est la raison pour laquelle la taille de ces machines reste de l'ordre de la dizaine de processeurs pour les machines à bus (exemple : Sequent [1], chapitre 3.7, pages 106 à 113) et quelques centaines de processeurs

avec des réseaux processeurs-mémoire plus élaborés mais nettement plus coûteux (exemple : BBN Butterfly [1], chapitre 3.2, pages 64 à 75).

Nous allons au contraire étudier les machines parallèles (ou multiprocesseurs) sans mémoire commune. Celles-ci sont constituées d'un ensemble de nœuds interconnectés par des liaisons bipoint (ou liens), chaque nœud étant lui-même une machine de Von Neumann composée d'un processeur et d'une mémoire locale privée.

En dehors de la puissance individuelle des nœuds, deux caractéristiques importantes d'un Multiprocesseur Sans Mémoire Commune (MSMC) sont le graphe défini par le réseau d'interconnexion et la bande passante des liens dont le réseau est constitué.

Le mode de programmation doit être totalement repensé du fait que l'échange de messages devient l'unique moyen de communication et de synchronisation entre les processeurs et doit être géré explicitement par le programmeur.

De plus, lorsqu'elle est possible, la répartition sur plusieurs processeurs d'un calcul donné n'en garantit pas systématiquement l'accélération. Elle implique en effet des synchronisations (au minimum en début et fin de calcul) et des communications (transferts des données initiales et regroupement des résultats) entre les processeurs ; dont le coût peut égaler ou excéder celui du calcul à effectuer.

La durée minimale du calcul suceptible d'être accéléré par exécution sur plusieurs processeurs définit le grain de parallélisme de la machine considérée et la finesse de la parallélisation de l'appli à effectuer.

Le mécanisme de communication par échange de messages, moins efficace que le partage de mémoire, se traduit par un grain de parallélisme moins fin pour les machines sans mémoire commune.

L'extensibilité des MSMC résultant de l'allègement des contraintes sur le réseau d'interconnexion permet par contre d'envisager des puissances de calcul remarquables par multiplication du nombre de processeurs.

1.3 De l'application au parallélisme physique

Le développement d'une application sur un MSMC peut être décomposé en trois étapes, correspondant à des degrés croissant de prise en compte des caractéristiques de la machine cible.

Analyse et programmation

La première étape sonsiste en une analyse analyse de l'application débouchant sur sa traduction sous la forme d'un programme parallèle sans tenir compte de la taille de la machine cible.

L'application sera au contraire analysée le plus finement possible de manière à détecter toutes les opprtunités de parallélisme qu'elle récèle.

Reléguer la prise en compte des caractéristiques physiques de la machine cible aux étapes ultérieures présente un triple intérêt :

- évaluation de l'adéquation entre l'application et une mise en œuvre sur un MSMC permet d'éviter un travail inutile de codage au cas où le gain (potentiel) de performances s'avèrerait insuffisant et de déterminer la taille maximale de la machine cible (dans le cas favorable),
- préservation du travail de codage de l'application en cas d'évolution du cahier des charges imposant un redimensionnement de la machine et
- une plus grande liberté dans la manière de répartir le travail entre les différents processeurs.

La structure du programme peut être condensée sous la forme d'un graphe (des flux) inter-tâches dont chaque sommet correspond à une tâche et chaque arc à un flux de messages entre les deux tâches qu'il relie. Pour le travail de placement, les sommets et les arcs de ce graphe seront pondérés par des estimations des charges de calcul et des volumes de messages à échanger.

Si nous remplaçons chaque sommet par un processeur et chaque arc par un lien, un tel graphe décrit de fait une machine virtuelle la plus rapide possible pour l'application.

Notons au passage que si toutes les tâches communiquent et se synchronisent deux à deux, le réseau d'interconnexion de cette machine devient un Réseau Totalement Maillé (RTM), c'est-à-dire tel qu'il existe un lien direct de tout processeur vers chacun des autres ; tous les processeurs étant donc directement voisins.

L'analyse et la programmation effectuées, l'utilisateur prend en compte les critères de coût et de performances du cahier des charges et détermine la (taille de la) machine physique à utiliser.

Les deux étapes suivantes consistent alors à projeter la machine virtuelle définie par le programme sur cette machine cible.

Placement des tâches

Dans un premier temps, chaque tâche du programme parallèle doit être affectée à un nœud de la machine cible : cette opération est connue sous le nom de placement des tâches sur les processeurs.

La recherche d'un placement optimal (temps d'exécution minimal) est une opération délicate qui doit à la fois tenir compte de la durée des calculs réalisés par les tâches et des délais de communication.

Le placement se décompose en fait en deux opérations :

- le regroupement des tâches à placer ensemble sur un même nœud, c'est-à-dire la contraction du graphe des tâches en un graphe (des flux) inter-processeurs, de degré égal à la taille de la machine cible.
- le placement proprement dit, c'est-à-dire l'association biunivoque des groupes de tâches ainsi formés aux processeurs de la machine cible.

Projection des flux de communication

Pour d'évidentes raisons technologiques, les MSMC ne peuvent être munies d'un réseau d'interconnexion totalement maillé.

Les MSMC ne fournissent donc qu'un nombre limité de liaisons bi-points entre les processeurs, au-dessus desquelles le graphe des flux inter-processeurs devra être reconstruit par multiplexage.

Notons que ce multiplexage entraîne une vitesse de communication non uniforme, qui va dépendre de la position relative des deux nœuds.

Le problème du placement optimal des tâches s'avère ainsi très complexe, puisque ce placement à la fois dépend et se répercute sur les coûts de communication entre les nœuds.

Les composantes de la machine virtuelle

La reconstruction de la machine virtuelle du programme de l'utilisateur peut donc être scindée en trois grandes parties :

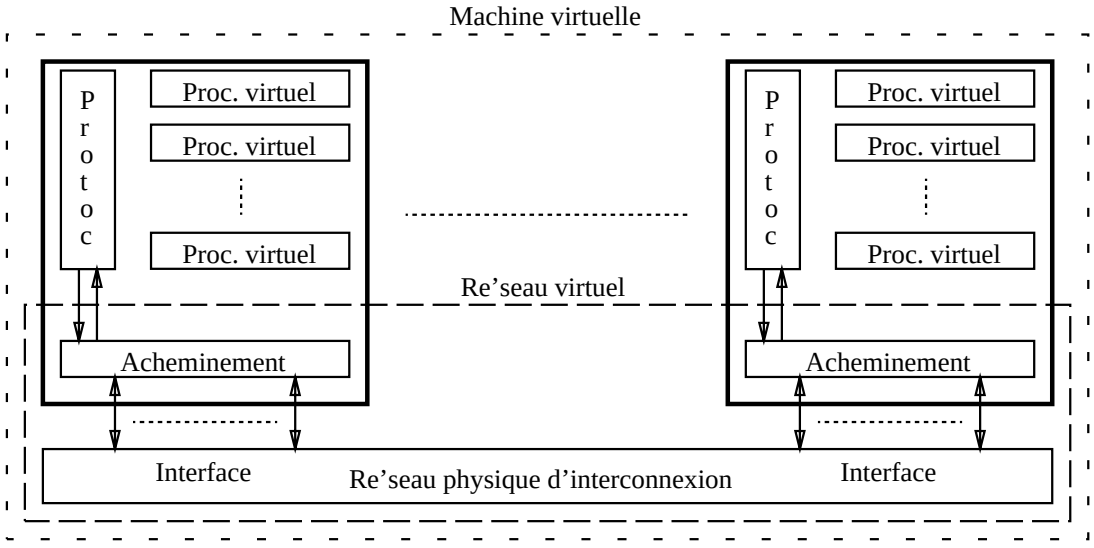


Figure 1.1: Construction de la machine virtuelle

- la simulation sur chaque nœud du nombre de processeurs virtuels nécessaires, par des techniques classiques de multiprogrammation,

- la gestion des protocoles, confiée à une unité du même nom, chargée de la traduction de ceux-ci en échanges de messages. Les communications intra-nœud seront réalisée directement par l'unité locale via la mémoire et

- l'acheminement des messages, via le réseau physique d'interconnexion, entre les unités des différents nœuds en cas de communications non locales ; qui fait l'objet de la présente étude.

1.4 Multiprocesseurs et réseaux

La structure présentée ci-dessus ne différe pas fondamentalement de celle d'un réseau informatique classique (Arpanet, Transpac, etc...). En particulier, la nécessité de reconstruire un réseau totalement maillé virtuel au dessus d'un réseau physique donné se retrouve dans les deux cas.

En dépit de cette similitude de strcuture, les résultats acquis sur les réseaux ne sont pas directement transposables aux MSMC. En effet, ces deux types d'architecture diffèrent notablement par la fiabilité des liaisons et leurs objectifs de service :

- Dans un MSMC, les processeurs sont physiquement proches les uns des autres et exécutent de concert un même algorithme. Ceci implique d'une part des liaisons courtes (de quelques centimètres à quelques mètres), donc fiables. D'autre part, la panne d'un processeur constitue une anomalie qui affecte le travail de toute la machine : le travail en cours peut alors être abandonné et réexecuté après neutralisation de l'élément défaillant.

- Dans un réseau tel que Transpac, les processeurs sont au contraire éloignés géographiquement et fonctionnent de manière beaucoup plus indépendante les uns des autres. Du fait des distances à parcourir (centaines, voire millier de kilomètres), la fiabilité des liaisons laisse au contraire à désirer (bruits électromagnétiques, avaries dues aux intempéries, etc...). D'autre part, l'arrêt momentané d'un nœud du réseau est un évènement relativement fréquent (panne de courant, maintenance matérielle ou logicielle, etc..) qui fait partie de la vie normale d'un réseau et doit perturber le moins possible le fonctionnement des autres processeurs..

Les différence de localité se traduisent également par de fortes disparités de vitesse relative entre calcul et communication :

- Les bandes passantes courantes des liens d'un MSMC sont d'environ un à dix mégaoctets par seconde. L'ordre de grandeur du délai de transfert d'un octet correspond donc à l'exécution d'une dizaine d'instructions machine.

- Dans des réseaux, au contraire, la bande passante des liaisons est généralement plus proche de la dizaine de kilooctets par seconde, soit de l'ordre du millier d'instructions par octet transféré.

En résumé, dans un réseau, la continuité du service est essentielle et les liaisons entre les noeuds tendent à former des goulets d'étranglement. Ceci explique le recours à des stratégies très élaborées de répartition du traffic entre les différentes liaisons et de récupération d'erreurs dont la complexité rend la mise en œuvre assez lourde.

Pour un MSMC, au contraire, la rapidité d'acheminement est vitale et les décisions concernant l'acheminement des messages doivent être prises en un temps réduit sous peine de ralentir la machine. Par contre, la fiabilité des éléments est relativement élevée et les contraintes de continuité de service moins fortes. C'est pourquoi les stratégies utilisées sont plus rudimentaires et la gestion des erreurs allégée (voire supprimée) ; les algorithmes, plus simples, pouvant être exécutés plus rapidement. Ce principe peut être poussé jusqu'à une mise en œuvre totalement câblée.

1.5 Machines à connectique fixe

Comme nous l'avons vu précédemment, le réseau d'interconnexion d'un MSMC ne comprend qu'un nombre limité de liens du réseau totalement maillé (RTM), sur lesquels le réseau de la machine virtuelle idéale sera reconstruit par multiplexage.

Les MSMC sont le plus souvent munis d'un réseau d'interconnexion à connectique fixe et régulière choisie définitivement lors de la construction de la machine.

Les réseaux d'interconnexion fréquemment utilisés sont l'hypercube (utilisé par NCUBE et INTEL iPSC [1], chapitre 4, pages 125 à 144) et la grille (AMETEK [1], chapitre 10.5, pages 342 à 345).

Le réseau peut être décrit par un graphe dont les sommets sont les processeurs et les arcs les liens ; représentation à laquelle nous nous référerons en parlant de topologie du réseau.

Tout message qui ne bénéficie pas d'une liaison directe devra être routé, c'est-à-dire propagé de (nœud) voisin en voisin en empruntant une suite de liens conduisant à sa destination.

Le chemin suivi par un message est construit incrémentalement au fur et à mesure de la progression de celui-ci, qui est précédé pour ce faire d'un en-tête encodant sa destination.

Le chemin à utiliser est prédéfini (routage déterministe) pour chaque paire (source,destination) et encodé dans autant de tables que de nœuds de la machine.

Un nœud recevant un message qui ne lui est pas adressé consulte sa table de routage locale et le réémet sur le lien de sortie spécifié par celle-ci.

Deux phénomènes réduisent la vitesse d'acheminement d'un message avec la distance à parcourir :

- Chaque nœud introduit un délai pour initialiser la recopie sur le lien de sortie.
- Les messages se partagent un même ensemble de liens, qui sont des ressources à accès exclusif. Deux messages parcourant des chemins non disjoints peuvent ainsi entrer en conflit pour l'utilisation d'un lien ; la progression d'un deux deux messages est alors suspendue jusqu'à la libération du lien par l'autre message.

  Notons que l'existence de tels conflits peut non seulement ralentir, mais stopper totalement la progression des messages par interblocage lorsqu'un cycle de dépendance est formé. La lutte contre l'interblocage sera abordée dans le chapitre 5.

Les performances de l'application vont donc dépendre fortement de la qualité du placement des tâches sur les processeurs.

Malheureusement, le calcul du placement optimal est voué à l'échec. En effet, celui-ci suppose la connaissance préalable de toute la trace d'exécution du programme, c'est-à-dire :

- la durée de chaque calcul élémentaire et
- le déroulement de toutes les communications.

Or une telle hypothèse ne peut être vérifiée. D'une part elle impliquerait que le programme soit totalement déterministe, cas peu réaliste ; d'autre part la durée des communications dépend elle-même du placement effectué.

En pratique, le placement est généré à partir d'estimations de charges de calcul, et de fréquence des communications, par des méthodes telles que le recuit simulé ou de partitionnement de graphe, adaptées à la complexité du problème.

1.6 Reconfiguration synchrone

Le principe même des machines à connectique fixe, les met à la merci d'une inadéquation plus ou moins pronocée entre la topologie choisie lors de la construction et le réseau souhaitable pour l'application.

De par ses répercussions sur la longueur des chemins et, plus précisément sur la fréquence des conflits d'accès aux liens, une telle inadéquation constitue un obstacle à l'obtention de performances.

Cet inconvénient peut être atténué par l'abandon de connexions fixes et directes entre les processeurs au profit de liaisons bipoint réalisées au travers d'un réseau de commutation.

La connectique du réseau devient programmable et nous aboutissons ainsi à la notion de réseau de commutation et de machine reconfigurable[1].

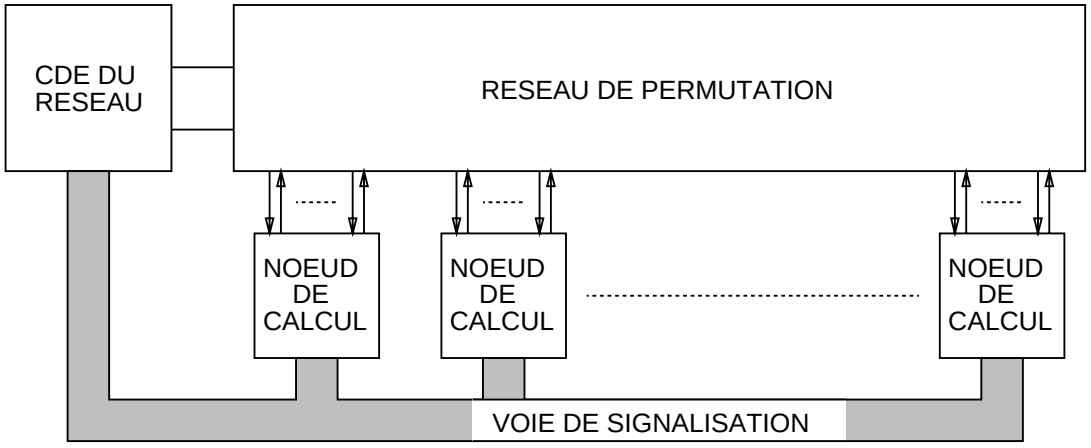


Figure 1.2: Structure d'une machine reconfigurable

L'exécution de chaque application sera ainsi précédée d'une reconfiguration (ou reprogrammation du réseau) selon la topologie la plus proche possible du réseau idéal pour l'application.

Nous qualifierons ce mode de fonctionnement de reconfiguration de :

- statique, comme intervenant avant et non au cours de l'exécution et
- synchrone dans la mesure où la reprogrammation est effectuée à des instants déterminés et porte sur l'ensemble des connexions réalisées par le réseau.

La reconfiguration dynamique synchrone convient aux applications décomposables en une séquence de calculs (ou étapes de calcul) parallèles. Chaque étape de calcul peut elle-même bénéficier d'une reprogrammation de la connectique du réseau selon la topologie la plus adéquate. Une transition d'étape donnant lieu à reconfiguration sera appelée point de reconfiguration.

Lorsqu'un processeur en a terminé avec les calculs associés à l'étape courante, il en informe le dispositif de commande du réseau de commutation. La mise en œuvre de la reconfiguration synchrone suppose l'existence d'une voie de signalisation pour l'acheminement de ces messages.

L'arrivée du message du dernier processeur indique que le point de reconfiguration a été atteint et la reprogrammation du réseau est effectuée. Lorsque cette reprogrammation est terminée, l'unité de commande du réseau diffuse en retour aux processeurs un message de reprise qui déclenche l'exécution de l'étape suivante de calcul.

[1]La reconfiguration ou réorganisation physique d'une machine n'est pas en elle-même une technique récente : la nouveauté réside dans son utilisation pour la recherche de performances, et non plus de tolérance aux pannes.

Les primitives de base pour la mise en œuvre de la reconfiguration dynamique synchrone sont donc la diffusion et le rendez-vous à n participants.

Que la reconfiguration soit, statique ou dynamique, entre deux reprogrammation du réseau, la machine est semblable à une machine à connectique fixe (mais éventuellement irrégulière) sur laquelle les messages doivent être routés lorsqu'ils ne bénéficient pas d'une liaison directe de source à destination.

1.7 Reconfiguration asynchrone

La méthode précédente, reposant entièrement sur une optimisation statique des coûts de communication en agissant sur le placement des tâches, est susceptible d'être mise en échec.

Une telle optimisation implique en effet un minimum de stabilité dans la répartition des flux de communications entre les tâches au cours de l'exécution, ce qui exclut les applications à comportement trop dépendant des données traitées (notamment dans le domaine de l'intelligne artificielle).

Cette répartition doit de plus être connue à priori lors du placement et, au-delà de la complexité intrinsèque des algorithmes de placement, cette approche butte sur l'élaboration d'outils d'estimation des communications à partir du texte du programme et/ou de statistiques d'exécution ("profiling").

Nous sommes ainsi amenés à rechercher un nouveau mode d'acheminement des messages permettant d'uniformiser les coût de communication entre les différents processeurs.

La reconfiguration synchrone n'offre finalement que deux moyens de lutter contre les conflits d'accès aux liens résultant d'un placement inapproprié :

- différer de la compilation à l'exécution le choix des topologies à utiliser pour mieux prendre en compte l'évolution des flux de communication, puis
- rapprocher les points de reconfiguration pour augmenter la précision des corrections apportées à ce choix.

Poussé à son extrème, ce raisonnement conduit à l'abandon du routage des messages de processeur en processeur au profit d'une méthode d'acheminement des messages par connexion bipoint à la demande.

Celle-ci consiste à établir systématiquement (par reconfiguration du réseau) une connexion directe de source à destination préalablement à (et pour la durée de) chaque transfert de message en retardant au besoin l'exécution de ce dernier lorsque cette connexion ne peut être établie immédiatement.

Dans la mesure où le délai d'établissement d'une connexion peut être rendu indépendant de la position géographique des correspondants, tous les processeurs deviennent virtuellement voisins deux à deux. La notion de distance entre processeurs devient alors caduque et les coûts de communication uniformes sur l'ensemble de machine.

Un tel fonctionnement n'est évidemment viable que si les liaisons peuvent être individuellement créées ou détruites indépendamment les unes des autres, c'est-à-dire sans affecter les communications en cours entre les nœuds : nous parlerons donc de reconfiguration incrémentale ou asynchrone par opposition à la reconfiguration synchrone affectant l'ensemble de la machine.

1.8 Le Projet Supernode

Ce travail a été entrepris dans le cadre du projet de recherche européen Supernode[2] visant à développer un supercalculateur à faible coût exploitant les techniques du parallélisme massif.

Du point de vue architectural, l'ensemble du projet repose sur trois décisions fondamentales :

1. la contruction d'un multiprocesseur sans mémoire commune
2. le recours à la reconfiguration (initialement de type statique) pour l'optimisation des communications et
3. le choix des microprocesseurs de la famille Transputer[3] comme processeurs de calcul.

La répartition des tâches entre les différents partenaires était en gros la suivante :

- conception matérielle par l'Université de Southampton et R.S.R.E (Royal Signal and Radar Etablishment); ce dernier assurant de plus la supervision du projet,
- construction et commercialisation des protoypes par TELMAT et THORN EMI,
- conception et fabrication par INMOS, dans le cadre et pour les besoins du projet, du transputer T800 intégrant une unité de calcul en virgule flottante,
- pour le Laboratoire de Génie Informatique, développement du logiciel de base et plus particulièrement de la gestion des communications et de la reconfiguration, auquel se rattache la présente étude et

[2]ESPRIT P1085

[3]Transputer et OCCAM sont des marques déposée d'INMOS

- écriture de bibliothèques d'exécution et de logiciels d'application par les Universités de Liverpool et Oxford[4] (calcul numérique) et par APSYS/APTOR (CAO électronique et réseaux locaux).

L'architecture modulaire reconfigurable issue du projet se décline en toute une gamme de puissances de calcul.

Elle a ainsi donné naissance à toute une famille de multiprocesseurs reconfigurables, aujourd'hui commercialisée par TELMAT et THORN EMI et les versions prototypes de ces machines disponibles au LGI ont servi de plateforme d'expérimentation de la reconfiguration.

Le travail se poursuit dans le cadre de ESPRIT II, dont le projet P2528 a pour objectif d'un système d'exploitation adapté à l'architecture Supernode.

D'autres machines reconfigurables à base de transputers sont également fabriquées par des constructeurs concurents tels que Meiko, Parsytec ou Cogent Research ([1], pages 165 à 175, 187 à 195 et 202 à 206).

1.9 Objectifs et plan de l'exposé

L'analyse des deux modes de reconfiguration, synchrone et asynchrone, auxquels est consacrée cette thèse, fait l'objet des chapitres 5 et 6.

Cette analyse répond à trois préoccupations différentes :

- l'identification et la résolution des problèmes spécifiques posés par la reconfiguration,
- la conception matérielle des machines en vue d'un support efficace de la reconfiguration et l'évaluation de l'architecture Supernode de ce point de vue,
- la définition précise des deux modes de reconfiguration et leur classification parmi les autres méthodes d'acheminement (routage) des messages.

La reconfiguration synchrone est étudiée dans le chapitre 5, dont l'essentiel est consacré au problème du routage des messages sans interblocage en présence des topologies irrégulières qu'elle permet d'utiliser.

Le reste du chapitre traite du problème annexe d'enchaînement des phases pour la reconfiguration synchrone dynamique d'une part ; et aux adaptations du routeur aux caractéristiques spécifiques des transputers et des machines Supernode.

Dans le chapitre 6, trois réseaux de complexité croissante seront successivement envisagés pour la mise en œuvre de la reconfiguration dynamique asynchrone (RDA) :

[4] En tant que sous-contractant

- un réseau totalement maillé pris comme réseau de commutation
- un crossbar
- un réseau multi-étages de Clos.

La première partie du chapitre revient sur le principe de liaison bipoint à la demande ; elle clarifie et regroupe au sein dans un modèle unifié les notions de routage et de reconfiguration.

Dans ce modèle, RDA et routage déterministe (de type "wormhole") constituent deux variantes extrèmes d'une forme générique de routage adaptatif, privilégiant respectivement l'optimalité et la vitesse de calcul des chemins empruntés par les messages.

Lorsque le réseau de commutation n'offre qu'un seul chemin par paire de processeurs (cas du réseau totalement maillé et du crossbar), la distinction entre routage et reconfiguration est sans objet.

L'utilisation d'un réseau de Clos est examin'ee dans un deuxième partie. La fin du chapitre illustre enfin l'utilisation de la reconfiguration dynamique asynchrone à des fins d'équilibrage de charge entre les processeurs.

La première partie de la thèse regroupe les connaissances préliminaires requises par ces deux derniers chapitres et constitue un document de référence pour les machines Supernode :

- Le chapitre 2 passe en revue les principaux réseaux de commutation et leurs propriétés topologiques et présente plus particulièrement les résaux de Clos sur lesquels repose l'architecture Supernode.
- Le chapitre 3 décrit les caractéristiques des transputers les plus significatives pour la construction de machines parallèles reconfigurables.
- Les fondements de l'architecture Supernode sont abordés dans le chapitre 4. Pour plus de clarté, la description détaillée des machines est reportée en annexe.

Chapitre 2

RESEAUX DE COMMUTATION

Un réseau de commutation est un dispositif réalisant des connexions entre ses m entrées d'une part et ses n sorties d'autre part de telle sorte que :

- toute entrée puisse être connectée à n'importe quelle sortie et que
- à tout moment toute sortie soit reliée à au plus une entrée

Sa présence en lieu et place de liaisons bipoint fixes, qui permet de programmer la topologie d'interconnexions entre les processeurs, est la caractéristique fondamentale de toute machine reconfigurable.

Le nombre d'entrées est alors égal à celui des sorties ; ce nombre définit la taille (ou nombre de liens commutées) du réseau de commutation, auquel le nom de réseau de permutation peut également être donné.

L'efficacité de reconfiguration synchrone repose sur une meilleure adaptation entre les flux de communication et le réseau d'interconnexion.

Celle-ci sera donc d'autant meilleure que la palette de topologies d'interconnexion sera plus étendue, d'où l'intérêt de disposer d'un réseau de commutation capable de réaliser toutes les permutations possibles de ses entrées vers ses sorties (notion de réarrangeabilité).

Cette propriété devient indispensable dans le cas de la reconfiguration dynamique asynchrone (RDA) ; auquel cas les modifications de topologies doivent de plus être effectuées de manière incrémentale (notion de blocage).

Deux autres caractéristiques sont également dignes d'intérêt :

- Comme nous le verrons dans le chapitre 3, le retard subi par les signaux commutés est susceptible de réduire la bande passante effective des liens des transputers, d'où l'intérêt de minimiser le nombre d'éléments de commutation traversés par chaque connexion.

- Les performances de la RDA sont directement liées à là complexité de l'algorithme de choix d'un chemin dans le réseau de commuation (lorsque celui-ci offre plusieurs manières de réaliser une même connexion).

Après élimination du crossbar, nous passerons rapidement en revue les principaux types de réseaux de commutation pour aboutir au seul réseau capable de satisfaire la condition d'absence de blocage et sur lequel repose l'architecture Supernode.

2.1 Réarrangeabilité at absence de blocage

Soit un réseau de permutation réalisant des liaisons entre un ensemble d'entrées E et un ensemble de sorties S.

Soient A et B deux sous-ensembles respectifs de E et S, à m éléments chacun. Soit f une bijection de A vers B.

Le réseau admet un état $X(A, B, f)$ lorsqu'il est capable de réaliser simultanément toutes les connexions entre A et B décrites par f.

Définition 1 : Un réseau est dit *réarrangeable* lorsque pour tout A,B et f, il admet l'état (A, B, f). En d'autre termes, un réseau réarrangeable permet de réaliser n'importe quelle bijection des entrées vers les sorties.

Soit $e \in E - A$ et $s \in S - B$ une entrée et une sortie non connectées.

Soit f' une bijection telle que :

$f'(x) = f(x)\ \forall x \in A\ et\ f'(e) = s$

Soit X' ($A \cup \{e\}$,$B \cup \{s\}$,f') un nouvel état du réseau réarrangeable obtenu en ajoutant une nouvelle connexion à X.

Définition 2 : Le réseau est dit *non bloquant* lorsqu'il est capable de réaliser directement les transitions de X vers X' ou de X' vers X quelque soit la paire (X, X').

Les deux définitions ci-dessus s'inspirent de l'article original de Charles Clos [9].

Autrement dit, un réseau non bloquant est capable d'établir ou de supprimer une connexion sans interrompre les liaisons existantes.

L'intérêt d'un réseau réarrangeable pour la reconfiguration synchrone est d'offrir une totale liberté dans le choix de la topologie du réseau d'interconnexion (sous réserve bien sûr qu'elle soit compatible avec le nombre de liens de chacun des nœuds).

L'absence de blocage ne représente quant à elle rien d'autre que la possibilité de modifier incrémentalement la connectique du réseau, c'est à dire de reconfigurer le réseau de manière asynchrone.

2.2 Le crossbar

Le réseau matriciel ou crossbar m vers r, obtenu par juxtaposition de r multiplexeurs indépendants m vers un, est le plus simple et le plus performant des réseaux de commutation (figure 2.1).

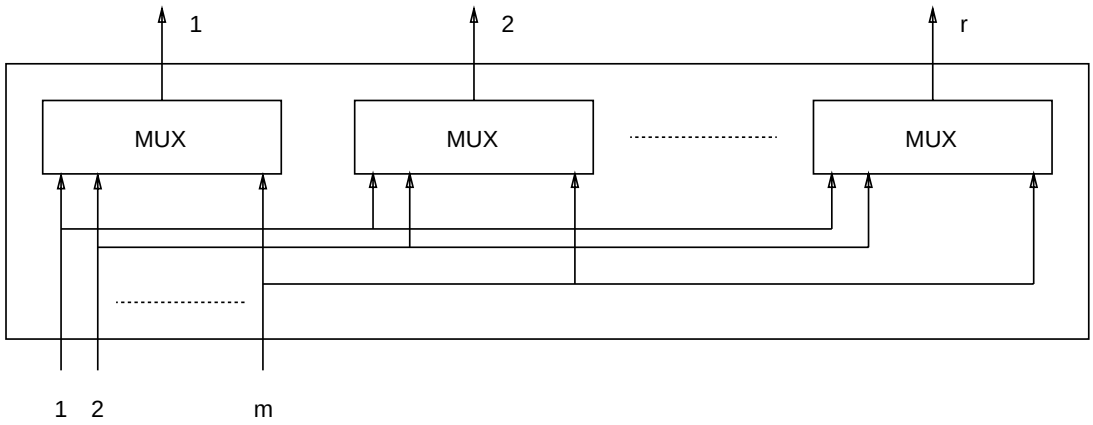


Figure 2.1: Le crossbar ou réseau matriciel

Le crossbar est en effet réarrangeable et non bloquant ; chaque connexion ne traverse qu'un seul multiplexeur et ne pose pas le problème du choix du chemin à utiliser (une seule possibilité).

Le coût de réalisation d'un crossbar (nombre de portes) est malheureusement proportionnel au carré du nombre de liens ($P = \alpha n^2$). La taille limitique d'un crossbar monilithique est de l'ordre de la centaine de multiplexeurs: au-delà le coût de réalisation devient rapidement prohibitif.

2.3 Topologies quelconques et étagées

Pour repousser cette limite, il existe de multiples façons de construire un réseau de permutation qui diffèrent par la manière d'interconnecter ceux-ci [8] [10].

Tous les réseaux d'interconnexion destinés aux machines à connectique fixe peuvent être transformés en réseaux de commutation.

En chaque sommet du graphe d'interconnexion, il suffit pour cela de remplacer le processeur par un crossbar et de connecter à ce dernier une entrée et une sortie supplémentaires.

Nous pouvons ainsi construire des réseaux de permutation à partir de toutes les topologies courantes telles que grilles, tores, ou hypercubes (figure 2.2).

Curieusement, cette manière de construire des réseaux de permutation est rarement abordée dans la littérature : par exemple, la question de la réarrangeabilité d'un réseau aussi populaire que l'hypercube ne semble pas avoir été définitivement tranchée [14].

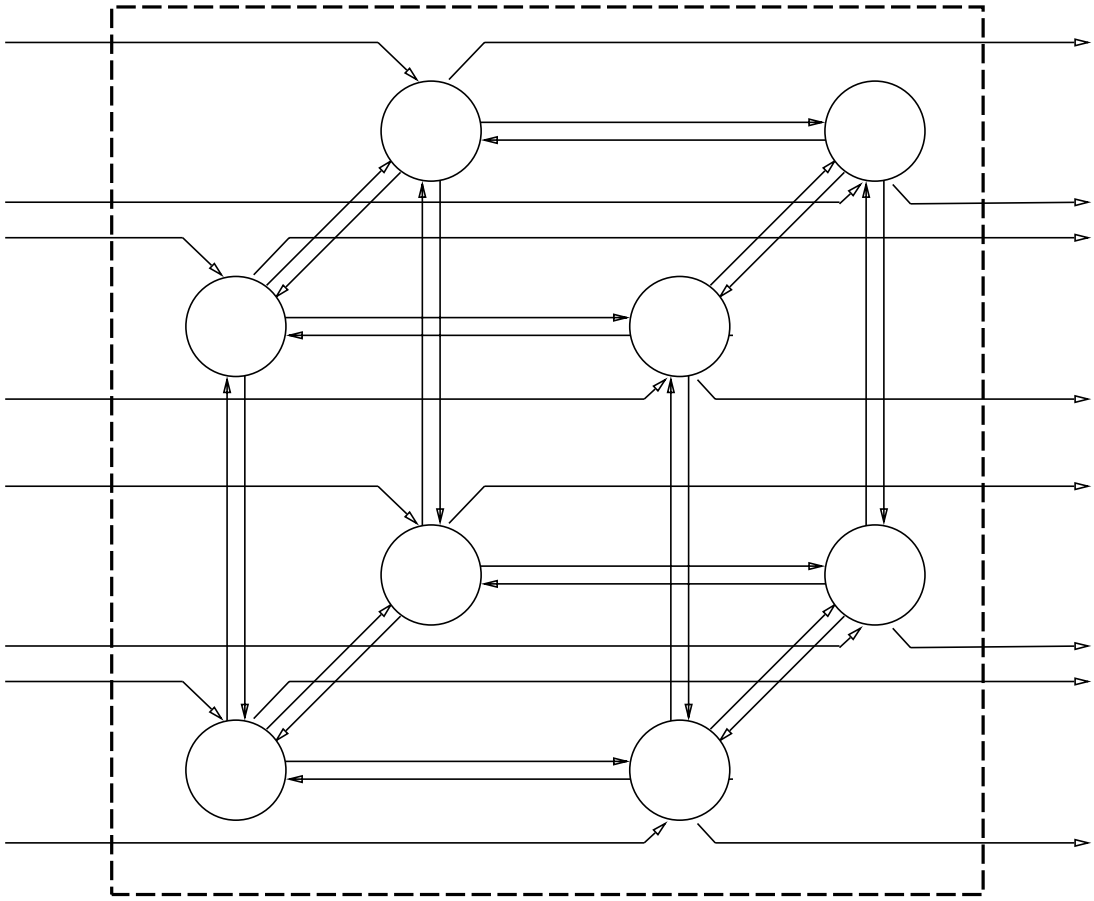


Figure 2.2: Transformation d'un hypercube en réseau de permutation

Nous nous limiterons à la seule catégorie des réseaux multi-étages qui regroupe la quasi totalité des réseaux de commutation effectivement étudiés et utilisés. Ces réseaux présentent trois caractéristiques :

- Le graphe d'interconnexion des crossbars est acyclique[1] : les crossbars sont regroupés en un ensemble ordonné d'étages de telle sorte que tout crossbar d'un étage ne soit connecté qu'à des crossbars de l'étage suivant.

[1]Exception faite des fausses boucles introduites par une transformation en un réseau unilatéral (voir description du réseau de Clos ci-dessous).

- Tous les liens d'entrée du réseau sont connectés aux seuls crossbars du premier étage.
- Inversement, tous les liens de sortie sont attachés aux crossbars du dernier étage.

Ils peuvent être à leur tour regroupés en deux familles : les réseaux de type dichotomique à base de crossbars deux vers deux et les réseaux de Clos.

2.4 Construction récursive par dichotomie

Un réseau de type dichotomique, de taille $N=2^n$ est une cascade de n étages composés chacun de N/2 crossbars deux vers deux.

Le graphe d'interconnexion est défini récursivement, le réseau de taille deux étant le crossbar deux vers deux lui-même.

La figure (2.3) présente à titre d'exemple la méthode de construction du réseau dit "baseline".

Il existe autant de variétés de réseaux à $Log_2(n)$ étages que de façons de réaliser la décomposition récursive (Baseline, Banyan, Omega, etc ...) ; chacun admettant une variante directe et une variante inverse obtenue en permutant les entrées et les sorties.

Tous ces réseaux sont équivalents du point de vue du nombre de permutations réalisables et aucun d'entre eux n'est réarrangeable [5] [4] [6] [13] [15].

Il en va de même pour le réseau ADM (Augmented Data Manipulator) (figure 2.3), de conception très voisine, bâti à partir de crossbars trois vers trois [12] .

Actuellement, ces réseaux sont surtout utilisés pour la construction de machines à mémoire commune.

2.5 Réseaux de Clos et de Benĕs

2.5.1 Principe

L'association de crossbars en réseaux de permutation réarrangeables et non bloquants a été étudié initalement dans le cadre de la téléphonie par Clos et Benĕs [9] [7].

Le réseau initial de Clos (figure 2.4) comprend :

- un étage intermédiaire muni de q crossbars à p entrées et p sorties reliant

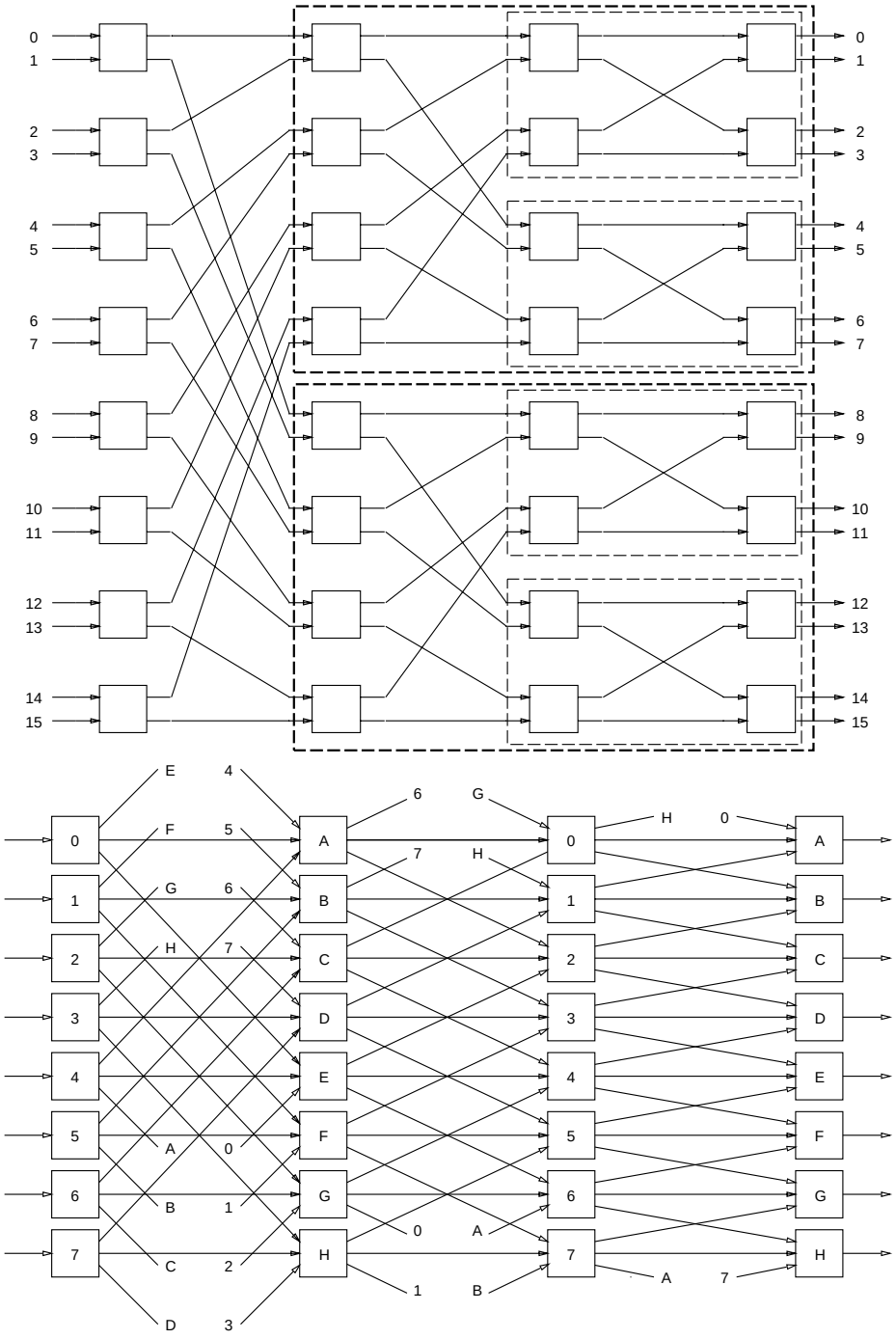


Figure 2.3: Réseaux baseline et ADM

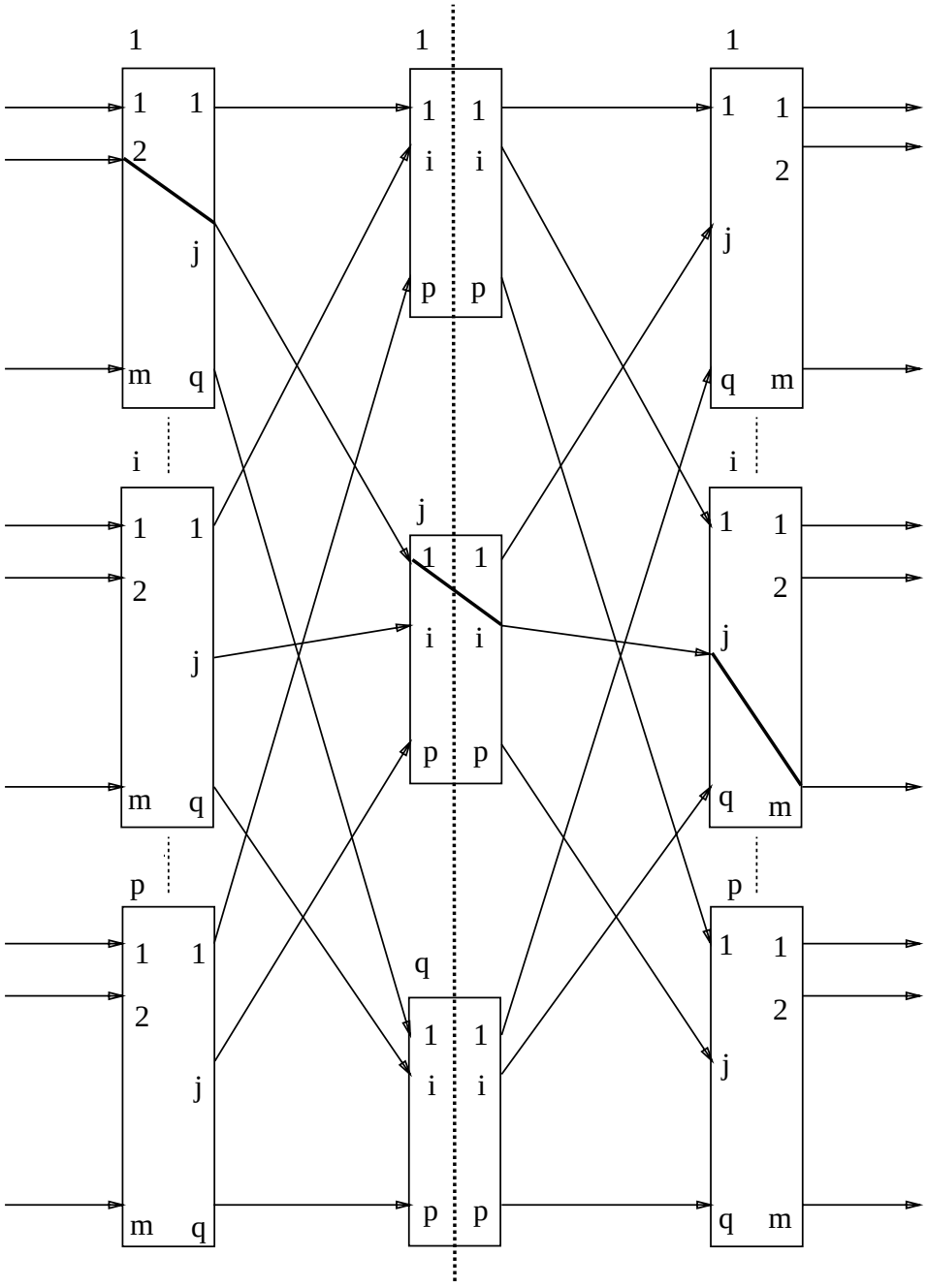


Figure 2.4: Le réseau original de C. Clos

- un étage d'entrée comprenant p crossbars à m entrées vers q sorties d'une part et
- un étage symétrique de sortie composé de p crossbars à q entrées et m sorties.

Sur la figure 2.4, l'entrée numéro deux du crossbar numéro un du premier étage est connectée à la sortie m du crossbar numéro i du troisième étage via le j^{ieme} crossbar de l'étage intermédiaire.

Un tel réseau est réarrangeable si :

$$q \geq m \tag{2.1}$$

Il est de plus non bloquant si :

$$q \geq 2m - 1 \tag{2.2}$$

On peut construire des réseaux de Clos de plus grande taille à 2k + 1 étages en remplaçant les crossbars de l'étage intermédiaire par des réseaux de Clos à 2k - 1 étages.

Un réseau de Benĕs (figure 2.5) à 2k-1 étages peut être défini comme la mise en série d'un réseau baseline à k étages et de son réseau inverse.

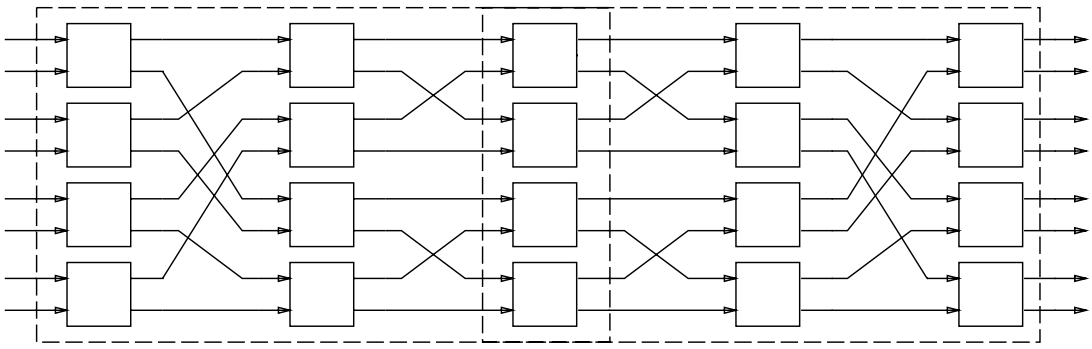


Figure 2.5: Réseau de Benĕs

Le réseau de Benĕs n'est qu'un cas particulier de réseau de Clos tel que m=p=q=2 ; il est donc réarrangeable.

2.5.2 Transformation en réseau unilatéral

Les réseaux de Clos présentés ci-dessus, dont les liens d'entrée et de sortie occupent les deux extrémités, peuvent être qualifiés de bilatéraux.

Le réseau de Clos peut être transformé en réseau unilatéral dont les entrées et les sorties se trouvent du même côté (figure 2.6).

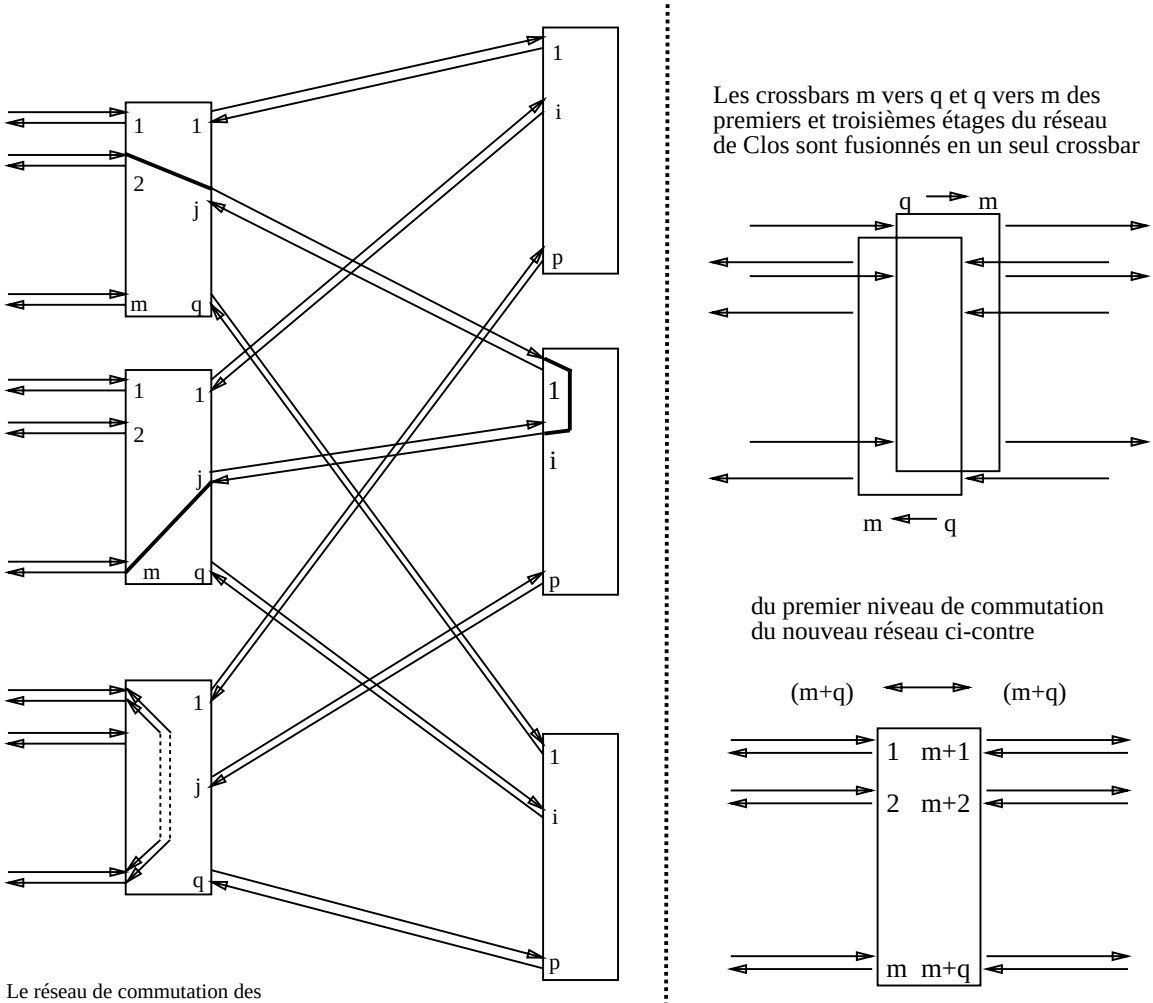


Figure 2.6: Le réseau unilatéral de Clos des machines Supernode

Il suffit pour cela de replier la figure 2.4 suivant son axe de symétrie et de remplaçer chaque paire de crossbars superposés (m vers q et q vers m) des premier et troisième étages par un crossbar unique m+q vers m+q.

L'étage intermédiaire, non modifié, devient le second des deux niveaux du réseau unilatéral de Clos ainsi obtenu ; le premier niveau de commutation correspondant à la fusion des anciens premier et troisième étages.

Les deux versions du réseau de Clos possèdent les mêmes propriétés de réarrangeabilité et d'absence de blocage.

Plus avantageuse, la version unilatérale est de plus capable de relier directement (sans faire intervenir le deuxième niveau de commutation) des entrées et des sorties appartenant à un même crossbar.

Le temps de traversée est ainsi réduit d'un facteur trois, ce qui justifie son emploi dans les machines Supernode.

2.5.3 Taille et coût

Il convient d'évaluer la taille du réseau de Clos à partir de laquelle sa part dans le prix de revient de la machine devient prohibitive pour déterminer si la construction de machines reconfigurables de taille significative est possible.

Une estimation simple de cette part consiste à comparer le nombre de crossbars et la taille du réseau (c'est-à-dire le nombre de processeurs).

J'ai donc entrepris de calculer respectivement :

- la taille t_k du réseau et
- le nombre total c_k de crossbars, absolu et rapporté au nombre de processeurs

d'un réseau unilatéral de Clos à k+1 niveaux en supposant que tous les crossbars ont une même taille de n liens.

Deux cas de figure ont été considérés : un réseau bilatéral de Clos respectivement non bloquant (q=2m) [2] et seulement réarrangeable (q=m).

Taille du réseau non bloquant

Le réseau de Clos à un seul niveau est le crossbar lui-même, d'où :

$$t_0 = n \; et \; c_0 = 1 \tag{2.3}$$

[2] Ne nous intéressant qu'à des ordres de grandeur, nous avons simplifié la condition 2.2 en tenant compte de ce que m est supérieur à un.

Par construction, le premier niveau de R comprend t_{k-1} crossbars, reliés chacun à $\frac{n}{3}$ processeurs (condition ci-dessus). En tenant compte de 2.3, nous obtenons :

$$t_k = \frac{n}{3} t_{k-1} = 3 \left(\frac{n}{3}\right)^{k+1} \tag{2.4}$$

Les autres niveaux sont constitués de $\frac{2n}{3}$ réseaux de Clos à k-1 niveaux, d'où, pour k>0 :

$$c_k = \frac{2n}{3} c_{k-1} + t_{k-1} = \frac{2n}{3} c_{k-1} + 3 \left(\frac{n}{3}\right)^{k} \tag{2.5}$$

La résolution des équations 2.5 et 2.3 donne alors :

$$c_k = (2^{k+2} - 3) \left(\frac{n}{3}\right)^{k} \tag{2.6}$$

d'où un nombre moyen de processeurs par crossbar de :

$$\frac{t_k}{c_k} = \frac{n}{2^{k+2} - 3} \tag{2.7}$$

Taille du réseau réarrangeable

Dans le cas du réseau réarrangeable, chaque crossbar du premier niveau est connecté à $\frac{n}{2}$ (liens vers les) processeurs et à $\frac{n}{2}$ crossbars du deuxième niveau.

Nous obtenons ainsi le nouveau jeu d'équations suivant :

$$t_k = \frac{n}{2} t_{k-1} = 2 \left(\frac{n}{2}\right)^{k+1} \tag{2.8}$$

$$c_k = \frac{n}{2} c_{k-1} + t_{k-1} = \frac{n}{2} c_{k-1} + 2 \left(\frac{n}{2}\right)^{k} \tag{2.9}$$

$$c_k = (2k + 1) \left(\frac{n}{2}\right)^{k} \tag{2.10}$$

$$\frac{t_k}{c_k} = \frac{n}{2k + 1} \tag{2.11}$$

La taille t, le nombre c de crossbars et leur rapport exprimant le prix de revient relatif d'un réseau unilatéral de Clos à k+1 niveaux sont regroupés dans les tableaux 2.1 (réseau réarrangeable) et 2.2 (réseau non bloquant).

Ces deux tableaux donnent également les résulats numériques pour soixante et quatre-vingt-dix liens, estimées correspondre aux tailles repectivement courante et maximale pour un crossbar monolithique.

k	n liens / crossbar			60 liens / crossbar			90 liens / crossbar		
	t	c	t/c	t	c	t/c	t	c	t/c
0	n	1	n	.06	.001	60	.09	.001	90
1	$\frac{n^2}{2}$	$\frac{3n}{2}$	$\frac{n}{3}$	1.8	.09	20	4.1	.14	30
2	$\frac{n^3}{4}$	$\frac{5n^2}{4}$	$\frac{n}{5}$	54	4.5	12	729	10.1	14
3	$\frac{n^4}{8}$	$\frac{7n^3}{8}$	$\frac{n}{7}$	1620	189	8.6	8201	637	12.9

Table 2.1: Taille et coût d'un réseau de Clos réarrangeable

k	n liens / crossbar			60 liens / crossbar			90 liens / crossbar		
	t	c	t/c	t	c	t/c	t	c	t/c
0	n	1	n	.06	.001	60	.09	.001	90
1	$\frac{n^2}{3}$	$\frac{5n}{3}$	$\frac{n}{5}$	1.2	0.1	12	2.7	0.15	18
2	$\frac{n^3}{9}$	$\frac{13n^2}{9}$	$\frac{n}{13}$	24	5.2	4.5	81	11.7	6.9
3	$\frac{n^4}{27}$	$\frac{29n^3}{27}$	$\frac{n}{29}$	480	232	2.1	2430	783	3.1

Table 2.2: Taille et coût d'un réseau de Clos non bloquant

Les valeurs numériques de t et c sont exprimées en millier de liens et de crossbars.

Ces tableaux méritent quelques commentaires :

- La version non bloquante est plus coûteuse : à nombre de niveaux égal, elle utilise plus de crossbars ($5n/3$ contre $3n/2$ pour k=1) pour relier moins de processeurs ($n^2/3$ contre $n^2/2$) ; l'écart allant croissant avec le nombre de niveaux.
- La version non bloquante à deux niveaux de commutation permet malgré tout d'atteindre un degré de parallélisme significatif de l'ordre

du millier de processeurs en utilisant moins d'un crossbar pour dix processeurs, ce qui reste raisonnable.

Au-delà de deux niveaux, le nombre de crossbars devient rapidement prohibitif.

Pour une proportion crossbars/processeurs comparable, la version réarrangeable autorise un niveau de plus, soit quelques dizaines de milliers de processeurs.

A taille équivalente, le nombre P de portes (coût) d'un réseau de Clos non bloquant est inférieur à celui d'un crossbar.

Ce coût est de αt^2 pour le crossbar et de $\alpha n^2 c$ pour le réseau de Clos (coût individuel multiplié par le nombre de crossbars).

Le tableau ci-dessous montre que pour n=60, un réseau de Clos à deux niveaux est déjà quatre fois plus économique qu'un crossbar et que le rapport croît rapidement avec le nombre de niveaux.

k	t	c	αP_{Clos}	$\alpha P_{Crossbar}$	Rapport	pour n = 60
0	n	1	n^2	n^2	1	1
1	$n^2/3$	$5n/3$	$5\,n^2/9$	$n^4/9$	$n/15$	4
2	$n^3/9$	$13\,n^2/9$	$13\,n^4/9$	$n^6/81$	$n^2/117$	30
3	$n^4/27$	$29\,n^3/27$	$29\,n^6/27$	$n^8/729$	$n^4/783$	16550

Table 2.3: Complexité en nombre de portes

Chapitre 3

LA FAMILLE TRANSPUTER

La famille transputer actuelle (T2,T4,T8) d'INMOS est une famille de microprocesseurs spécifiquement conçue pour la réalisation de multiprocesseurs sans mémoire commune, d'où son emploi dans les machines Supernode.

Ce chapitre en présente brièvement les principales caractéristiques, ainsi que quelques particularités plus techniques ayant influé sur la conception des machines Supernode, à savoir le fonctionnement des liens et les signaux de service.

La conception des transputers est allée de pair avec la définition du langage parallèle destiné à leur programmation, nommé OCCAM.

Du fait de la remarquable facilité d'expression du parallèlisme qu'il offre, ce langage, dérivé de C.S.P., nous servira de support de description des algorithmes de reconfiguration et fait l'objet d'une courte présentation dans le dernier paragraphe.

La vocation de la famille transputer se traduit par un certain nombre de caractéristiques originales :

- Une optimisation de l'interface mémoire et l'intégration de liaisons bidirectionnelles série pemettent d'obtenir des nœuds de calcul compacts, économiques et faciles à interconnecter.

- L'utilisation du parallélisme à l'intérieur des circuits intégrés permet un fonctionnement simultané des liens, du processeur de calcul et d'éventuelles untiés additionnelles (par exemple de calcul en virgule flottante), d'où des performances accrues.

- La gestion des processus et des communications par rendez-vous de OCCAM sont directement câblées et/ou supportées par le jeu d'instructions.

3.1 Les processeurs

Le plus connu des transputers de la famille actuelle (T2,T4,T8) est le T800 (figure 3.1), composé de :

- un processeur 32 bits inspiré de la technologie RISC,
- une unité de calcul en virgule flottante (1.5 mégaflop/s)
- une petite mémoire vive statique (4 kilooctets)
- quatre liens bidirectionnels série fonctionnant à 10 ou 20 Mégahertz
- un dispositif d'accès à une mémoire externe (jusqu'à quatre gigaoctets) intégrant l'essentiel du support pour l'utilisation de mémoire dynamique.

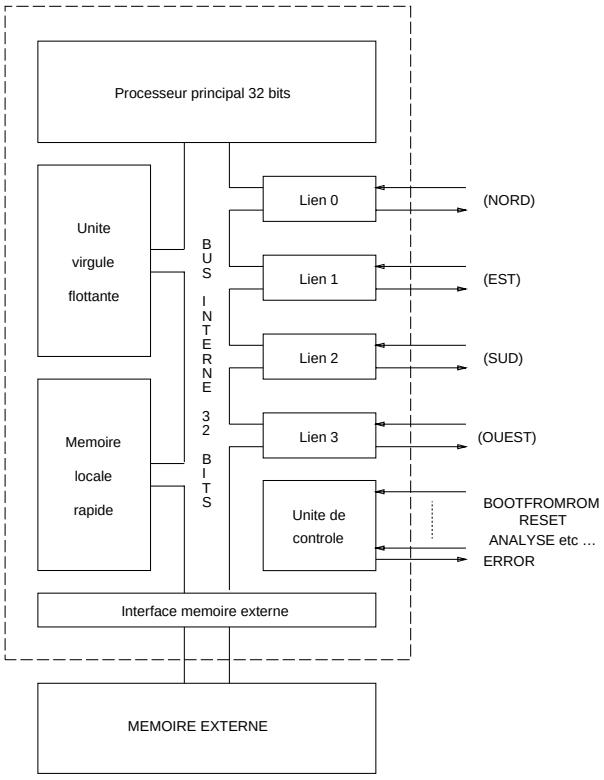


Figure 3.1: Synoptique du T800

Trois des membres de cette famille de transputers entrent dans la composition des machines Supernode :

- Les calculs proprement dits sont confiés au plus puissant d'entre eux, le T800, déjà cité.
- La supervision de la machine et la gestion des périphériques sont l'apanage du T414, transputer 32 bits dépourvu d'unité de calcul en virgule flottante.
- Les T212, qui ne travaillent que sur seize bits, remplissent seulement des fonctions marginales d'interface de commande du réseau de commutation.

3.2 Fonctionnement des liens

3.2.1 Un fonctionnement synchrone

Les transputers actuels n'offrent aucun support spécifique pour le routage des messages et l'unique primitive de communication sur les liens est le transfert de bloc mémoire entre voisins.

Chaque interface de lien est munie d'un dispositif d'accès direct (DMA) à la mémoire locale du nœud.

Une communication sur un lien est déclenchée lorsque deux processus exécutent aux deux extrémités du lien une intruction respectivement d'entrée et de sortie ; après quoi leur progression est suspendue jusqu'à la terminaison de la communication.

Tout transfert implique donc un rendez-vous préalable entre les deux processus émetteur et récepteur ; du fait de ce rendez-vous, la communication est dite de type synchrone.

Les adresses des blocs respectivement à émettre est à recevoir sont transmises aux interfaces de lien correspondantes, ainsi que le nombre d'octets à transférer. Il doit y avoir accord entre les deux processus sur la taille du message à transférer et cette propriété est vérifiée statiquement par le compilateur OCCAM.

Comme sur une ligne série asynchrone (RS232), chacun des octets émis à tour de rôle est délimité par un préfixe (deux bits à un) et une marque de fin (un bit à zéro).

Les liens se trouvent en concurence, entre eux et avec le processeur de calcul, pour l'accès à la mémore locale de chaque nœud. Un mécanisme appelé contrôle de flux doit donc être mis en œuvre afin d'éviter que les octets soient émis sur le lien plus vite que l'interface réceptrice ne peut les recopier en mémoire.

C'est pourquoi, pour chaque information élémentaire transférée (chaque octet en l'occurence), l'interface de lien réceptrice émet en retour un acquittement (un bit à un suivi d'un bit à zéro) dont la réception par l'unité émettrice conditionne l'envoi de l'octet suivant.

Le premier octet du message joue en fait le rôle d'une requête de rendez-vous émanant du processus émetteur, le premier acquittement de signal de réalisation de rendez-vous (donc de début de transfert), et le dernier de signal de fin de communication.

Des communications simultanées dans les deux sens supposeraient l'emploi d'une paire de fils pour les données et d'une autre, faiblement occupée, pour les seuls acquittements.

Pour réduire le coût de la connectique, les liens bidirectionnels du transputer ne comprennent que deux fils, sur chacun desquels sont multiplexés les données d'une voie et les acquittements de l'autre voie circulant dans

le même sens.

La séparation des données et des acquittements repose alors sur l'examen du deuxième bit de chaque séquence.

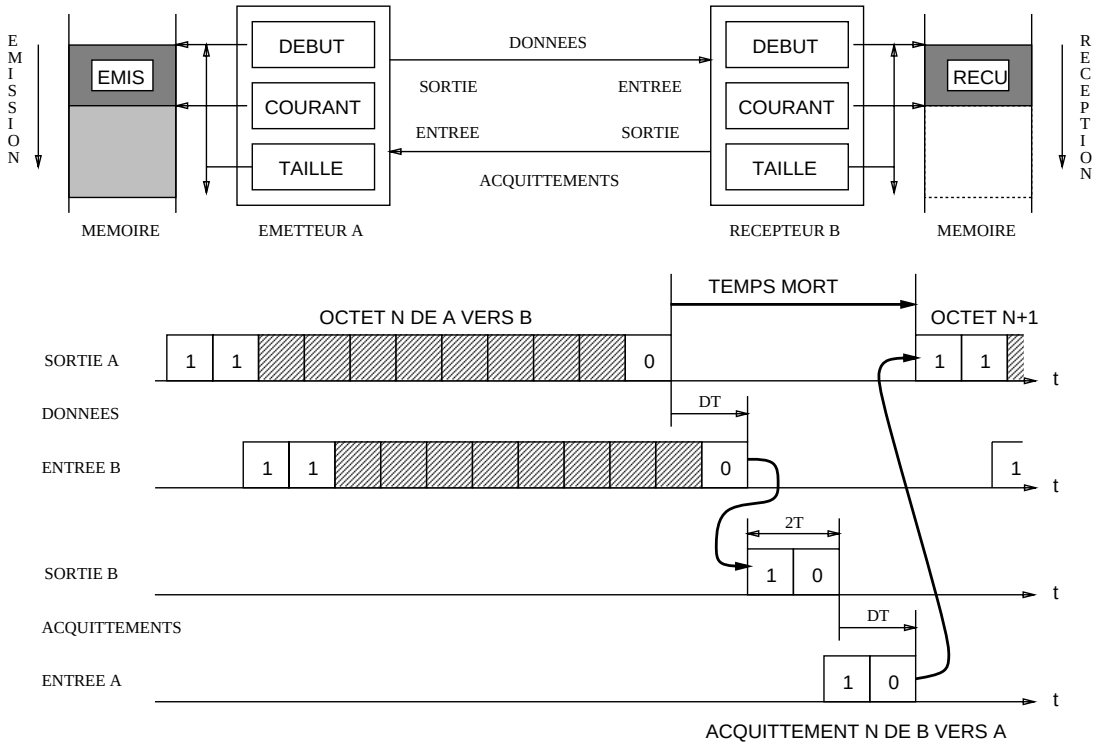


Figure 3.2: Les liens mettent en œuvre un mécanisme d'acquittement

D'autre part, à la concurence d'accès à la mémoire près, les interfaces des liens fonctionnent de manière totalement indépendante les unes des autres et il n'existe en particulier aucun mécanisme permettant d'asservir (à l'intérieur d'un même transputer) le débit d'un lien de sortie à la vitesse de réception d'un lien d'entrée.

Cette contrainte implique, dans le cas du routage, d'attendre d'avoir reçu la totalité du message ou du paquet avant d'en commencer la réémission.

3.2.2 Emission anticipée et fenêtrage

L'inconvénient du protocole ci-dessus, mis en œuvre par les premiers membres de la famille transputers, est la présence d'un temps mort entre la fin de l'émission d'un octet et le début de celui de l'octet suivant (figure 3.2).

Soit f=$\frac{1}{T}$ la fréquence de fonctionnement du lien et DT le délai de traversée du lien (temps de propagation des signaux d'un transputer à l'autre au travers des fils et des divers amplificateurs) exprimés en nombre de bits.

En l'absence de temps mort, la bande passante du lien serait $B_{\max} = \frac{1}{11T}$.

Ce temps mort, égal à la somme :

- du délai de traversée du lien (aller) par l'octet (DT),
- de la durée des deux bits de l'acquittement (2T) et
- du délai de traversée du lien (retour) par l'acquittement (DT),

se traduit par une réduction de la bande passante effective du lien :

$B = \frac{1}{11T+DT+2T+DT} = B_{\max}\frac{1}{1+\frac{2D+2}{11}}$

Il existe une parade à cette perte de bande passante effective du lien : le fenêtrage à l'émission consiste à enchaîner l'émission de l'octet (ou des n octets) suivant, sans attendre le retour de l'acquittement courant.

L'interface réceptrice doit par contre être munie des registres supplémentaires nécessaires à l'éventuel stockage des données dont le transfert a été anticipé.

La différence maximale entre le nombre de données émises et celui des acquittements reçus en retour est appelée fenêtre d'émission.

Souvent mis en œuvre dans les réseaux informatiques à commutation de paquets, le fenêtrage n'a pas été utilisé dans les transputers (la fenêtre est égale à l'unité, ce qui revient à une synchronisation stricte).

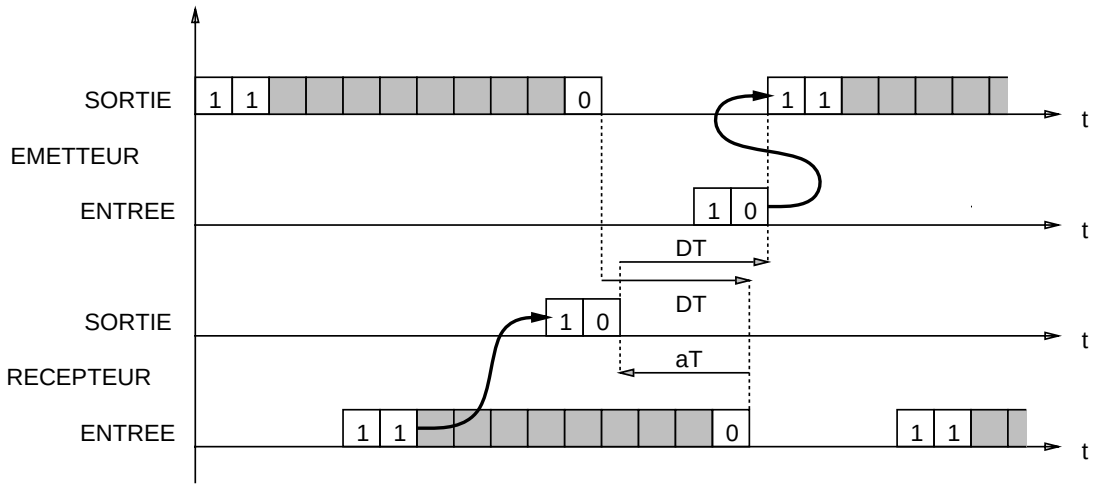


Figure 3.3: Emission anticipée de l'acquittement sur les liens du T800

Toutefois, en l'absence de gestion d'erreur (les liens étant supposés fiables), les acquittements se réduisent à de simples signaux en tout ou rien, indépendants du contenu de la valeur des octets acquittés.

Les informations à prendre en compte se limitant au type (données ou acquittement) de la séquence de bits reçue, l'envoi d'un acquittement peut

être entrepris dès la réception du deuxième bit (d'en-tête) et précéder la fin de la réception de l'octet d'un délai aT[1].

Cette anticipation permet de déclencher plus tôt le transfert de l'octet suivant et d'optimiser la bande passante réelle du lien :

$$B_{reelle} = B_{\max} \frac{1}{1+\frac{2D-a}{11}}$$

la bande passante optimale B_{max} pouvant être atteinte lorsque le délai de traversée du lien ne dépasse la durée d'un ou deux bits.

Le délai de traversée introduit par les crossbars du réseau d'interconnexion a fait l'objet d'une attention particulière lors de la conception de l'architecture Supernode pour ne pas ralentir exagérément les liens.

3.3 Fonctions de service

Le transputer est également muni de broches de commande de diverses fonctions de service, dont la voie de signalisation devra permettre la manipulation par l'unité de supervision de chaque machine.

3.3.1 Initialisation et démarrage

Après réinitialisation, le transputer peut, comme les autres microprocesseurs, exécuter un programme de démarrage stocké dans une mémoire morte.

Une telle procédure suppose cependant que chacun des processeurs d'une machine parallèle sans mémoire commune soit muni de sa propre mémoire morte contenant une copie du code de démarrage, ce qui nuit à la compacité des nœuds et en augmente le coût.

C'est pourquoi les transputers supportent également une procédure de démarrage par les liens, en deux étapes :

- La séquence de code initiale, d'une longueur (encodée dans le premier octet) limitée à 256 octets est tout d'abord téléchargée via l'un des liens par l'un des voisins.

- Le chargement et l'activation du code de l'application (ou d'une amorce auxiliaire de taille quelconque) ne sont effectués que dans un deuxième temps par cette première amorce, qui réalise également diverses initialisations complémentaires.

[1]La valeur de a, non fournie par le constructeur, est été estimée à environ quatre périodes. Je n'ai pas entrepris de le vérifier par des mesures précises et cette valeur est indiquée sous toutes réserves.

Le stockage de l'amorce dans une mémoire morte n'est alors nécessaire que sur un seul nœud (la racine) qui initialisera de proche en proche l'ensemble des processeurs via le réseu d'interconnexion.

Au cours de la réinitialisation du transputer (broche Reset), la broche Analyse permet de forcer une sauvegarde partielle de l'état du processeur à des fins de mise au point et le choix du mode de démarrage est fixé via la broche BootFromRom.

Nous envisagerons les répercussions de ce protocole d'amorçage pour l'installation d'un noyau de routage dans le chapitre 5.7.2.

3.3.2 Gestion des erreurs

La gestion des erreurs (division par zéro, instruction invalide, etc ...) par le transputer est des plus rudimentaires.

Celle-ci se limite en effet à la mise à jour d'un unique booléen d'erreur, dont le contenu est réflété par la broche Error du transputer ; et ce à l'exclusion de tout mécanisme de déroutement.

La mémoire externe des nœuds de calcul des machines Supernode est d'autre part munies d'un dispositif de vérification de parité.

3.3.3 Autres signaux

Le transputer possède une seule source d'interruption appelée "évènement" (signaux Event Req et Event Ack), qui, du point de vue de la programmation, est traitée comme un pseudo-canal OCCAM, sur lequel aucun message n'est transféré.

Les communications et synchronisations sur la voie de signalisation font un usage intensif de cette source d'interruption.

Les autres signaux annexes sont essentiellement destinés à paramétrer l'interface d'accès à la mémoire externe et la vitesse des liens.

3.4 Autres composants

Les transputers sont acompagnés de divers composants destinés à faciliter la construction de multiprocesseurs reconfigurables.

La connexion des transputers avec leur environnement est souvent réalisée par un adaptateur de lien (C012). Celui-ci permet en effet de piloter un lien via le bus mémoire de n'importe quelle machine, telle qu'une station de travail pour le dialogue avec l'utilisateur. Il permet également de rajouter un lien à un transputer dont les quatres premiers sont déjà occupés.

Il existe également un crossbar à trente-deux liens, le C004, contrôlé via un lien supplémentaire qui en constitue l'unique interface de commande.

En se propageant le long des câbles et en traversant des divers amplificateurs, les signaux électriques subissent diverses dégradations, dont une modification de la largeur des créneaux. C'est pourquoi les sorties du C004 sont munies d'un dispositif de remise en forme pour éviter que des dégradations cumulatives n'entraînent des erreurs de transmission sur les liens.

Le recalibrage effectué par le C004 est particulièrement utile lorsque les distances à parcourir dépassent le mètre (cas des liaisons entre les deux niveaux dans les machines supernode); mais il présente l'inconvénient d'introduire un retard de traversée d'environ deux bits.

Les projet Supernode a da'utre part donné lieu à la réalisation d'un crossbar spécifique "supernode" à 72 liens, sans recalibrage et muni d'une interface de commande parallèle. Nous l'appelerons quelquefois crossbar RSRE, du nom du partenaire qui en a assuré la conception.

3.5 Le futur

INMOS a récemment annoncé le développement d'une nouvelle famille de transputers offrant notamment les améliorations suivantes :

- une vitesse de calcul accrue, le futur processeur H1/T9000 étant crédité d'une puissance de 10 à 20 Mflops/s,
- une fréquence maximale de fonctionnement des liens portée à 100 Mhertz,
- l'acheminement des messages est directement supporté par un routeur câblé et physiquement séparé du processeur (routeur C104) et
- un meilleur support des primitives nécessaires à la réalisation d'un système d"exploitation.

3.6 Le langage OCCAM

En OCCAM, comme en C.S.P., les communications entre les processus sont de type bipoint et synchrones, ce dernier terme indiquant que toute communication implique une synchronisation préalable (par rendez-vous) entre les deux correspondants avant tout transfert de message.

Le nommage direct des correspondants de C.S.P. a cependant été remplacé par une désignation indirecte au travers d'un nouvel objet du lanage, le canal, qui établit une liaison unidirectionnelle de l'émetteur vers le récepteur.

Les instructions OCCAM de base sont les suivantes :

- l'affectation d'une valeur à une variable (variable := expression),
- la sortie ou émission d'un message sur un canal (canal ! message),
- l'entrée ou réception d'un message dans une variable (canall ? variable)
- SKIP, qui équivaut à une instruction vide
- STOP, instruction qui stoppe l'exécution (modélisation des cas d'erreur).

Les instructions composées sont formées à partir des constructeurs du langage : SEQ, PAR, IF WHILE, ALT ; dont la portée est délimitée par le "niveau d'indentation" ou nombre d'espaces en début de ligne.

Le constructeur SEQ réalise l'enchaînement de l'exécution des instructions, implicite dans les langages séquentiels :

```
SEQ
  instruction_1
  instruction_2
   ...
  instruction_n
```

Le constructeur PAR correspond au contraire à l'exécution en parallèle des différentes instructions avec synchronisation du type "fork/join".

```
PAR
  instruction_1
  instruction_2
   ...
  instruction_n
```

De même, le constructeur itératif WHILE a la même signification que dans les langages séquentiels :

```
WHILE
  condition
    instruction_n
```

Le constructeur conditionnel IF, semblable à celui de LISP, exécute la première instruction dont la condition qui précède est vraie :

```
IF
  condition_1
    instruction_n
  condition_2
    instruction_2
   ...
  condition_n
    instruction_n
```

Dans l'alternative, une instruction d'entrée est ajoutée à la condition locale telle qu'on la trouve dans un IF ; l'ensemble étant appelé une garde.

```
ALT
  condition_1 & canal_1 ? variable_1
    instruction_1
  condition_2 & canal_2 ? variable_2
    instruction_2
   ...
  condition_n & canal_n ? variable_n
    instruction_n
```

Une garde est dite passante lorsqu'à la fois :

- la condition locale est vérifiée et
- qu'un rendez-vous a été établi avec un émetteur sur le canal.

L'exécution de l'alternative est suspendue jusqu'à ce qu'une des gardes devienne passante ; la communication sur le canal est alors effectuée, et l'instruction correspondante exécutée.

Lorsque plusieurs gardes sont simultanément passantes, l'une d'entre elles est sélectionnée de manière non déterministe.

Dans ce contexte, SKIP est une garde toujours passante, qui ne génère aucune communication.

Chapitre 4

L'ARCHITECTURE SUPERNODE

L'objectif de ce chapitre est de donner une vue d'ensemble de l'architecture Supernode en insistant sur les bases théoriques sur lesquelles celle-ci repose.

Certains passage des chapitres 5 et 6 relatifs à l'évaluation des performances de la reconfiguration exigent une connaissance relativement précise des machines Supernode qui ont servi de support d'évaluation.

Cette partie plus technique a été reportée en annexe : l'annexe B décrit en détail le réseau de commutation de ces machines et le fonctionnement du bus de contrôle est traité à part dans l'annexe C.

4.1 Objectifs et contraintes

Le système d'interconnexion des liens a été conçu en fonction des objectifs suivants :

- taille : les performances visées (de l'ordre du Gigaflop/seconde) imposent environ mille processeurs, soit quatre mille liens à gérer.
- réarrangeabilité : tout graphe de connexion de transputers doit être réalisable. La machine peut être utilisée comme support de développement d'une application avant transfert sur un réseau dédié, qu'il est donc souhaitable de pouvoir reproduire exactement. Il est aussi possible d'expérimenter plusieurs réseaux différents pour la même application.
- dynamicité : la possibilité de changer le graphe de connexion est souhaitable, et ce aussi bien dynamiquement, c'est-à-dire en cours d'exécution du programme, que préalablement à celle-ci. Il faut donc minimiser le délai entre la prise de décision et la réalisation de la

reconfiguration. Cela suppose des commutateurs élémentaires rapides et un algorithme de pilotage efficace, donc de faible complexité.

- transparence : l'introduction du réseau d'interconnexion sur un lien entre deux processeurs ne doit pas en réduire le débit. Le délai de traversée du réseau devra rester faible de façon à conserver les avantages de l'anticipation d'acquittement.
- extensibilité et homogénéité : en accord avec l'approche de parallélisme massif, le réseau d'interconnexion doit permettre la construction d'une gamme homogène de machines de différentes tailles présentant un rapport prix/performances aussi constant que possible. Nous verrons que cet objectif a été atteint par une approche modulaire.

La réalisation d'un unique crossbar gérant les quatre mille liens des transputers étant bien évidemment totalement hors de question, le réseau d'interconnexion a été défini en deux étapes.

- La première étape consiste à réduire la taille du problème. Le domaine des connexions légales entre les liens des processeurs a été réduit tout en conservant la possibitlité de réaliser le même ensemble de graphes de degré quatre. La taille du réseau a ainsi été réduite d'un facteur quatre.

  En contrepartie, certaines manières d'étiqueter les arc des graphes de connexion ne sont plus autorisées et une opération de renommage des liens devient nécessaire pour transformer ce graphe en un graphe légal équivalent réalisable par les machines.
- La seconde étape substitue au crossbar un réseau de Clos à plusieurs étages, au prix d'une plus grande complexité de pilotage.

4.2 Réduction de la taille du problème

Considérons un réseau connexe quelconque de transputers :

- Relions les liens libres (c'est-à-dire non connectés ou reliés à l'extérieur de la machine) entre eux de façon à obtenir un réseau complètement connecté.
- Considérons le graphe de degré quatre associé à ce réseau en ignorant les numéros des liens des transputers.

Intéressons-nous aux propriétés topologiques de ce graphe : tous les sommets sont de degré quatre, donc de degré pair.

D'après la théorie des graphes, il est donc possible de trouver un cycle eulérien, c'est-à-dire passant une et une seule fois par chacune des arêtes, comme le montre la figure 4.1.

Après numérotation des arcs le long du cycle, il est possible d'extraire deux sous-graphes comprenant chacun les arêtes respectivement paires et impaires et l'ensemble des sommets. Par construction, tous les sommets de ces sous-graphes sont de degré pair et chaque sous-graphe comprend un ou plusieurs cycles eulériens dont les arcs forment deux ensembles disjoints.

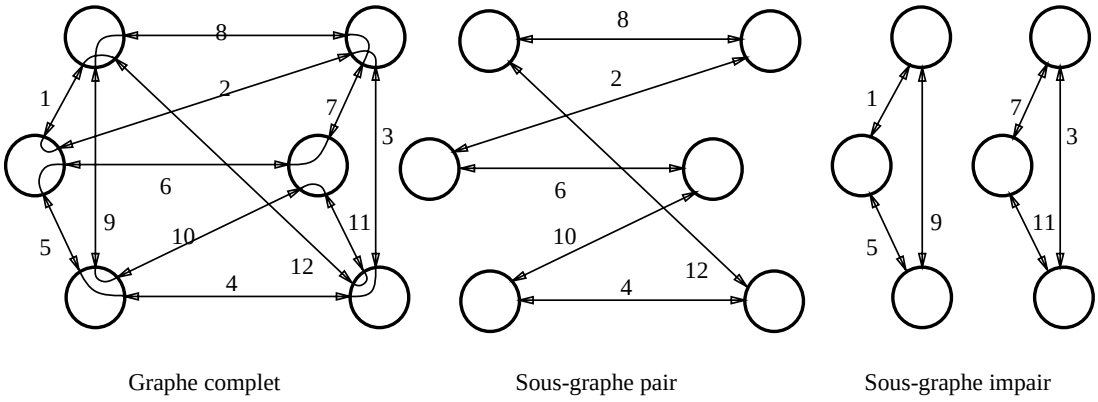


Figure 4.1: Tout graphe de degré pair admet un cycle eulérien

Ceci montre qu'il est possible, moyennant un renommage des liens physiques des processeurs, de scinder le crossbar initial à 4n liens en deux crossbars à 2n liens traitant chacun deux liens de chacun des n processeurs, tout en conservant le même éventail de topologies réalisables.

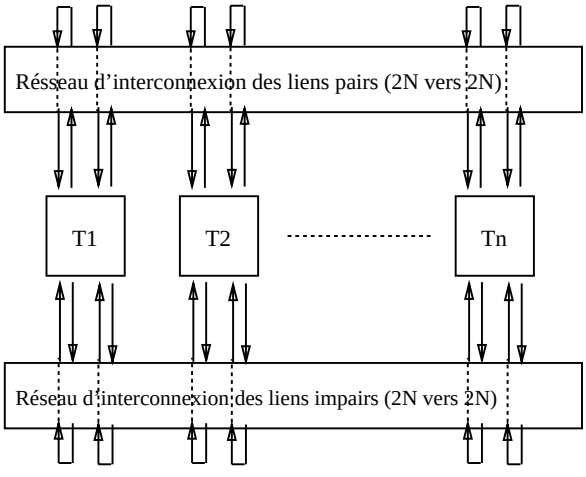


Figure 4.2: Un réseau à 2 crossbars

Il est donc possible de réaliser tout graphe d'interconnexion de degré quatre avec deux crossbars 2n entrées vers 2n sorties au lieu d'un crossbar unique 4n vers 4n (figure 4.2).

Les liens du sous-graphe pair (respectivement impair) seront alors regroupés sur un même crossbar dit pair (respectivement impair).

Par contre, si l'on tient compte des numéros des liens physiques des transputers, un réseau quelconque ne sera pas réalisable tel quel et devra être remplacé par un renommage des liens en un réseau équivalent respectant la règle de parité des connexions.

La méthode est-elle réapplicable ? Considérons le sous-graphe pair obtenu précédemment : il est effectivement possible de le décomposer à nou-

veau et de la même façon en deux nouveaux sous-graphes (figure 4.3).

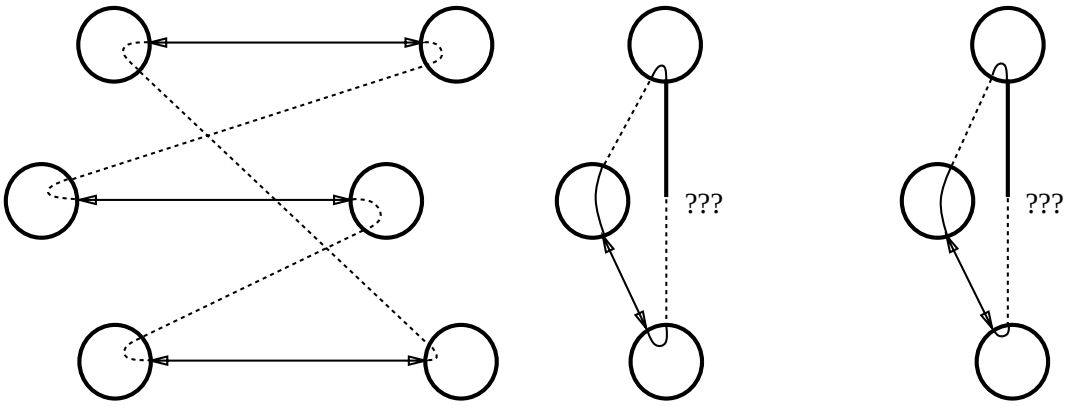


Figure 4.3: Une nouvelle décomposition pose problème

Si nous considérons maintenant l'un des cycles de l'autre sous-graphe, nous voyons que la méthode n'est pas réapplicable lorsque le nombre de sommets est impair. En effet, cela implique alors un nombre impair de liens et il se pose un problème pour le dernier lien.

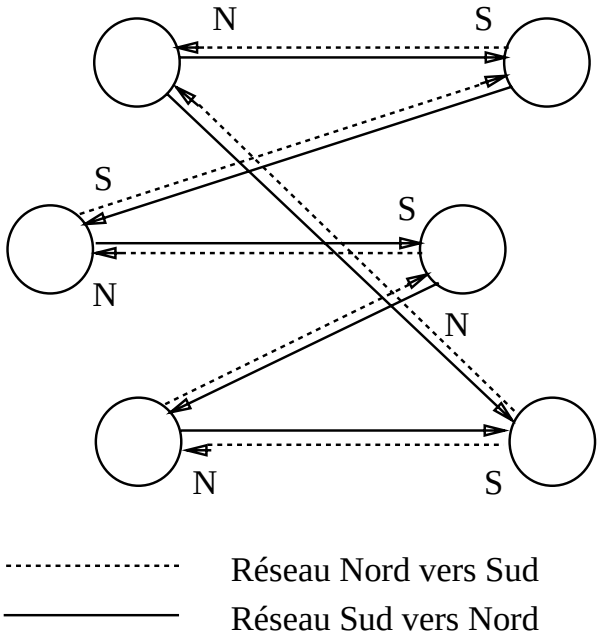


Figure 4.4: Séparation de la voie aller et de la voie retour des liens

Toutefois, les liens des transputers étant bidirectionnels (c'est-à-dire constitués de deux fils distincts), il est malgré tout possible de gagner encore un facteur deux.

En effet, parcourons le cycle en étiquetant chaque extrémité de lien alternativement par exemple Nord et Sud (figure 4.4). L'ensemble des connexions à réaliser se décompose alors en deux sous-ensembles disjoints : sorties Nord vers entrées Sud (traits pleins) et sorties Sud vers entrées Nord (pointillés) [17] [20] [18].

Chaque crossbar à 2n liens peut donc être à nouveau scindé en deux crossbars à n liens, traitant chacun une moitié des liens de tous les trans-

puters. Les liens du sous-graphe pair seront ainsi étiquettés Nord et Sud, et ceux du sous-graphe impair Est et Ouest.

Nous obtenons donc la structure finale de la figure 4.5, commune à l'ensemble des machines de la gamme Supernode.

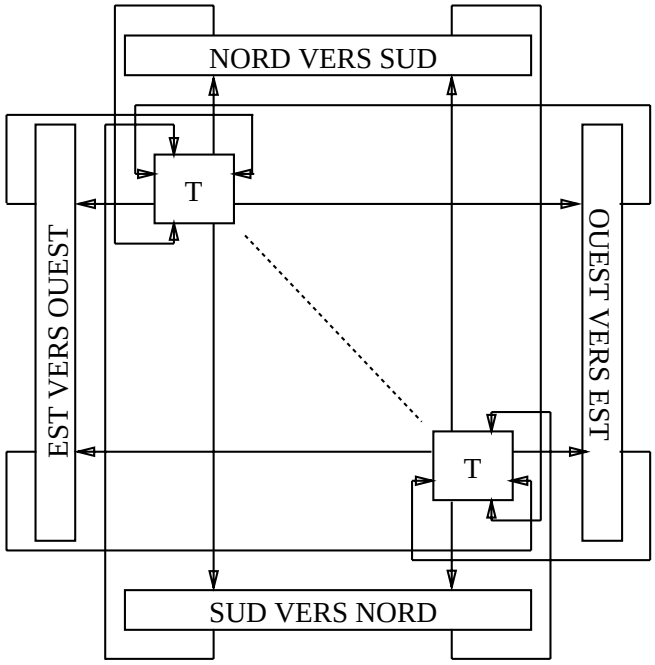


Figure 4.5: Le réseau se décompose en quatre sous-réseaux

Nous pouvons dresser un bilan de la décomposition :

Le remplacement du crossbar à 4n entrées vers 4n sorties par quatre crossbars n entrées vers n sorties a permis de gagner un facteur quatre sur la taille du réseau. Un précalcul est maintenant nécessaire : il consiste à chercher un cycle eulérien et à visiter chacun de ses sommets deux fois (étiquettage pair/impair puis Nord/Sud et Est/Ouest).

D'autre part, l'entrée et la sortie d'un même lien sont connectées à des circuits différents. Il est donc devenu impossible de les reboucler l'une sur l'autre de façon à isoler le lien. On ne pourra isoler des liens d'un même processeur qu'en les rebouclant par paires Nord-Sud ou Est-Ouest.

4.3 Les réseaux multi-étages

L'étape suivante consiste à remplacer chaque crossbar par un réseau à plusieurs étages.

La recherche d'un délai de traversée minimal a conduit à l'adoption d'un réseau unilatéral de Clos à deux niveaux.

En effet, dans les cas favorables (entrée et sortie reliées à un même crossbar du premier niveau), celui-ci n'introduit pas de délai supplémentaire par rapport au crossbar.

Les réseaux à plusieurs étages ont par contre pour inconvénient une complexité accrue de pilotage, liée à la recherche des chemins à emprunter pour réaliser chacune des connexions.

La possibilité de connexions locales permet également de réduire d'autant le nombre de chemins (au travers du deuxième étage) à calculer.

La réduction des coûts de réalisation a été privilégiée au détriment de la dynamicité. Le réseau de Clos des machines Supernode est donc réarrangeable, mais bloquant (p=q=32) ; la taille des crossbars des deux niveaux étant respectivement de 64 et 32 liens. L'objectif d'un réseau de permutation de taille 1024 est donc bien atteint.

La figure 4.6 illustre la structure du réseau de commutation des liens Nord et Sud dans une machine à deux niveaux. La moitié supérieure de la figure représente le réseau de commutation Nord vers Sud. En haut de la figure se trouvent les q crossbars du deuxième niveau. Ensuite viennent les p crossbars du premier niveau. Enfin, au centre de la figure, quatre des p fois q transputers ont été représentés. On retrouve par symétrie les mêmes éléments dans le réseau de commutation Sud vers Nord représenté dans la moitié inférieure de la figure.

4.4 Construction modulaire des machines

L'architecture modulaire des machines Supernode repose sur la propriété de localité du réseau unilatéral de Clos.

En effet, considérons un module composé d'une double paire (Nord-Sud et Est-Ouest) de crossbars de même rang du premier niveau et les transputers qui leur sont connectés (cadre en pointillé sur la figure précédente). Ce module possède une structure identique à celle de la figure 4.5 et il est capable de gérer localement toutes les connexions entre les transputers qui lui appartiennent. Ce module n'est donc rien d'autre qu'une machine Supernode de taille plus petite.

La gamme Supernode comprend donc deux types de machines : les machines de petite taille organisées autour d'un unique niveau de commutation de liens et les machines hiérarchisées construites en interconnectant des petites machines par un module spécial regroupant les crossbars du deuxième niveau du réseau.

Le nombre de liens de ces crossbars (donc le nombre maximum de modules) est de 32, d'où la nécessité de modules acceptant 32 processeurs pour construire une machine de taille 1024.

La taille des crossbars d'un module de base doit être au minimum égale au double du nombre de processeurs et c'est à partir de ce cahier des

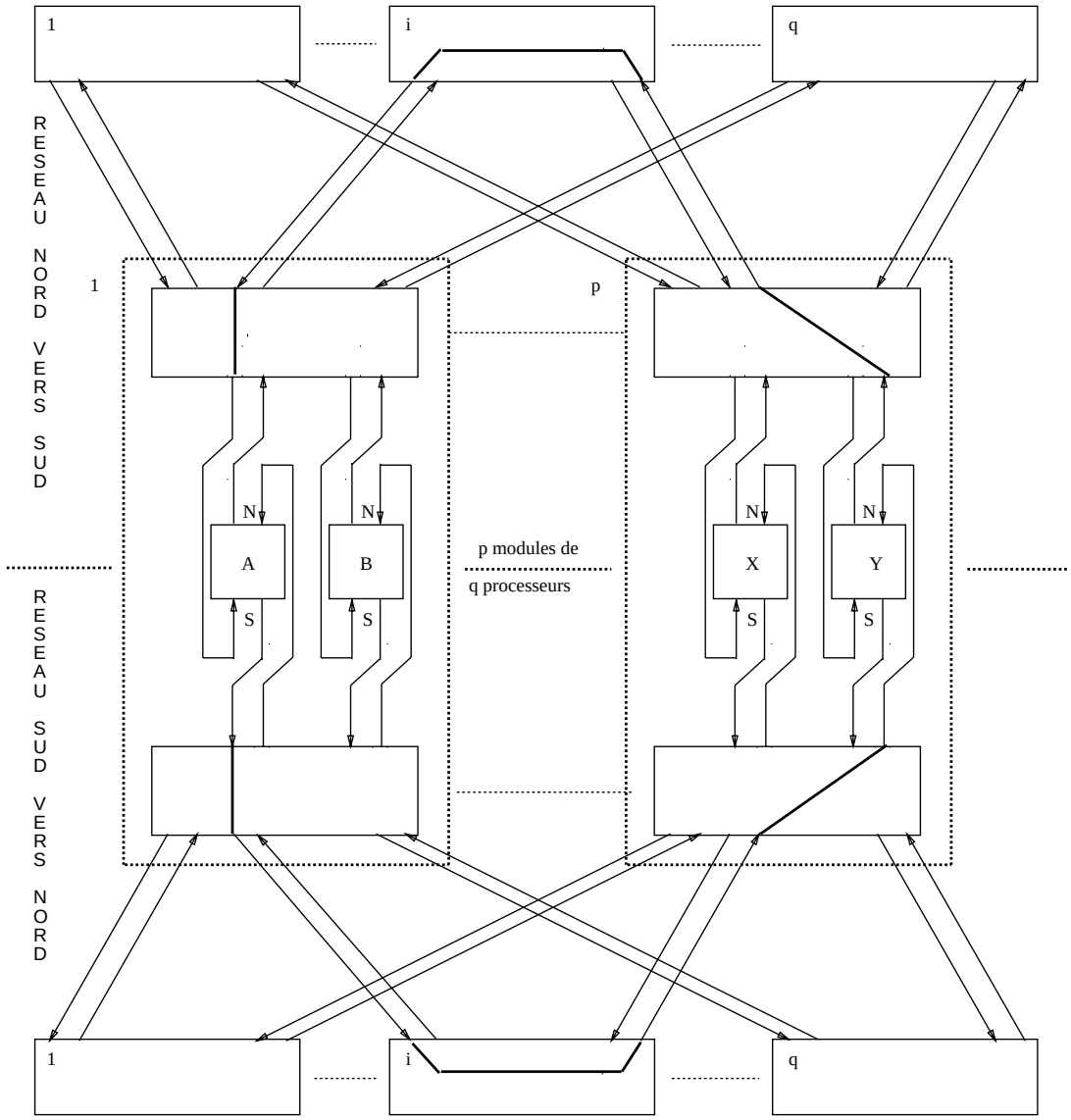


Figure 4.6: Les réseaux multi-étages nord et sud

charges que le crossbar spéficique Supernode a été développé dans le cadre du projet.

4.5 Les modules de base

Schématiquement, un module de base autonome se compose d'un certain nombre de transputers de travail (64 au maximum) interconnectés par deux paires (Nord-Sud et Est-Ouest) de crossbars Supernode.

Dans un module de base, les distances sont courtes et un seul étage doit être traversé. Le crossbar "Supernode" n'effectue donc pas de recalibrage des signaux, ce qui permet de réduire le délai de traversée.

L'ensemble du module est géré par un transputer contrôleur qui commande ces crossbars via son bus mémoire externe.

Ce contrôleur peut également accéder à une voie de signalisation appelée bus de contrôle, dont il est le maître, et via laquelle il peut piloter les fonctions de service des transputers de travail.

Le dialogue contrôleur-transputers de travail engendré par la reconfiguration dynamique (synchrone ou asynchrone) est également supporté par ce bus de contrôle qui offre pour cela deux primitives de communication :

- le transfert d'un octet de l'interface d'un esclave donné vers celle du maître. Ce transfert est réalisé via un cyle de lecture sur le bus.
- la diffusion d'un octet de l'interface maître vers les interfaces d'une grappe d'esclaves. Cette diffusion est obtenue par un cycle d'écriture sur le bus.

et deux primitives de synchronisation :

- la requête OU, qui permet à un esclave quelconque d'un ensemble donné d'émettre un signal de requête vers le maître. Le maître a la possibilité, pour acquitter cette requête, de générer en retour un signal vers cet esclave.
- l'acquittement ET, qui permet au maître de diffuser un signal de requête vers un ensemble donné d'esclaves. Il recevra alors en retour un signal lorsque tous les esclaves de l'ensemble auront acquitté cette requête.

4.6 Les machines hiérarchisées

Le nombre maximal de transputers de travail des modules de base intégrés dans une machine hiérarchisée est réduit de moitié.

Grâce aux liens ainsi libérés, ces modules de bases sont interconnectés au travers d'un module de commutation additionnel spécifique, qui regroupe l'ensemble des crossbars du deuxième niveau de commutation des réseaux de Clos (figure 4.7).

Compte-tenu du nombre de circuits traversés et de la distance à parcourir du fait de l'encombrement de la machine, les liens doivent obligatoirement être amplifiés et resynchronisés pour traverser le deuxième étage, d'où le choix de crossbars C004 qui recalibrent les signaux (mais introduisent un retard).

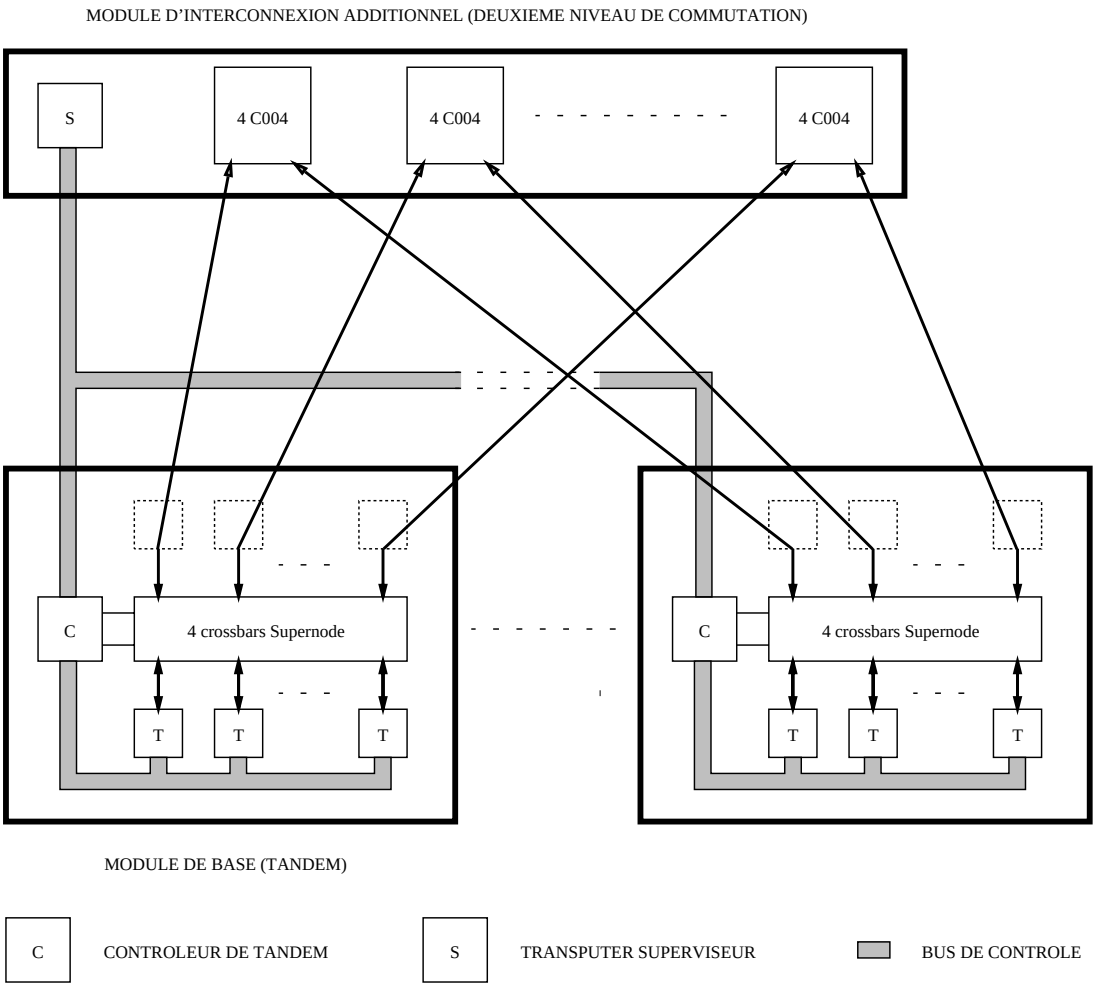


Figure 4.7: Vue d'ensemble d'une machine hiérachisée

L'ensemble est coordonné par un transputer superviseur qui pilote le deuxième niveau de commutation.

Les contrôleurs de modules sont à leur tour connectés en tant qu'esclaves sur un bus de contrôle de niveau supérieur dont le transputer superviseur est le maître. Ce bus inter-modules est en tous points comparable aux bus internes des modules de base.

4.7 La gamme Supernode

L'architecture modulaire conçue dans le cadre du projet a donné naissance à une gamme de machines commercialisées par TELMAT, ainsi que THRON EMI, via sa filiale PARSYS :

- Le point d'entré dans la gamme est une version èconomique de module de base autonome appelée nœud simple, qui accepte 32 processeurs de calcul.
- Le module de base exploitant toutes les possibilités offertes par les crossbars RSRE est le tandem. Dans sa version autonome, un tandem peut comporter jusqu"à 64 transputers de travail.
- A l'autre extrémité de la gamme, on trouve les machines hié rarchisées àdeux niveaux de commutation, construites par assemblage de tandems ; seules capables de dépasser la centaine de mégaflops par seconde.

La plus grosse configuration de machine effectivement construite comporte 256 transputers de travail, soit le quart de la taille maximale permise par les crossbars.

Les machines effectivement construites sont présentées en détail dans les annexes B et C.

Mentionnons pour terminer trois points de divergence par rapport à l'architecture générique présentée ci-dessus :

- Un nœud simple ne possède qu'une paire de crossbars Supernode.
- La commande des crossbars Supernode d'un tandem est répartie entre deux transputers contrôleurs.
- Comme les modules de base, le module de commutation de deuxième niveau possède un bus de contrôle local et son propre contrôleur. Ce dernier est connecté en temps qu'esclave sur le bus de contrôle inter-modules.

Chapitre 5

RECONFIGURATION SYNCHRONE ET ROUTAGE

La reconfiguration synchrone statique permet d'optimiser la topologie du réseau d'interconnexion : l'exécution de chaque application est précédée d'une programmation de l'ensemble du réseau, en une seule fois, selon la topologie la plus appropriée.

En cours d'exécution, comme pour une machine à connectique fixe, les messages ne bénéficiant pas d'une connexion directe entre source et destination sont acheminés par routage de nœud en nœud.

Le fil conducteur de ce chapitre est donc la conception d'un routeur pour multiprocesseurs reconfigurables.

Dans les réseaux informatiques, la lenteur et le manque de fiabilité des liaisons entre les nœuds justifient le recours à des stratégies élaborées de gestion des liens et de récupération des erreurs.

Dans un multiprocesseur, au contraire, la vitesse d'exécution de l'algorithme de routage peut s'avérer une limite plus sévère que la bande passante des liens.

De plus, l'algorithme de routage utilise le processeur et la mémoire locale du nœud sur lequel il s'exécute ; et ce au détriment des tâches de l'utilisateur confiées à ce nœud.

Le contexte d'utilisation exige donc une nouvelle approche du routage basée sur la simplification des algorithmes pour en accélérer l'exécution.

Résumons les contraintes imposées à l'algorithme de routage. Celui-ci doit :

- délivrer tout message en un temps fini à (l'unité de gestion des protocoles de) sa destination, supposée le consommer en un temps fini également ;
- accepter toutes les topologies à la disposition du programmeur, qui peuvent être irrégulières et

- s'adapter à toutes les tailles de machines, donc s'exécuter à espace mémoire et temps de calcul constants.

La simplification des algorithmes permet de plus d'envisager la construction de routeurs entièrement câblés et de réduire le délai de traversée d'un nœud d'un ordre de grandeur.

La première partie du chapitre décrit la construction "naïve" d'un tel routeur, en faisant abstraction des phénomènes d'interblocage. Ce routeur accepte cependant tous les graphes d'interconnexion acycliques.

Le cœur du chapitre concerne la prévention de l'interblocage en présence de topologies irrégulières, qui est la principale difficulté posée par la reconfiguration synchrone.

Le paragraphe 5.3 présente deux techniques originales de prévention conçues au L.G.I. pour les besoins du projet Supernode.

La dernière partie du chapitre est plus dépendante des caractèristiques des machines Supernode.

Le paragraphe 5.7 décrit les modifications apportées au routeur pour tenir compte des particularités des transputers.

Les applications construites sous la la forme d"une séquence (d'ètapes de calcul) sont du ressort de la reconfiguration dynamique synchrone (paragraphe 1.6).

Une transition entre deux étapes de calcul est appelée point de reconfiguration et donne lieu à une reprogrammation du réseau d'interconnexion.

La gestion des transitions d'étapes fait l'objet du paragraphe 5.8, qui termine ce chapitre.

5.1 Construction d'un routeur

5.1.1 Fonction de routage tabulée

Rôle : La fonction de routage détermine les chemins empruntés par les messages et doit en minimiser la longueur (la longueur à considérer devrait cependant être pondérée par le taux d'utilisation des liens pour tenir compte des phénomènes de congestion locale dans le réseau).

De plus, les chemins générés par la fonction de routage seront acycliques pour éviter qu'un message ne circule indéfiniment dans le réseau.

Le long d'un chemin, chaque processeur intermédiaire recevant un message sur l'un de ses liens d'entrée invoque cette fonction de routage qui retourne le numéro du lien de sortie à utiliser pour réémettre le message.

La fonction de routage peut être déterminisme, c'est-à-dire n'autoriser qu'un seul lien de sortie à chaque appel. Le déterministe a l'avantage d'alléger l'algorithme de routage et l'inconvénient de ne tenir aucun

compte de la répartition géographique réelle des flux de communication au cours de l'exécution pour corriger le choix initial des chemins.

La fonction peut aussi autoriser plusieurs liens, dont l'algorithme peut tenir compte de l'état réel d'occupation, au prix d'un coût de mise en œuvre plus élevé. Nous parlerons alors de routage adaptatif. Nous reviendrons sur les mérites respectifs du routage déterministe et du routage adaptatif dans le chapitre 6.

La régularité des topologies utilisées dans les machines à connectique fixe se traduit généralement par une fonction de routage suffisamment simple pour être calculée localement sur chaque processeur au cours de l'exécution.

Dans l'hypercube, par exemple, le lien de sortie à emprunter peut être déterminé en comparant bit à bit l'adresse locale du nœud à la destination du message.

Il n'en va plus de même pour des connectiques irrégulières pour lesquelles une vue de l'ensemble du réseau est indispensable. La fonction est alors précalculée et encodée dans des tables : les tables de routage.

La fonction de routage devrait normalement tenir compte de l'identité du processeur qui l'invoque et de la destination et de l'origine du message à acheminer.

Dans ce cas de figure, les calculs des chemins sont indépendants les uns de autres et les flux de communication peuvent être traités de manière individuelle pour l'équilibrage statique (avant exécution) de l'occupation des liens.

En dépit de ces avantages, la source du message n'est généralement pas prise en compte pour que la taille des tables reste proportionnelle au nombre de processeurs et non plus du carré de celui-ci : le rapport est de un mégaoctet contre un kilooctet pour les mille processeurs de la plus grosse configuration de machine Supernode.

Ce choix a toutefois un inconvénient : une même entrée d'une table étant partagée par plusieurs chemins, il convient de prendre des précautions dans le calcul des tables pour éviter de générer des chemins cycliques.

Lors de la lutte contre l'interblocage, nous adopterons un compromis consistant à ne tenir compte que du lien d'entrée sur lequel a été reçu le message dans le nœud courant et non plus de sa source.

L'équilibrage statique de la charge des liens, susceptible d'entrer ultérieurement en conflit avec la prévention de l'interblocage, n'a pas été envisagé dans cette première étape, d'où l'adoption systématique des plus courts chemins dans la fonction de routage, dont le principe même garantit également l'absence de cycle dans les chemins.

Les tables peuvent être calculées séquentiellement sur un processeur unique et installées ensuite sur les processeurs (voir paragraphe 5.7.2). Le calcul est toutefois parallélisable sur la machine cible elle-même, suppri-

mant du même coup l'étape de chargement des tables (annexe A).

5.1.2 Contrôle de flux

Entre deux processeurs voisins, le routeur doit assurer un contrôle du flux de l'émetteur pour éviter que le débit en émission n'excède la cadence de transfert que peut supporter le récepteur, ce qui provoquerait des erreurs de transmission.

Le routeur doit donc gérer un mécanisme d'acquittement, avec ou sans fenêtrage à l'émission, dont le principe a déjà été présenté au chapitre 3.2.1.

5.1.3 Famine

Sur un nœud donné, plusieurs messages reçus sur les liens d'entrée peuvent entrer en compétition pour l'accès à un même lien de sortie. Il convient alors d'éviter qu'un message n'attende indéfiniment le lien de sortie dont il a besoin pour poursuivre sa progression.

En fonctionnement normalement, tout message occupant un lien de sortie atteindra sa destination et libérera le lien au bout d'un temps fini. Il suffit donc d'utiliser un algorithme d'allocation garantissant une forme minimale d'équité pour éviter toute famine.

5.1.4 Les techniques de routage

Le mécanisme de transmission utilisé permet de regrouper les techniques d'acheminement de routage en deux grandes familles : les méthodes basées sur la recopie des messages et celles qui privilégient au contraire la construction préalable de l'ensemble du chemin à emprunter.

"Store-and-forward" et commutation de paquets

Considérons un message traversant un nœud du réseau pour rejoindre sa destination.

Lorsqu'il n'est pas possible de coupler directement le lien d'entrée et le lien de sortie, la traversée du nœud doit être réalisée par recopie en deux étapes :

- réception de la totalité du message dans un tampon, puis
- réémission du message à partir du tampon.

Cette technique, connue sous le nom de "store-and-forward" (stocker et réexpédier), réalise en fait une "commutation de messages". Elle se distingue malheureusement par un délai de traversée du nœud égal au temps de transmission du message entre deux voisins.

Pour réduire la taille des tampons et le coût associé à la gestion de messages de taille variable, les (gros) messages sont généralement découpés en (petits) messages limités à une taille fixe, routés indépendamment les uns des autres : les paquets. On parle alors de commutation de paquets.

Le délai de traversée se ramène alors à la durée de transmission d'un paquet, d'où l'intérêt d'utiliser des paquets les plus petits possibles. Cette approche est toutefois rapidement limitée par le surcoût lié à l'ajout à chaque paquet des informations nécessaires à son routage.

"Wormhole" et commutation de circuit

Nous parlerons au contraire de commutation de circuit lorsque :

- le lien d'entrée peut être relié directement au lien de sortie,
- cette liaison est empruntée par la totalité du message en respectant l'ordre d'émission
- la traversée de cette liaison n'introduit pas de délai dans la progression du message.

Pour n'introduire réellement aucun délai, la connexion du lien de sortie au lien d'entrée devrait être effectuée avant l'arrivée du message, et seule la reconfiguration dynamique asynchrone pourrait prétendre au titre de commutation de circuit.

Le "wormhole" est l'adaptation du principe de commutation de circuit au routage. Il consiste, contrairement au "store-and-forward", à limiter le stockage du message à la seule information nécessaire à l'établissement de la connexion, c'est-à-dire la destination du message. La réémission du message est entreprise dès que la connexion est réalisée.

Le message se propage dans le réseau à la manière d'un ver dans une galerie, la tête ouvrant le chemin pour le reste du message et la queue détruisant les connexions et libérant les liens après le passage du message, d'où le nom de "wormhole".

La réduction du délai d'acheminement des messages est particulièrement importante dans le cas d'un MSMC et il convient donc d'employer la technique "wormhole" ou l'une de ses variantes ([32]) chaque fois que cela sera possible.

5.1.5 Le routeur

Ce paragraphe, présente un routeur sous forme de programme en pseudo-OCCAM. La figure 5.1 en décrit un exemplaire à quatre liens ; chaque rectangle représentant un processus OCCAM et chaque flêche, un canal OCCAM.

Cette première version met en œuvre un mécanisme d'acquittement. Celui-ci fait double emploi avec celui déjà effectué par les transputers sur leurs liens.

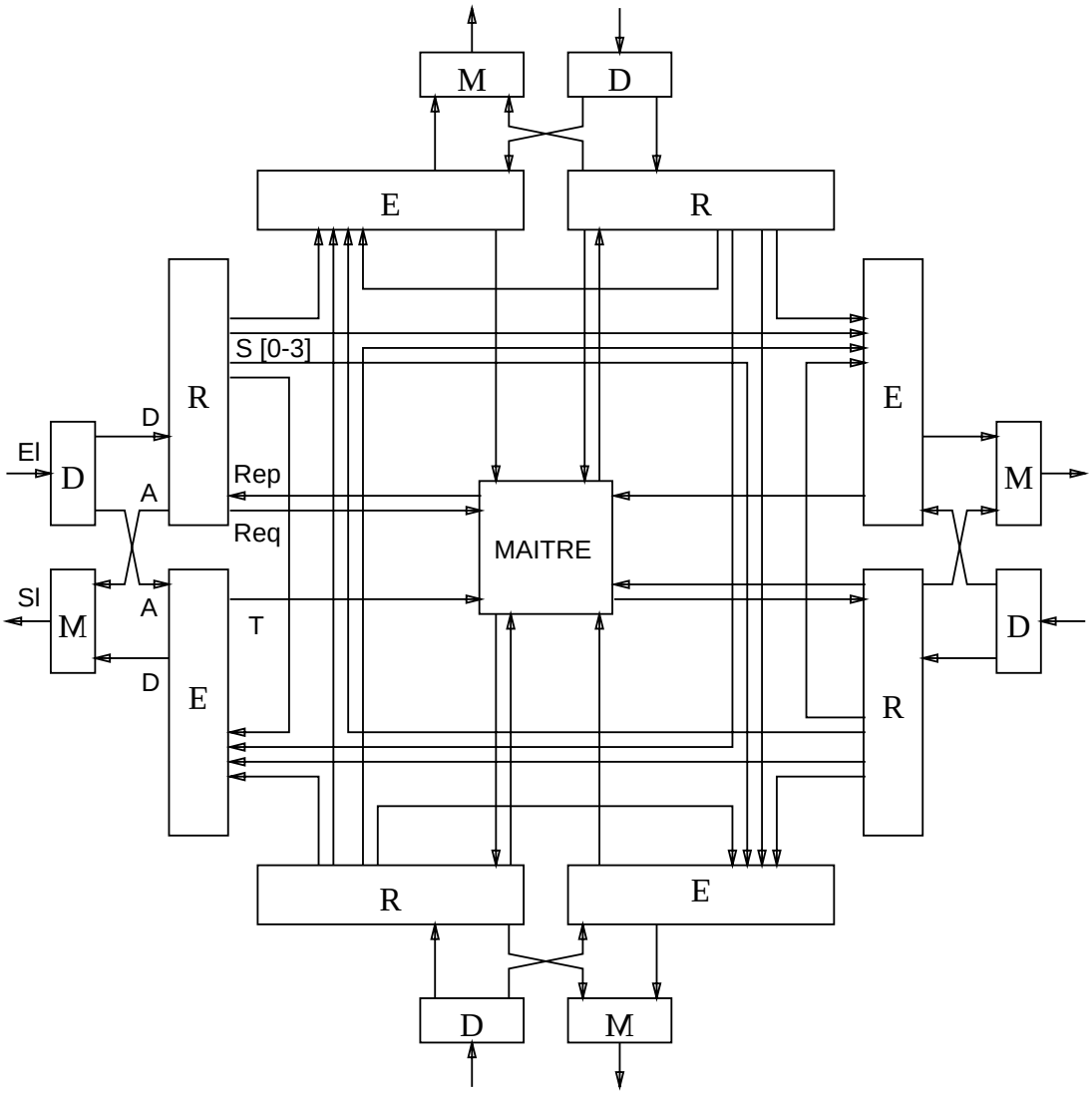


Figure 5.1: Exemple de routeur gérant quatre liens

Il exploite la technique "wormhole", accepte les fonctions de routage qui autorisent plusieurs liens de sortie par destination, gère le contrôle de flux de voisin à voisin et garantit une allocation équitable des liens de sortie suivant l'algorithme premier arrivé, premier servi par un mécanisme de

files d'attente.

Chaque message est composé d'un en-tête et d'un contenu. L'en-tête comprend au minimum deux champs :

- la destination du message et
- la longueur du contenu du message qui suit.

Dans le cas d'un routage adaptatif, plusieurs liens de sortie sont susceptibles d'être réclamés simultanément par un même message, auquel par contre un seul d'entre eux doit être alloué.

Lors de l'attribution d'un lien (précédemment occupé) à un message, la requête correspondante devra donc être retirée simultanément des files d'attente de tous les liens précédemment requis, de manière atomique.

Cette contrainte conduit donc logiquement à centraliser la gestion des files dans un unique processus.

Un routeur adaptatif peut être bâti autour de quatre sortes de processus entre lesquels le travail est réparti de la manière suivante :

- L'allocation des liens, donc la gestion des files d'attente, et la consultation de la table de routage sont attribuées à un processus maître qui constitue le cœur du routeur.
- La mise en œuvre du protocole d'acquittement sur chaque lien unidirectionnel de sortie est confiée à un processus émetteur. Pour les mêmes raisons, chaque lien unidirectionnel d'entrée est muni d'un processus récepteur, également chargé d'extraire l'identité de la destination du message.
- Deux processus auxiliaires assurent les fonctions de multiplexage associées à la conversion entre liens uni- et bidirectionnels.

Pour un routeur déterministe, le processus maître et le fonctionnement séquentiel peuvent être abandonnés au profit d'une structure décentralisée.

En effet, la gestion de chaque file d'attente peut être assurée par le processus émetteur correspondant ; les processus récepteurs consultant eux-même directement la table de routage.

Le processus récepteur de lien

Le processus récepteur R qui gère un lien d'entrée reçoit les données sur l'entrée D et renvoie les acquittements sur la sortie A.

Après réception de l'en-tête, il interroge le maître qui lui alloue le lien de sortie (libre) à utiliser. Il entre ensuite dans une boucle de recopie du message vers le lien de sortie.

```
PROC Recepteur (CHAN D,A,S[n],Req,Rep)
WHILE TRUE
  SEQ
    D ? <destination,longueur>
    Req ! destination
    Rep ? sortie
    S [sortie] ! <destination,longueur>
    A ! ANY
    SEQ i=0 FOR longueur
      D ? tampon
      S [sortie] ! tampon
      A ! ANY
```

Le processus émetteur de lien

Le processus émetteur E, qui gère de façon symétrique un lien de sortie, émet les données sur la sortie D et reçoit les acquittements sur l'entrée A.

```
PROC Emetteur (CHAN E[n],D,A,T)
WHILE TRUE
  SEQ
    T ! ANY
    ALT entree = 0 FOR n
      E [entree] ? <destination,longueur>
      SEQ
        D ! <destination,longueur>
        A ? ANY
        SEQ i=0 FOR longueur
          E [entree] ? tampon
          D ! tampon
          A ? ANY
```

Lorsqu'un message arrive sur l'une de ses entrées E, il exécute une boucle de recopie du message sur le lien de sortie; après quoi, via la sortie T, il informe le maître de la libération du lien qui peut être alloué à un autre message.

Le processus maître

Le processus maître reçoit les requêtes des récepteurs sur ses entrées Req et alloue les liens de sortie.

Il gère une file de type "premier arrivé premier servi" par lien de sortie.

Notons c un client (récepteur de lien), l un lien, L et L' deux ensembles de liens de sortie. Le maître utilise les primitives suivantes :

- **inter (L,L')** retourne, s'il existe, un élément de $L \cap L'$, et NUL dans le cas contraire.

- **ajouter_lien (l,L)** ajoute le lien l à l'ensemble L.
- **retirer_lien (l,L)** retire le lien l de l'ensemble L.
- **ajouter_client (c,L)** ajoute le client c en queue dans la file de chaque élément de L.
- **retirer_client (l)** retourne, s'il existe le premier client de la file associée au lien l et le retire de toutes les files dans lesquelles il a été placé ; retourne NUL si la file est vide.

Dans le processus maître décrit ci-dessous :

- **Libres** est l'ensemble (initialement vide) des liens de sortie inactifs.
- **Routage (d)** est la fonction de routage qui retourne l'ensemble des liens autorisés pour la destination d.

```
PROC Maitre (CHAN Req[n],Rep[n],T[n])
WHILE TRUE
  ALT
    ALT sortie=0 FOR n
      T [sortie] ? ANY     -- le lien sortie est libre
         SEQ
           client = retirer_client (sortie)
           IF client = NUL
               ajouter_lien (sortie,libres)
             TRUE
               Rep [client]! sortie   -- allouer le lien
    ALT entree=0 FOR n
      Req [entree] ? dest -- un message a router
        SEQ
          L_client = routage (dest)
          sortie = inter (L_client, libres)
          IF sortie = NUL
              ajouter_client (entree,L_client)
            TRUE
              SEQ
                ajouter_client (sortie,libres)
                Rep [client] ! sortie
```

Les processus de multiplexage

Pour simplifier le câblage, les liens unidirectionnels d'entrée et de sortie de mêmes extrémités sont associés par paires.

Les données circulant dans un sens et les acquittements des données transmises en sens inverse sont alors multiplexés sur une même voie de communication.

Le multiplexeur munit chaque donnée et chaque acquittement d'un préfixe de type et les multiplexe sur la voie de sortie.

```
PROC Multiplexeur (CHAN D,A,Sl)
WHILE TRUE
  ALT
    D ? tampon
      Sl ! <type-donnee; tampon>
    A ? ANY
      Sl ! type-acquittement
```

Le démultiplexeur gère la voie d'entrée, interprète les préfixes et effectue la tâche inverse de séparation.

```
PROC Demultiplexeur (CHAN El,D,A)
WHILE TRUE
  SEQ
    El ? type
    IF
      type=type-donnee
        SEQ
          El ? donnee
          D ! donnee
      ELSE
        A ! ANY
```

5.2 Le phénomène d'interblocage

5.2.1 Origine

Quelque soit la méthode de routage utilisée, l'acheminement des messages utilise un certain nombre de ressources :

- les liens, dans le cas de la commutation de circuits et
- les tampons, dans le cas de la commutation de paquets.

Les messages vont entrer en compétition pour ces ressources, lorsque leurs chemins ne sont pas disjoints.

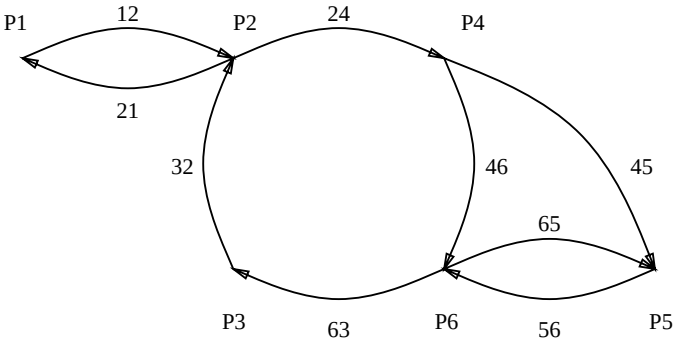
De cette compétition peut résulter un interblocage qui stoppe indéfiniment la progression d'un ensemble de messages.

A partir d'un exemple, nous analyserons le processus de formation d'un interblocage, puis les moyens d'y remédier et nous étudierons essentiellement les techniques de prévention (à opposer aux méthodes de détection et guérison).

Nous serons ainsi amenés à présenter deux méthodes originales de prévention de l'interblocage, adaptées à des réseaux à connectique irrégulière qu'autorise la reconfiguration synchrone.

5.2.2 Exemple

Considérons l'exemple de la figure 5.2 et les chemins définis par la fonction de routage (les signes + et - se rapportent à la section 5.4).



S \ D	1	2	3	4	5	6
1	-	12	- - + 12 24 46 63 - + 12 24 45 56 63	+ 12 24	- 12 24 45 - - 12 24 46 65	- - 12 24 46 - 12 24 45 56
2	21	-	- + 24 46 63 - + 24 45 56 63	+ 24	- 24 45 - - 24 46 65	- - 24 46 - 24 45 56
3	+ 32 21	+ 32	-	+ + 32 24	- - 32 24 45 - - - 32 24 46 65	- - - 32 24 46 - - 32 24 45 56
4	+ + 45 56 63 32 21 + + 45 63 32 21	+ + 45 56 63 32 + + 45 63 32	+ 45 56 63 - + 46 63	-	45 - 46 65	- 46 45 56
5	+ + 56 63 32 21	+ + 56 63 32	+ 56 63	+ + + 56 63 32 24	-	56
6	+ + 63 32 21	+ + 63 32	+ 63	+ + + 63 32 24	65	-

Figure 5.2: Interblocage sur un cycle de réseau physique

Supposons qu'à l'état initial les processeurs P_2, P_3, P_4 et P_6 produisent chacun un message :

- P_2 un message m_{26} à destination de P_6
- P_3 un message m_{34} à destination de P_4
- P_4 un message m_{43} à destination de P_3
- P_6 un message m_{62} à destination de P_2

Après invocation de la fonction de routage sur chaque processeur, à chaque message est alloué un lien sur lequel le transfert commence. Les messages occupent donc quatre liens :

- M_{62} occupe le lien L_{63}
- M_{43} occupe le lien L_{46}
- M_{34} occupe le lien L_{32}
- M_{26} occupe le lien L_{24}

Malheureusement, aucun message n'est alors en mesure de progresser. Chaque message attend la libération du lien suivant dans le boucle, lui-même occupé par un message qui attend la libération du lien suivant, et ainsi de suite. La boucle est bouclée, la progression de chaque message est bloquée indéfiniment : les liens L_{63}, L_{32}, L_{24} et L_{46} sont interbloqués.

5.2.3 Caractérisation

La notion de dépendance entre les liens pour l'étude de l'interblocage a été introduite par [28] pour la technique wormhole.

Définition Il existe une relation de dépendance d'un lien L_{ij} vers un lien L_{jk} si et seulement si $L_{ij}L_{jk}$ est une sous-chaîne d'un (au moins un) chemin défini par la fonction de routage.

L'interprétation de cette notion de dépendance est la suivante : la libération d'un lien occupé par un message m, implique l'acquisition par ce dernier du lien suivant de son chemin.

Par hypothèse, tout message délivré à sa destination y étant consommé au bout d'un temps fini, on démontre aisément par énumération de l'ensemble des liens bloqués que tout interblocage se traduit nécessairement par un cycle dans le graphe de (la relation de) dépendance (figure 5.3).

Remarques :

- Par définition, les liens reliés par un cycle dans le graphe de dépendance forment un cycle dans le réseau physique. Tout réseau physique acyclique sera donc libre d'interblocage.
- Si les chemins définis par la fonction de routage sont effectivement acycliques, tous les cycles du graphe de dépendance sont de longueur supérieure à deux.
- Pour démontrer l'absence d'interblocage dans une méthode quelconque, il suffira de montrer que la relation de dépendance est une relation monotone non constante, le graphe d'une telle fonction étant acyclique.

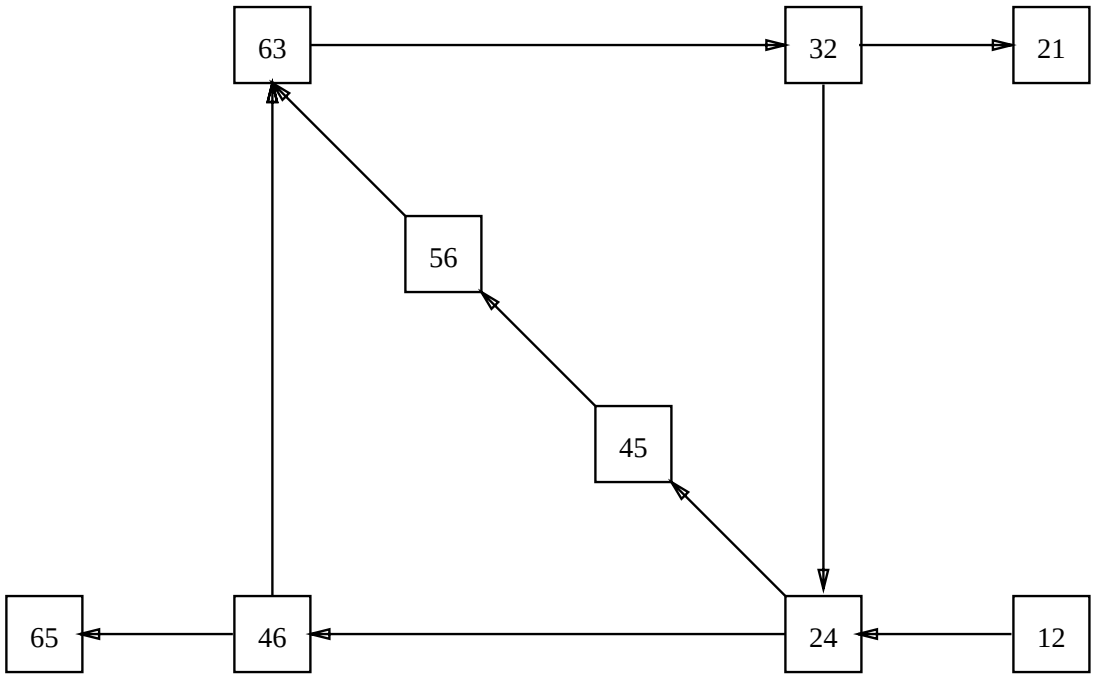


Figure 5.3: L'interblocage est associé à un cycle dans le graphe

5.2.4 Classification des méthodes de lutte

De très nombreuses méthodes de lutte contre l'interblocage ont été proposées, dont l'étude exhaustive sort du cadre de cette étude [33].

Ces méthodes se regroupent en deux grandes classes, dont une seule est compatible avec nos objectifs.

Action à postériori

Les méthodes d'action à postériori, dites de détection et guérison, agissent après la formation de l'interblocage. Elles reposent sur un mécanisme de détection des interblocages, chargé de déclencher une procédure particulière de guérison de l'interblocage.

Nous ne considérerons pas ces méthodes, qui présentent de manière générale les inconvénients suivant :

- Elles sont souvent conçues pour la commutation de paquets et le "store-and-forward", que nous voulons éviter.
- Les algorithmes de détection sont coûteux, à la fois en temps d'exécution sur les processeurs et en nombre de messages additionnels générés.
- La résorption de l'interblocage implique généralement :

- la destruction d'un certain nombre de paquets, d'où la nécessité de protocoles de gestion d'erreur de bout en bout chargés de régénérer ces paquets ou
- la mise en œuvre d'une politique spéciale de gestion des tampons pour la résorption de l'interblocage.

qui conduit dans les deux cas à alourdir l'algorithme de routage.

Action à priori

L'action à priori, qui consiste à réellement empêcher la formation de l'interblocage, caractérise au contraire les méthodes de prévention.

Les méthodes de prévention de l'interblocage destinées aux multiprocesseurs à connectique fixe reposent essentiellement sur trois approches :

- la recherche de fonctions de routage à graphe de dépendance acyclique,
- la virtualisation, qui consiste à recréer artificiellement le nombre nécessaire de ressources pour éviter l'interblocage et
- la mise en œuvre d'un routage adaptatif résolvant simultanément le problème de l'interblocage et celui de l'équilibrage dynamique de la charge des liens ;

dont nous allons étudier la transposition à des topologies quelconques.

5.3 Fonctions à graphe acyclique

Exhiber une fonction de routage à graphe de dépendance acyclique peut s'avérer une tâche aisée pour certaines topologies régulières.

En particulier, dans une grille à n dimensions, il suffit de parcourir les liens par ordre croissant de dimensions pour prévenir tout interblocage. Cet algorithme est connu sous le nom d'algorithme E-cube.

Ceci a conduit à rechercher comment générer une fonction de routage dont le graphe de dépendance reste acyclique pour des topologies quelconques.

Deux techniques de recherche de telles fonctions ont été conçues au L.G.I.. Elles supposent que les processeurs sont reliés par des liens bidirectionnels ; hypothèse vérifiée par la plupart des multiprocesseurs.

Messieurs Mugwaneza et Sakho [34] ont proposé une méthode basée sur l'existence d'un cycle eulérien, qui convient particulièrement bien aux réseaux de degré pair.

Mon collègue Gonzalez [30] et moi-même avons mis au point une autre méthode qui exploite un arbre recouvrant et ne priviégie pas une forme particulière de réseau.

5.3.1 Arbre recouvrant

Tout graphe connexe admet un arbre recouvrant.

Définitions : Le niveau N(α) d'un nœud α quelconque est la distance qui dans l'arbre le sépare de la racine.

Tout lien L_{xy} allant du nœud x au nœud y appartient par définition à l'une des trois catégories ci-dessous :

1. ascendant (c'est-à-dire orienté vers la racine) lorsque N(y) < N(x),
2. descendant (c'est-à-dire orienté vers les feuilles) lorsque N(y) > N(x),
3. horizontal lorsque N(y) = N(x) (le lien ne faisant pas partie de l'arbre).

Dans cet arbre, chaque lien bidirectionnel se décompose en une paire de liens unidirectionnels de liens opposés qui forment un cycle de longueur deux. Ceux-ci étant les seuls cycles présents dans l'arbre, l'absence d'interblocage est garantie.

Une fonction de routage triviale consiste à emprunter la suite de liens ascendants et la suite de liens descendant reliant les deux extrémités à leur ancêtre commun.

Par contre :

1. Les nœuds de parenté éloignée sont séparés par des chemins de longueur importante.
2. Les liens des niveaux proches de la racine se trouvent rapidement saturés.

Proposition : Il est possible de remédier à ces inconvénients en utilisant les liens n'appartenant pas à l'arbre pour raccourcir les chemins et les deux règles suivantes garantissent l'absence d'interblocage :

1. tout lien ascendant ne peut être précédé que d'un lien ascendant
2. aucun lien horizontal ne peut être précédé d'un autre lien horizontal.

Principe de la démonstration Soient i et j deux entiers tels que $1 \subseteq i \subseteq k$ et $1 \subseteq j \subseteq k$ avec k > 2

Soit un ensemble D de k+1 dépendances de liens de la forme $L_{y_{i-1}y_i} L_{y_i y_{i+1}}$

S'il existe une fonction g de l'ensemble des liens vers l'ensemble des réels telle que :

$$Il\ existe\ j\ tel\ que\ g(L_{y_j y_{j+1}}) > g(L_{y_{j-1}y_j})\ et \tag{5.1}$$

$$pour\ tout\ i\ diff\acute{e}rent\ de\ j\ ,\ g(L_{y_iy_{i+1}}) \geq g(L_{y_{i-1}y_i}) \tag{5.2}$$

alors l'ensemble des dépendances de liens ne peut pas former un cycle de longueur supérieure à deux et le réseau est libre d'interblocage.

Cette propiété peut être prouvée par l'absurde. En effet, additionnons membre à membre les inéquations définies en 5.1 et 5.2 :

$$\sum_{i=1}^{k} g(L_{y_iy_{i+1}}) > \sum_{i=1}^{k} g(L_{y_{i-1}y_i}) \tag{5.3}$$

$$g(L_{y_ky_{k+1}}) + \sum_{i=1}^{k-1} g(L_{y_iy_{i+1}}) > g(L_{y_0y_1}) + \sum_{i=2}^{k} g(L_{y_{i-1}y_i}) \tag{5.4}$$

Effectuons le changement de variable l = i - 1 :

$$g(L_{y_ky_{k+1}}) + \sum_{i=1}^{k-1} g(L_{y_iy_{i+1}}) > g(L_{y_0y_1}) + \sum_{l=1}^{k-1} g(L_{y_ly_{l+1}}) \tag{5.5}$$

$$d'o\grave{u}\ g(L_{y_ky_{k+1}}) > g(L_{y_0y_1}) \tag{5.6}$$

Par définition, si D forme un cycle de longueur k, alors :

$$y_k = y_o \tag{5.7}$$

$$y_{k+1} = y_1 \tag{5.8}$$

L'inéquation 5.6 devient alors :

$$d'o\grave{u}\ g(L_{y_0y_1}) > g(L_{y_0y_1}) \tag{5.9}$$

ce qui est absurde.

Démonstration : Soit la fonction g suivante :

- $g(L_{ij}) = +N(i)$ lorsque $N(j) > N(i)$
- $g(L_{ij}) = -N(i)$ lorsque $N(j) \leq N(i)$

Considérons tous les types de dépendance de lien L_{xy} L_{yz} :

1. Lien ascendant suivi d'un lien ascendant :

$$N(x) > N(y) \text{ et } N(y) > N(z), \text{ d'où :} \tag{5.10}$$

$$g(L_{yz}) = -N(y), g(L_{xy}) = -N(x) \text{ et } g(L_{yz}) > g(L_{xy}). \tag{5.11}$$

2. Lien ascendant suivi d'un lien descendant ou horizontal :

$$N(x) > N(y) \text{ et } N(y) \leq N(z) \text{ d'où :} \tag{5.12}$$

$$g(L_{yz}) = N(y), g(L_{xy}) = -N(x) \text{ et } g(L_{yz}) > g(L_{xy}). \tag{5.13}$$

3. Lien horizontal ou descendant suivi d'un lien ascendant : violation de la règle 1.

4. Lien horizontal suivi d'un lien horizontal : violation de la règle 2.

5. Lien horizontal suivi d'un lien descendant :

$$N(x) = N(y) \text{ et } N(y) < N(z) \text{ d'où :} \tag{5.14}$$

$$g(L_{yz}) = N(y)\ ,\ g(L_{xy}) = N(x) \text{ et } g(L_{yz}) = g(L_{xy}). \tag{5.15}$$

6. Lien descendant suivi d'un lien descendant ou horizontal :

$$N(x) < N(y) \text{ et } N(y) \leq N(z) \text{ d'où :} \tag{5.16}$$

$$g(L_{yz}) = N(y)\ ,\ g(L_{xy}) = N(x) \text{ et } \mathrm{g}(L_{yz}) > g(L_{xy}). \tag{5.17}$$

Tout chemin légal de longueur supérieure à deux générant au moins une dépendance des types 1,2 ou 6, le graphe de dépendance est acyclique et le réseau est libre d'interblocage.

Remarque La règle numéro deux peut être affaiblie si l'on ne crée pas de cycle de dépendance entre les seuls liens horizontaux du niveau considéré[1].

5.3.2 Cycle eulérien

Soit $G = (V, E)$ le graphe de degré pair associé au réseau considéré, chaque sommet de V étant associé à un processeur et chaque arc de E étant associé à un lien bidirectionnel (figure 5.4).

G admet un cycle eulérien dont nous noterons le graphe associé $G' = (V', E)$. Chaque sommet de degré 2n de V est représenté par n sommets de G' (figure 5.5).

Eliminons un arc quelconque $\mathrm{L}_{\alpha\beta}$ de G' et parcourons G' dans le sens α vers β et appliquons une numérotation croissante des sommets f telle que $f(\alpha) = 0$ et $f(\beta) = Card(V') - 1$.

Notons G'^{+} (respectivement G'^{-}) le graphe orienté obtenu en parcourant G' suivant l'ordre croissant (respectivement décroissant) des sommets.

[1]Soit H_i l'ensemble des liens horizontaux du niveau i et C_i son cardinal. Si le graphe est acyclique, il existe une fonction f qui à tout lien de H_i associe un numéro de 0 à C_i-1 tel que pour toute dépendance $\mathrm{L}_{ij}\mathrm{L}_{jk}$ on ait $\mathrm{f}(\mathrm{L}_{ij}) < \mathrm{f}(\mathrm{L}_{jk})$. La démonstration reste valide en modifiant la fonction g comme suit :

$\mathrm{g'}(\mathrm{L}_{ij}) = \mathrm{g}(\mathrm{L}_{ij})$ si $N(j) \neq N(i)$ et $\mathrm{g'}(\mathrm{L}_{ij}) = \mathrm{g}(\mathrm{L}_{ij}) + \mathrm{f}(\mathrm{L}_{ij})/\mathrm{C}_{N(i)}$ si $\mathrm{N(j)} = \mathrm{N(i)}$

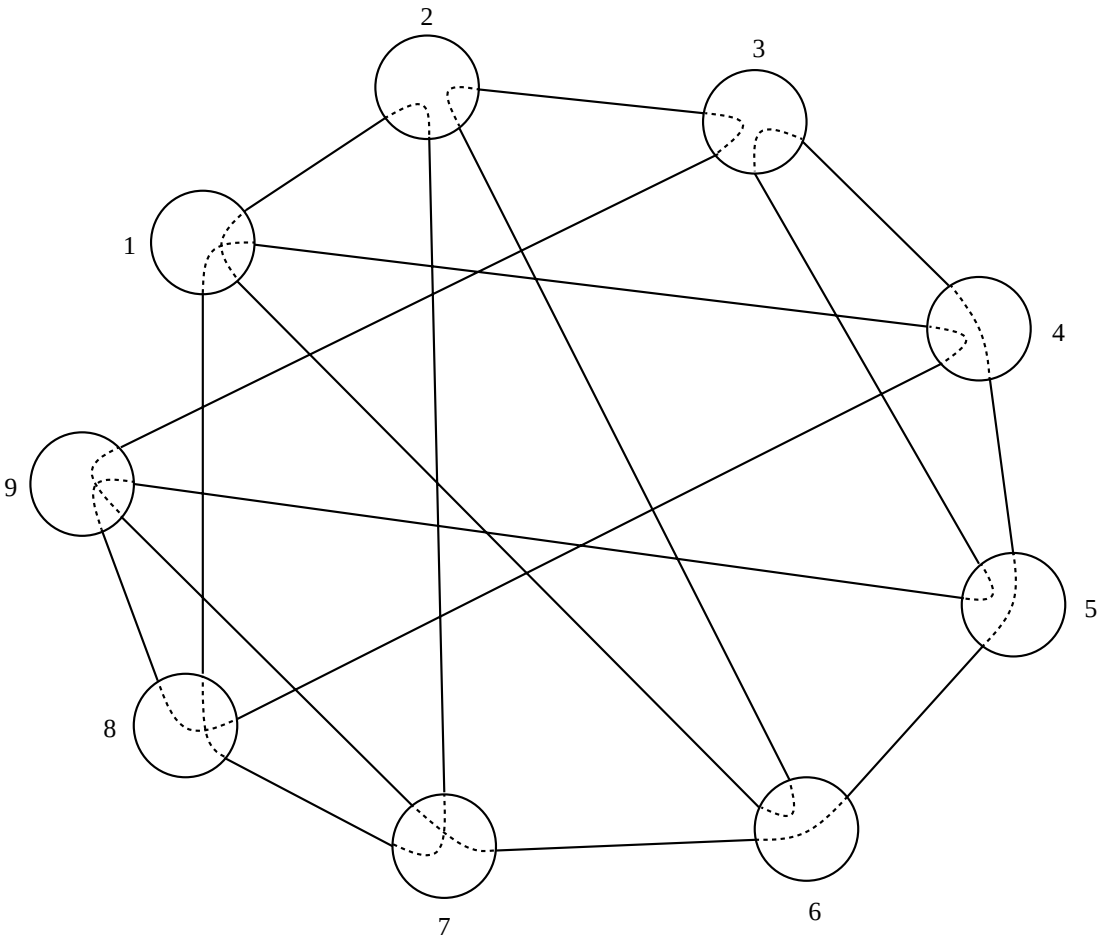


Figure 5.4: Le graphe G

Routage dans G' Soit une paire (x,y) de sommets de G'. Pour aller de x à y, nous parcourerons le graphe :

$$G'^{+} \text{ si } f(y) > f(x) \tag{5.18}$$

$$G'^{-} \text{ si } f(y) < f(x) \tag{5.19}$$

(l'arc $L_{\alpha\beta}$, n'appartenant ni à G'^{-} ni à G'^{+}, ne peut être utilisé)

Ces règles garantissent un graphe de dépendance acyclique, la relation de dépendance étant une relation d'ordre partiel. De plus, tout chemin comportant une sous-chaîne reliant deux sommets s' et s" de V' associés à un même sommet de V peut être raccourci en sautant directement de s' à s".

Ainsi, dans l'exemple, la chaîne $L_{84}L_{41}L_{18}$ peut être retirée du chemin $L_{98}L_{84}L_{41}L_{18}L_{87}$ allant du sommet numéro quatre au sommet numéro neuf, d'où le chemin réduit $L_{98}L_{87}$.

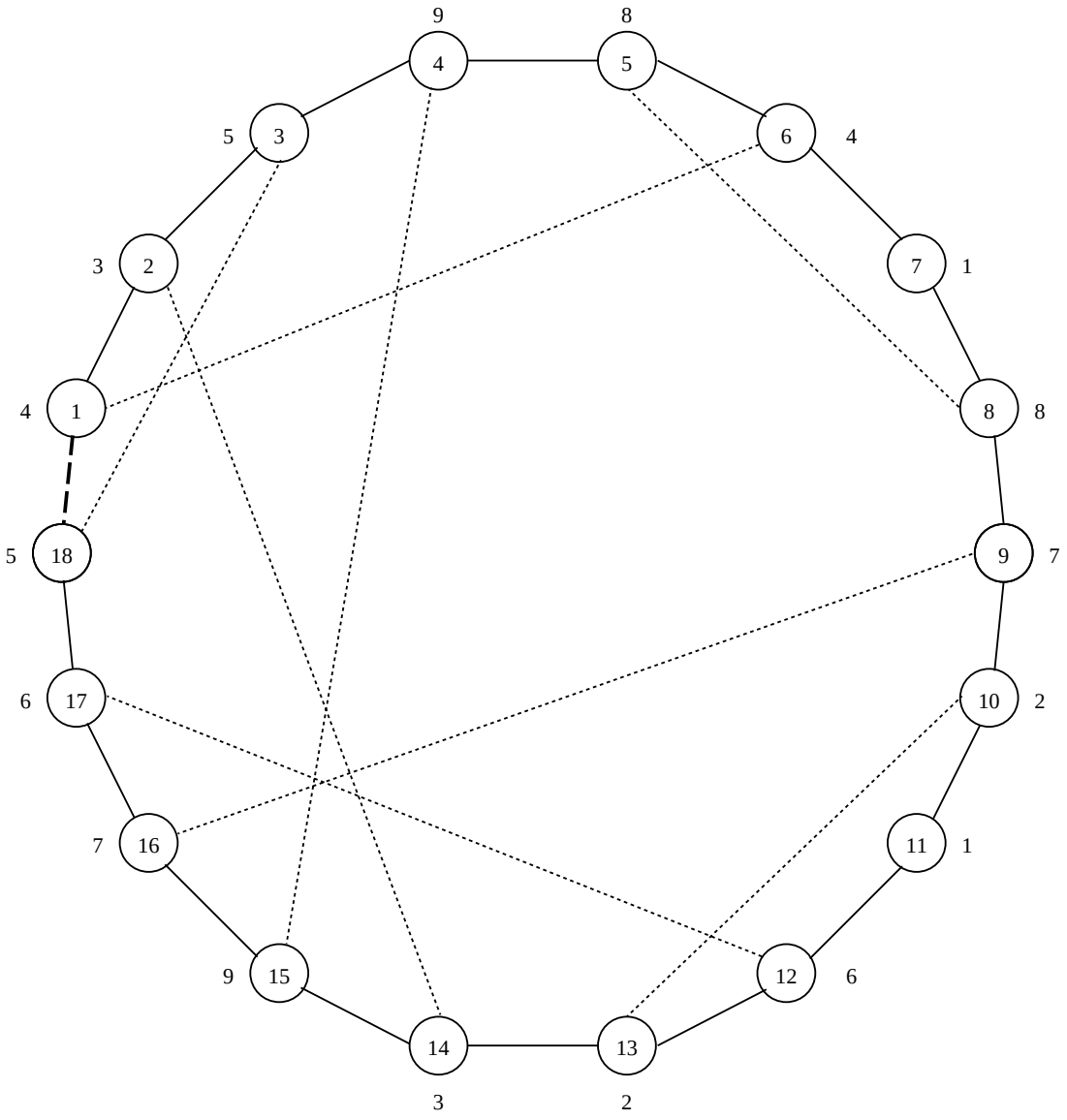


Figure 5.5: Le graphe G'

Nous utiliserons de la sorte les chemins suivant de G'^{+} pour aller :

1. De 2 à 9 : $L_{35}L_{59}L_{98}L_{87}$ (sommets 2, 3, 4, 5/8 et 9 de G'^{+}).
2. De 2 à 16 : $L_{39}L_{97}$ (sommets 2/14, 15 et 16)
3. De 9 à 14 : $L_{72}L_{23}$ (sommets 9, 10/13 et 14)
4. De 14 à 16 : $L_{39}L_{97}$ (sommets 14, 15 et 16)

et, symétriquement, les chemins suivant de G'^{-} pour aller :

1. De 9 à 2 : $L_{78}L_{89}L_{95}L_{53}$ (sommets 9, 8/5, 4, 3 et 2 de G'^{-}).

2. De 16 à 2 : $L_{79}L_{93}$ (sommets 16, 15 et 14/2)
3. De 14 à 9 : $L_{32}L_{27}$ (sommets14, 13/10 et 9)
4. De 16 à 14 : $L_{79}L_{93}$ (sommets 16, 15 et 14)

Routage dans G Considérons un processeur p qui doit router un message m vers une destination d :

- Lorsque m a été reçu sur un lien d'entrée L_{xp} un sommet p' de G' et un sens de parcours (+ ou -) sont identifiés de façon unique (faute de quoi les règles 5.18 et 5.19 seraient violées). M peut alors emprunter tout chemin partant de p':
  - respectant le sens de parcours
  - aboutissant à un sommet d' associé à la destination d.
- Lorsque la source du message m est locale, m peut emprunter tout chemin :
  - partant d'un sommet de V' associé à p et
  - aboutissant à un sommet de V' associé à d

Dans l'exemple, un message m destiné au processeur numéro 7 et reçu sur le lien L_{43} identifie le sommet 2 de G'^{+}. M peut donc utiliser les chemins de 2 à 9 et de 2 à 16 de G'^{+}, d'où le choix d'un des liens de sortie L_{35} et L_{39}.

E \ S	L32	L34	L35	L39
L23	INTERDIT	INTERDIT	INTERDIT	AUTORISE
L53	INTERDIT	INTERDIT	INTERDIT	INTERDIT
L43	INTERDIT	INTERDIT	AUTORISE	AUTORISE
L93	AUTORISE	INTERDIT	INTERDIT	INTERDIT
Locale	AUTORISE	INTERDIT	AUTORISE	AUTORISE

Figure 5.6: Table : processeur 3 vers processeur 7

Le message m reçu sur le lien L_{93} identifie le sommet 14 de G'^{-}. Le seul chemin autorisé va de 14 à 9, d'où le choix du lien de sortie L_{32}. Si le lien d'entrée était L_{53}, aucun chemin ne serait autorisé. Un tel cas ne doit cependant pas se produire, la table de routage du processeur 5 n'autorisant pas le lien de sortie L_{53} pour la destination 7.

La table de la figure 5.6 récapitule l'ensemble des chemins définis pour le processeur numéro 3 et la destination numéro 7.

Pour l'extension de la méthode au graphe possédant des nœuds de degré impair, nous distinguerons les cas suivant :

1. Seuls deux nœuds sont de degré impair : il existe alors un chemin eulérien reliant les deux nœuds.

2. Nœud de degré un : retirer ce nœud du graphe. Le calcul des chemins pour ce nœud se ramène au calcul des chemins pour le nœud auquel il est connecté.

3. Nœud N dont la suppression d'un lien entraîne la création dans le graphe de deux composantes connexes disjointes :

   - traiter séparément chaque composante connexe.
   - pour aller d'un processeur x à un processeur y n'appartenant pas à la même composante, concaténer le chemins de x à N et de N à y.

4. Nœud de degré impair (supérieur à un) dont la suppression ne rompt pas la connexité du graphe : éliminer un lien (qui ne fera donc partie d'aucun chemin de longueur deux ou plus).

Remarque : Dans le cas des machines de la famille Supernode, un cycle eulérien est déjà calculé pour générer les commandes des circuits du réseau d'interconnexion.

5.4 Virtualisation

Il est possible de reconstruire par multiplexage un nombre quelconque de liens virtuels sur un même lien physique. Chaque lien virtuel possédant son propre contrôle de flux, doit être muni d'un tampon capable de stocker au minimum une donnée et le mécanisme d'acquittement doit être reconstruit en tenant compte des numéros de lien virtuel.

[Dally] et [Jesshope] montrent sur différents réseaux à connectique régulière comment l'introduction des liens virtuels dédiés chacun à un ensemble donné de destinations permet de briser les cycles du graphe de dépendance.

Appliqué à des topologies quelconques, le principe de la méthode reste valide ; il nous suffit d'en établir la démonstration que ne donnent pas les articles originaux.

Hypothèse de travail : Pour tout nœud D du réseau, pris comme destination, il existe au moins un lien de tout cycle C du graphe de dépendance n'appartenant à aucun des chemins définis pour D par la fonction de routage.

Cette hypothèse est en particulier une condition que doivent vérifier les fonction de routage générant des chemins acycliques lorsque l'histoire du message, c'est-à-dire son origine et le lien d'entrée sur lequel il a été reçu, n'est pas prise en compte.

Notations Soit deux entiers i et y tels que :

$$0 \leq y \leq n \text{ et } 0 \leq i \leq n \tag{5.20}$$

Notons dest(L) l'ensemble des destinations des chemins auxquels appartient le lien L et lien(N) l'ensemble des liens appartenant aux chemins conduisant à N.

Notons $(x)_n$ la fonction x modulo n.

Soit C un ensemble de liens formant un cycle élémentaire de longueur n (c'est-à-dire tel qu'il n'existe pas de cycle de longueur inférieure à n dont les sommets soient tous des liens de C) du graphe dépendance.

Par définition du graphe de dépendance, tout lien L d'un cycle est tel que $dest(L) \neq \phi$.

Choisissons un élément x parmi l'un des ensembles de destinations des liens du cycle.

Les liens peuvent être numérotés d'après leur position dans le cycle, soit :

$C = \{L_{s_i s_{(i+1)_n}}; 0 \leq n\}$

$L_{s_0 s_1}$étant un lien tel que :

$dest(L_{s_0 s_1}) \bigcap \{x\} = \phi$,

dont l'hypothèse de travail garantit l'existence.

Soit la fonction signe :

$$\text{Si } x > 0, S(x) = +1 \text{ et } s[x] = + \tag{5.21}$$

$$\text{Si } x = 0, S(x) = 0 \text{ et } s[x] = 0 \tag{5.22}$$

$$\text{Si } x < 0, S(x) = -1 \text{ et } s[x] = - \tag{5.23}$$

Principe : Soit $d\varepsilon \bigcup_{i=0}^{n-1} dest(L_{s_i s_{(i+1)_n}})$une destination de l'un des chemins contenant (au moins) un lien du cycle.

Soit $L_{s_y s_{(y+1)_n}}$le (ou l'un des) lien du cycle tels que :

$$dest(L_{s_y s_{(y+1)_n}}) \bigcap \{d\} \neq \phi \tag{5.24}$$

Pour briser le cycle C dans le graphe de dépendance, il suffit de remplacer chaque lien $L_{s_i s_{(i+1)_n}}$par une paire de liens virtuels $L^{+}_{s_i s_{(i+1)_n}}$et $L^{-}_{s_i s_{(i+1)_n}}$et de subtituer au lien $L_{s_i s_{(i+1)_n}}$le lien virtuel $L^{s[(y-i)_n]}_{s_i s_{(i+1)_n}}$dans chaque chemin conduisant à d (figures 5.2 et 5.7).

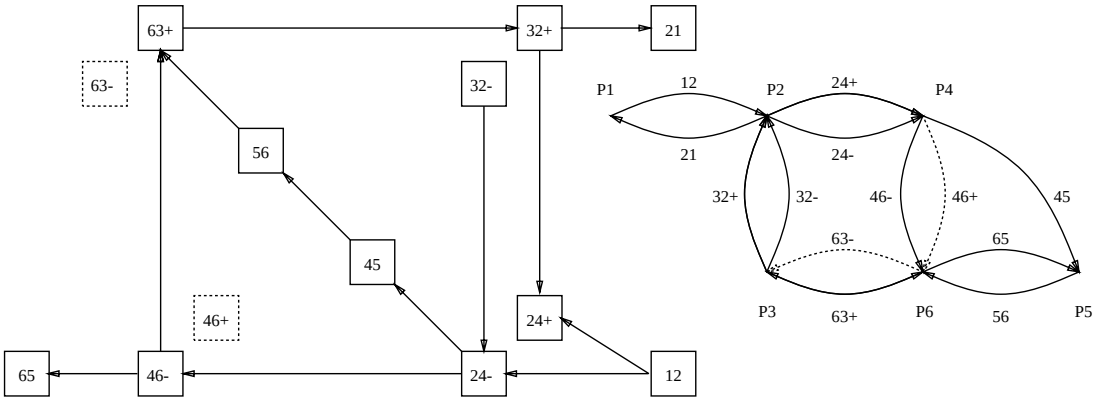


Figure 5.7: La virtualisation brise les cycles du graphe

Démonstration : Nous avons vu que tout cycle du graphe de dépendance est de longueur supérieure à deux :

$$n > 2 \tag{5.25}$$

Toute dépendance entre deux liens du cycle C créé par un chemin menant à d est de la forme :

$$L^{s[X1]}_{s_{(j)_n} s_{(j+1)_n}} vers L^{s[X2]}_{s_{(j+1)_n} s_{(j+2)_n}} avec X_1 = (y-j)_n et X_2 = (y-(j+1))_n \tag{5.26}$$

La dépendance ne peut porter sur $L_{s_y s_{(y+1)_n}}$, qui n'appartient par définition à aucun chemin :

$$y < j < y + n \tag{5.27}$$

$$(j)_n \neq y \tag{5.28}$$

$$(j+1)_n \neq y \tag{5.29}$$

Ce qui donne :

$$X1 = y - (j)_n \tag{5.30}$$

$$X2 = y - (j+1)_n \tag{5.31}$$

La combinaison des équations ci-dessus implique que les termes X1 et X2 sont non nuls :

$$X1 \neq 0 \text{ et } X2 \neq 0 \tag{5.32}$$

- Si $(j)_n$=n-1 alors :
  - l'équation 5.30 devient X1 = y - (n-1) et la combinaison de 5.20 et 5.32 donne :

$$X1 < 0, S(X1) = -1 \text{ et } s[X1] = -. \tag{5.33}$$

  - l'équation 5.31 devient X2 = y et 5.20 implique alors :

$$X2 < 0, S(X2) = +1 \text{ et } s[X2] = +. \tag{5.34}$$

- Si $(j)_n \neq n-1$ alors la combinaison de 5.30 et 5.31 donne :

$$X2 - X1 = (j+1)_n - (j)_n \text{ d'où } X2 = X1 - 1. \tag{5.35}$$

  Dans ce cas :

  - X1<0 implique X2<0. On a alors :

$$S(X1) = S(X2) = -1 \text{ et } s[X1] = s[X2] = -. \tag{5.36}$$

  - X1>0 implique X2>0 (X2 = 0 contredisant 5.32). On a alors :

$$S(X1) = S(X2) = +1 \text{ et } s[X1] = s[X2] = +. \tag{5.37}$$

La fonction est monotone croissante.

L'application répée du processus aux autres destinations permet ainsi briser successivement tous les cycles du graphe de dépendance. Dans le pire des cas, un lien physique donnera naissance à un ensemble de liens virtuels associés chacun à une seule destination.

Au prix d'une consommation de mémoire supplémentaire, il est bien sûr possible de diviser directement chaque lien physique en autant de liens virtuels que de destinations possibles des chemins auquel le lien appartient, ce qui évite une étape de calcul. Dans ce cas, le graphe de dépendance se décompose en autant de sous-graphes disjoints que de destinations et dans chaque sous-graphe, les propriétés de la fonction de routage garantissent par hypothèse l'absence de cycle.

Citons enfin pour mémoire les méthodes de structuration de l'ensemble des tampons, qui ont été conçues pour le "store-and-forward". Celles-ci reposent sur un principe analogue à celui des liens virtuels, la source ou la destination des messages définissant le critère d'accès à un tampon ou un groupe de tampons.

5.5 Routage adaptatif

Lorsque les chemins de routage sont déterminés statiquement (avant l'exécution de l'application) et de manière déterministe (un seul chemin autorisé par paire de processeurs), on assiste généralement lors de l'exécution à la formation dans le réseau d'interconnexion de congestions locales plus ou moins temporaires qui ralentissent notablement l'acheminement des messages.

Le routage adaptatif autorise au contraire plusieurs liens de sortie, parmi lesquels le choix est réalisé en temps réel en fonction de l'occupation des liens, ce qui réduit les conséquences et la probabilité d'apparation de congestions.

Les deux méthodes citées ci-dessous mettant en jeu une forme de routage adaptatif reposant sur un mécanisme de transmission des messages qui ne génère pas de dépendance entre les liens.

La première, tire parti des propriétés topologiques des graphes dits "hypercubes" et la seconde repose sur un mécanisme d'échange de messages.

L'étude de ces deux techniques va montrer que si l'interblocage est évité de manière intrinsèque par le principe de fonctionnement du routeur ; des précautions particulières doivent être prises pour éviter les phénomènes de famine ou de circulation infinie des messages dans le réseau.

5.5.1 Exploration des chemins dans un hypercube

Considérons un hypercube d'ordre k et un message à transférer d'un nœud source S à un nœud destination D.

La méthode consiste à construire un chemin de liens libres partant de S et dont la longueur augmente d'une unité à chaque étape de l'algorithme de construction jusqu'à ce que la destination soit atteinte.

Soit E l'ensemble des nœuds du réseau déjà explorés et N le nœud courant à l'étape considérée.

A l'état initial, E={S} et N=S.

A chaque étape, un lien libre de N conduisant à un nœud N' non encore exploré est réservé et ajouté au chemin en cours de construction.

L'algorithme est exécuté de manière récursive et se termine lorsque la destination D devient le nœud courant. Lorsque le nouveau nœud courant N' n'est pas égal à la destination D du message, N' est ajouté à l'ensemble E des nœuds déjà explorés et devient le nouveau nœud courant.

Lorsque tous les voisins auxquels est directement connecté le nœud courant ont été explorés sans succès, le dernier lien est retiré du chemin en construction et libéré et son extrémité redevient le nœud courant.

Sachant que tout nœud possède k liens, il existe k chemins disjoints deux à deux entre toute paire de processeurs. L'algorithme présenté [25] utilise une heuristique de recherche qui élimine de l'ensemble des candidats tous les chemins passant par un nœud déjà exploré.

Par construction, cet algorithme de routage s'adapte en temps réel à l'évolution de la répartition des flux de communication sur les liens.

De plus, tout lien occupé est libéré au bout d'un temps fini par épuisement des chemins candidats passant par le lien considéré (en cas d'échec) ou par transfert du message (de durée finie) en cas de succès dans l'établissement d'un chemin. Tout interblocage est donc éliminé par construction.

Un phénomène de famine peut toutefois se produire lorsque la saturation du réseau d'interconnexion est proche. En effet, chaque recherche de chemin bloque momentanément les liens du chemin en cours de construction.

Même si aucun transfert de message n'a lieu, une recherche de chemin peut aboutir à un échec dû à l'occupation temporaire d'un ou plusieurs liens faisant partie d'autres chemins en construction. La multiplication des recherches simultanées de chemins peut ainsi dégénérer et aboutir à une succession non bornée d'échecs alors même qu'aucun transfert de message n'est en cours sur les liens.

Ce problème peut être contourné de deux manières :

- Par un dimensionnement adéquat du réseau d'interconnexion et le recours à des opérateurs rapides (câblés) de recherche de chemin, la probabilité d'apparition d'un tel phénomène peut être rendue aussi faible que nécessaire.
- En ménageant une transition vers l'algorithme E-cube en cas d'échec.

La souplesse d'adaptation de cet algorithme laisse espérer des gains de performances importants par rapport au seul algorithme E-cube.

L'adaptation de cet algorithme ne semble pas réalisable, les propriétés spécifiques de l'hypercube qui permettent la limitation du nombre de chemins à explorer n'ayant pas d'équivalent dans les connectiques irrégulières.

5.5.2 Routage adaptatif généralisé

Soit un réseau de processeurs interconnectés par des liens bidirectionnels et un routeur déterministe routant par commutation de paquets les messages suivant les plus courts chemins.

Considérons un nœud y muni d'un nombre de tampons supérieur ou égal au nombre de ses liens et qui reçoit sur ses liens d'entrée m paquets à router.

Dans le cas favorable, les paquets sont réacheminés via m liens de sortie différents. Les flux d'entrée et de sortie étant équilibrés, le nombre de tampons occupés reste constant.

Il n'en va plus de même lorsque les paquets doivent être routés sur le même lien de sortie. Un seul paquet emprunte alors le lien et les m-1 autres sont stockés en attendant la libération du lien. Le flux d'entrée excède alors le flux de sortie et m-1 tampons libres sont consommés par le stockage des paquets en attente.

Lorsque l'ensemble des tampons libres est épuisé, les liens d'entrée se bloquent; et ce blocage va se propager de nœud en nœud. Lorsqu'un cycle de liens bloqués est formé, le réseau est interbloqué et la progression des messages stoppée définitivement.

Du raisonnement ci-dessous :

1. Lorsqu'un paquet rencontre sur le plus court chemin conduisant à sa destination une congestion locale (plusieurs paquets en attente d'un lien de sortie occupé), sa progression est stoppée jusqu'à la libération de ce lien.

   On peut espérer accélérer son acheminement et la résorption de la congestion en reroutant ce paquet le long d'un chemin plus long, mais composé de liens libres, qui lui permette de contourner la zone congestionnée.

2. Aucun interblocage ne se produit lorsque sur chaque nœud les flux d'entrée et de sortie sont équilibrés.

3. En l'absence de tampons libres, il est toujours possible d'échanger les contenus de deux tampons.

découle le principe de fonctionnement suivant :

- Le routage est assuré par échange de paquets entre voisins, chaque message suivant en temps normal le plus court chemin conduisant à sa destination.

- Lorsque des tampons sont libres, l'un des deux messages échangés peut être nul, ce qui revient à transférer un tampon libre d'un processeur vers son voisin.

- En cas de conflit d'accès à un même lien de sortie, le lien est alloué à l'un des paquets. Les autres paquets peuvent être stockés sur place lorsque des tampons libres sont disponibles ou reroutés vers d'autres liens (qui peuvent appartenir aux chemins déjà parcourus par les paquets).

Par construction, l'échange de deux paquets entre voisins étant toujours possible, tout interblocage est exclu.

Par contre, un message peut être évincé par d'autres paquets pour l'accès aux liens conduisant à sa destination, et rerouté indéfiniment dans le réseau sans jamais atteindre sa destination.

Un autre problème se pose lors de l'injection des paquets dans le réseau lorsque ces paquets ne sont pas échangés avec des paquets destinés au nœud émetteur. L'injection des paquets consomme donc des tampons de stockage. Pour renouveler le stock de tampons libres, il convient de garantir que chaque nœud émetteur recevra au bout d'un temps fini un message nul ou à lui adressé, ce qui lui permettra de récupérer un tampon libre pour l'injection de nouveaux paquets.

Pour contrer ces deux phénomènes, [35] propose :

- Une numérotation des paquets par date, les paquets les plus anciens étant prioritaires pour l'accès aux liens de sortie.
- Un mécanisme assez lourd de marquage des messages transférés qui permet d'assurer qu'un tampon du nœud source d'un paquet sera libéré au bout d'un temps fini, rétablissant l'équilibre rompu lors de l'injection du paquet.

5.6 Comparaison des méthodes

Trois techniques différentes de prévention de l'interblocage pour connectiques quelconques ont été présentées dans les paragraphes précédents :

- la virtualisation, qui recrée artificellement autant de ressources que nécessaire,
- la restriction à des fonctions de routage à graphe de dépendance acyclique,
- le routage adaptatif généralisé basé sur le mécanisme d'échange

qu'il convient de comparer suivant les critères enumérés ci-dessous.

5.6.1 Espace de mémorisation

La taille de l'espace de mémorisation consommé sur chaque nœud par le routage doit être bornée et minimisée, ce qui permet d'intégrer la mémoire sur la même puce que le routeur (accélération du routeur) d'une part et garantit l'extensibilité des machines.

Contrairement aux deux autres méthodes, la virtualisation ne permet pas de travailler à espace constant puisque l'augmentation de la taille du réseau est toujours susceptible d'accroître le nombre de liens virtuels par lien physique sur un même nœud.

5.6.2 Longueur des chemins

Avant l'exécution de l'application, l'ensemble des chemins doit être précalculé et encodé dans des tables de routage.

Seule la restriction des fonctions de routage empêche (par définition) d'aboutir systématiquement à tous les plus courts chemins.

Les premiers résultats obtenus avec cette méthode laisse cependant espérer une approximation acceptable de ceux-ci [30].

5.6.3 Temps de calcul des tables

La méthode de routage adaptatif généralisé ne demande que le calcul des plus courts chemins, dont le calcul peut être parallélisé (annexe A).

Le même principe de parallélisation reste valable pour la méthode de l'arbre recouvrant et [34] présente également la parallélisation du calcul des tables à partir du cycle eulérien.

Toutes les méthodes sont donc comparables de ce point de vue, à l'exception de celle des liens virtuels. En effet, créer un lien virtuel par lien physique et par destination demande une quantité de mémoire très excessive, d'où l'introduction d'une coûteuse étape de minimisation du nombre de liens virtuels, difficilement parallélisable.

5.6.4 Tables de routage

L'encodage le plus compact des tables autorisant tous les plus courts chemins associe un booléen à chaque paire (lien physique, destination).

Pour la virtualisation, ce booléen n'est pas suffisant et doit être remplacé par le numéro du canal virtuel associé à cette destination pour le lien physique considéré, d'où une augmentation de la mémoire consacrée aux tables de routage.

5.6.5 Résistance aux congestions locales

Le routage adaptatif généralisé est par construction la méthode qui se comporte le mieux en cas de déséquilibre de charge (permanente ou transitoire) sur les liens. Les deux autres méthodes offrent seulement une forme limitée de routage adaptatif lorsque plusieurs chemins sont autorisés vers une même destination, avec un léger avantage pour la virtualisation qui accepte tous les plus courts chemins.

5.6.6 Coût de mise en œuvre

Prenons comme référence le routeur présenté au paragraphe 5.1.5, et évaluons les mofications apportées par les différentes méthodes :

- La restriction n'apporte aucune modification et peut donc être considérée comme optimale de ce point de vue.
- Pour la virtualisation, chaque donnée et chaque acquittement transmis sur le lien doit être accompagné du numéro du lien virtuel auquel il appartient, d'où une réduction du débit effectif des liens. D'autre part, du fait du nombre variable de liens virtuels, la gestion des files doit être programmée et non plus câblée, d'où une nouvelle perte d'efficacité.
- Dans le cas du routage adaptatif généralisé, l'opération de base est l'échange de messages, d'où un algorithme de routage plus complexe et plus coûteux en temps de calcul. De plus, pour éviter les cas de famine, une technique de réservation des tampons doit être mise en œuvre, auquel est associée un flux d'informations, qui vient, comme dans le cas précédent, réduire le débit effectif des liens.

5.6.7 Choix retenu

Le critère de l'occupation mémoire justifie à lui seul la mise à exclure la virtualisation.

Le coût de mise en œuvre a conduit à écarter ensuite le routage adaptatif généralisé.

La deuxième justification de ce choix réside dans le fait que la reconfiguration permet déjà de combattre le phénomène de congestion locale par un choix judicieux de la topologie pour chaque application.

La prévention de l'interblocage reposera donc sur l'une des deux méthodes du paragraphe 5.3, qui permettent de conserver un routeur simple, facile à câbler et qui ne monopolisera pas les ressources du nœud de calcul qui l'exécute s'il est programmé (cas des transputers actuels).

5.7 Adaptation aux machines Supernode

Deux modifications ont été apportées au routeur en vue de son exécution sur les machines Supernode. Elles sont intimement liées au caractéritiques des liens des transputers ; les deux points clés étant :

- le mode de fonctionnement par transfert de bloc mémoire et
- le protocole utilisé lors de l'amorçage.

5.7.1 Emploi du "store-and-forward"

Le protocole d'acquittement et la conversion entre liens uni- et bidirectionnels du transputer sont figées et directement supportées par le matériel, ce qui accroît d'autant les performances (voir chapitre 3).

Ceci réduit la composante logicielle du routeur ; les processus de multiplexage et les canaux de gestion des acquittements disparaissant du programme OCCAM.

L'unique primitive de communication sur le lien est le transfert de bloc mémoire à déclenchement logiciel, d'où le recours obligatoire à la technique du "store-and-forward", (avec un découpage en paquets pour les messages qui excèdent la taille des tampons) avec ses répercussions sur le délai d'acheminement des messages.

L'absence de dispositif matériel de synchronisation entre un lien de sortie et un lien d'entrée d'un même transputer, qui aurait permis de déclencher la réémission d'un paquet avant de l'avoir reçu en totalité (et de conserver ainsi le faible délai de traversée autorisé par le Wormhole), s'avère donc particulièrement regrettable .

5.7.2 Installation du noyau

Dans les machines Supernode, les transputers sont dépourvus de mémoire morte et le routeur doit être installé à partir de la mémoire non volatile d'un nœud particulier, que nous appelerons racine du réseau.

Lorsque le noyau est capable de s'autodupliquer, la racine initialise ses voisins immédiats qui peuvent à leur tour initialiser directement leurs propres voisins et ainsi de suite.

Dans le cas contraire, le noyau seront également installés par ordre croissant de distance à la racine, mais pour chaque processeur une nouvelle copie du code à charger doit être puisée dans la mémoire permanente de la racine et routée de celle-ci jusqu'au processeur à amorcer.

Une amorce ne peut être acheminée par le routeur que précédée d'un en-tête. Or ce dernier est inévitablement incompatible avec le protocole de téléchargement imposé par le transputer, suivant lequel le contenu de l'amorce ne doit être précédé que d'un simple octet de longueur.

C'est pourquoi le routeur a été muni d'un mode spécial de suppression d'en-tête qui permet de ne délivrer que le seul contenu d'un paquet à sa destination.

Le routeur modifié

Pour réduire le temps de calcul, le routeur utilisé est de type déterministe, d'où la suppression du processus maître.

Il ne reste plus que deux types de processus : les récepteurs de liens qui assurent la consultation des tables de routage et les émetteurs de liens, décrits ci-dessous :

Le récepteur de lien

```
Recepteur (l)
WHILE TRUE
  SEQ
    -- recevoir l'en-tete
    entree[l] ? en_tete[l]
    -- et le contenu eventuel dans le tampon t
    IF en_tete[l].longueur <> 0
      entree[l] ? t[l][0 FOR en-tete[l].longueur]
    IF
      en_tete[l].destination = NUMERO_LOCAL
        -- destination interne : vers protocole
        SEQ
          -- signaler l'arrivee du message
          vers_protocole[l] ! ANY
          -- attendre fin de consommation
          -- du message par le protocle
          vers_protocole[l] ! ANY
      TRUE
        -- trouver le lien de sortie
        SEQ
          correspondant = table[l][destination]
          -- delivrer et attendre consommation
          vers_emetteur[correpondant] ! ANY
          -- du message par l'emetteur
          vers_emetteur[correspondant] ! ANY
```

L'émetteur de lien

```
Emetteur (l)
WHILE TRUE
  -- l'unite de protocole est un
  -- client supplementaire
  ALT cli = 0 FOR Nb_ls + 1
    vers_emetteur[l] ? ANY
      SEQ
        -- envoi de l'en_tete si pas
        -- de mode special pour les amorces
        IF (NOT (en_tete[cli].suppression) OR
           en_tete[cli].destination <> voisin[l])
          sortie[l] ! en_tete[cli]
        -- emission du contenu si present
        IF en_tete[cli].long <> 0
          sortie[l] ! t[cli][0 FOR en_tete[cli].long
```

Plusieurs versions de routeur basées sur cet algorithmeont été développées pour les besoins du projet.

Sur la plus rapide d'entre elles, écritre en langage d'assemblage, la durée estimée du calcul associé à la traversée d'un nœud par un message est située dans une fourchette alllant de vingt et quarante microsecondes selon les conditions d'exécution (fréquence d'horloge du transputer, mémoire interne ou externe, etc ...).

5.8 Reconfiguration dynamique

Ce paragraphe présente le déroulement d'une transition d'étape et la durée approximative de celle-ci sera évaluée.

Chaque transition recouvre en fait quatre opérations élémentaires :

1. la détection de la fin d'étape de calcul,
2. la reconfiguration du réseau (chrgement des tables de commandes dans les crossbars),
3. la mise à jour des tables de routage et
4. la diffusion d'un signal de reprise déclenchant l'exécution de l'étape suivante.

5.8.1 Détection de fin d'étape

Un fin d'étape correspond à la suspension de l'exécution de chacune des tâches suite à la rencontre d'un point de reconfiguration. Elle équivaut de ce fait à la réalisation d'un rendez-vous sur l'ensemble des processeurs virtuels.

Les machines Supernode offrent un support matériel adapté à chacun des niveaux de détection :

1. Le jeu d'instructions de transputer supporte directement la détection de la fin des tâches exécutées sur un même nœud.
2. A l'intérieur d'un nœud simple ou d'un tandem, la détection de rendez-vous entre les processeurs de travail est directement réalisée par le mécanisme de diffusion et synchronisation de type ET du bus de contrôle local.
3. La même remarque s'applique au bus de contrôle inter-modules pour la détection de rendez-vous entre les tandems.

5.8.2 Activation des tables de commande et de routage

La faible taille des tables de commande (trente-deux octets pour le C004 et soixante-douze pour le crossbar Supernode) permet de les préinstaller avec le code de l'algorithme de reconfiguration dans la mémoire locale des transputers qui pilotent les crossbars.

Le travail de reprogrammation du réseau revient alors à charger dans les crossbars puis activer un nouveau jeu de tables de commandes à partir de cette mémoire.

Un crossbar conçu pour la reconfiguration synchrone dynamique devrait permettre de dissocier les deux opérations. La durée du préchargement pourrait alors être masquée en effectuant celui-ci pendant le déroulement de l'étape. Lors de la transition, le réseau pourrait ainsi être reconfiguré quasi-instanément, par diffusion aux crossbars d'un simple signal d'activation des nouvelles tables.

Du point de vue de la mise en œuvre de la reconfiguration synchrone dynamique, le choix du C004 et la conception du crossbar Supernode sont criticables puisqu'aucun de ces deux circuits n'offre une telle fonctionnalité.

L'essentiel du délai de commande du réseau provient donc du vidage séquentiel du contenu des tables dans les crossbars.

Nœud simple : Dans un nœud simple, les crossbars Supernode sont pilotés via le bus mémoire externe du transputer contrôleur. Chaque commutation d'une sortie demande six accès[2] à celui-ci, soit environ une microseconde.

Chaque cycle d'accès ne permettant de piloter qu'un des deux crossbars, ce délai doit être doublé et multiplié par le nombre de liens ; soit un temps total de reconfiguration voisin de cent cinquante microsecondes.

Tandem : Les deux contrôleurs d'un tandem travaillant en parallèle chacun sur une paire de crossbars, cette valeur reste valable pour le tandem. La synchronisation entre les deux contrôleurs (via le bus de contrôle ou un lien) rajoute cependant quelques microsecondes.

Deuxième niveau : La reprogrammation d'une sortie exigeant trois octets, la reconfiguration complète d'un C004 exige le transfert d'une centaine d'octets sur le lien de commande.

Les T212 étant capables de soutenir des transferts simultanés à un mégaoctet par seconde sur leurs liens, les cartes de commutations peuvent être reconfigurées (simultanément) en cent microsecondes.

[2]Pour les machines protoypes du projet uniquement ; les versions commerciales ayant ètè optimisées de ce point de vue.

A cette durée doit encore être ajoutés le délai de propagation des signaux de début et de fin de reconfiguration entre les cartes de commutation et le superviseur, via le contrôleur de deuxième étage et la hiérarchie de bus de contrôle.

Tables de routage Les tables de routage peuvent également être préinstallées sur les transputers de calcul et activées par le signal de reprise du calcul en fin de reconfiguration.

5.8.3 Reprise du calcul et durée totale

L'exécution de la nouvelle étape de calcul suppose enfin la diffusion d'un signal de fin de reconfiguration.

Ce signal pourra transiter par la même voie que pour la détection de fin d'étape : il suffit d'utiliser le mécanisme de diffusion-synchronisation ET du bus de contrôle.

Dans une machine Supernode à deux niveaux, le scénario d'une transition d'étape sera donc le suivant :

1. Détections successives de fin d'étapes par :
   (a) par chaque transputer de travail, puis
   (b) par chaque contrôleur maître de tandem (qui propage au passage le signal au contrôleur esclave) et enfin
   (c) par le contrôleur superviseur de la machine ; l'ordre de reconfiguration étant ensuite transmis de celui-ci aux cartes de commutation via le contrôleur de deuxième étage et la hiérarchie de bus de contrôle.
2. Reconfiguration simultanée des deux niveaux de commutation.
3. Emission en retour vers le superviseur d'un signal de fin de reconfiguration par :
   - les cartes de commutation (même chemin qu'en 1) et
   - les contrôleurs de tandem (toujours via le bus de contrôle intermodules).
4. Diffusion groupée du signal de reprise de calcul du superviseur aux contrôleurs de tandem ; puis de ceux-ci aux processeurs de travail.

Si l'on tient compte du délai de reprogrammation évalué précédemment et des huit primitives de diffusion/synchronisation ET exécutées sur les bus de contrôle aux étapes 1, 3 et 4, la durée totale d'une transition d'étape sera d'environ deux cents microsecondes.

Ce résultat appele deux commentaires :

- Ce délai ne représente que le temps de transfert d'une centaine d'octets de voisin à voisin et peut donc être considéré comme négligeable devant la durée des étapes de calcul.

- Moyennant une révision du bus de contrôle et une optimisation de l'interface de commande des crossbars, la reconfiguration pourrait se résumer à la propagation d'un simple signal et ne plus durer que quelques microsecondes.

  Cette optimisation peut présenter un intérêt pour des algorithmes de type systolique dans lesquels tous les processeurs communiquent au même moment. Une reconfiguration synchrone à chaque pas de calcul pourrait être envisagée et éliminer le routage.

Chapitre 6

RECONFIGURATION ASYNCHRONE

6.1 DEFINITION ET CLASSIFICATION DES METHODES

Revenons sur le principe de liaison bipoint à la demande abordé dans l'introduction.

Pour communiquer entre elles, les unités de gestion des protocoles (ou UGP) sont munies chacune d'une (lien de) sortie et d'une (lien de) entrée.

Par hypothèse (et ce pour éviter une coûteuse gestion dynamique de mémoire par la couche d'acheminement des messages), tout message délivré à une UGP sera consommé par celle-ci en un temps fini : celle-ci peut être qualifiée d'esclave de son lien d'entrée.

Toute UGP est par contre maître de son lien de sortie, puisqu'elle seule peut prendre l'initiative d'émettre un message et en définir la destination.

La simulation d'un réseau totalement maillé par routage de type "Store-and-Forward", qui procède par recopie des messages de proche en proche, a déjà été présentée dans le chapitre précédent.

La technique de liaison bipoint à la demande consiste au contraire à établir en cours d'exécution une connexion entre les UGP source et destination pour le transfert de chaque message et pour la durée de ce transfert.

Ces liaisons sont réalisées au travers d'un réseau de commutation qui, outre le réseau d'interconnexion des nœuds, à connectique fixe ou programmable, comprend également tous les dispositifs de commutation (multiplexeurs et démultiplexeurs appartenant ou non à des crossbars) éventuellement présents à l'intérieur de chaque nœud.

Nous baptiserons aiguilleur l'ensemble formé par le réseau de commutation et sa logique de commande (figure 6.1).

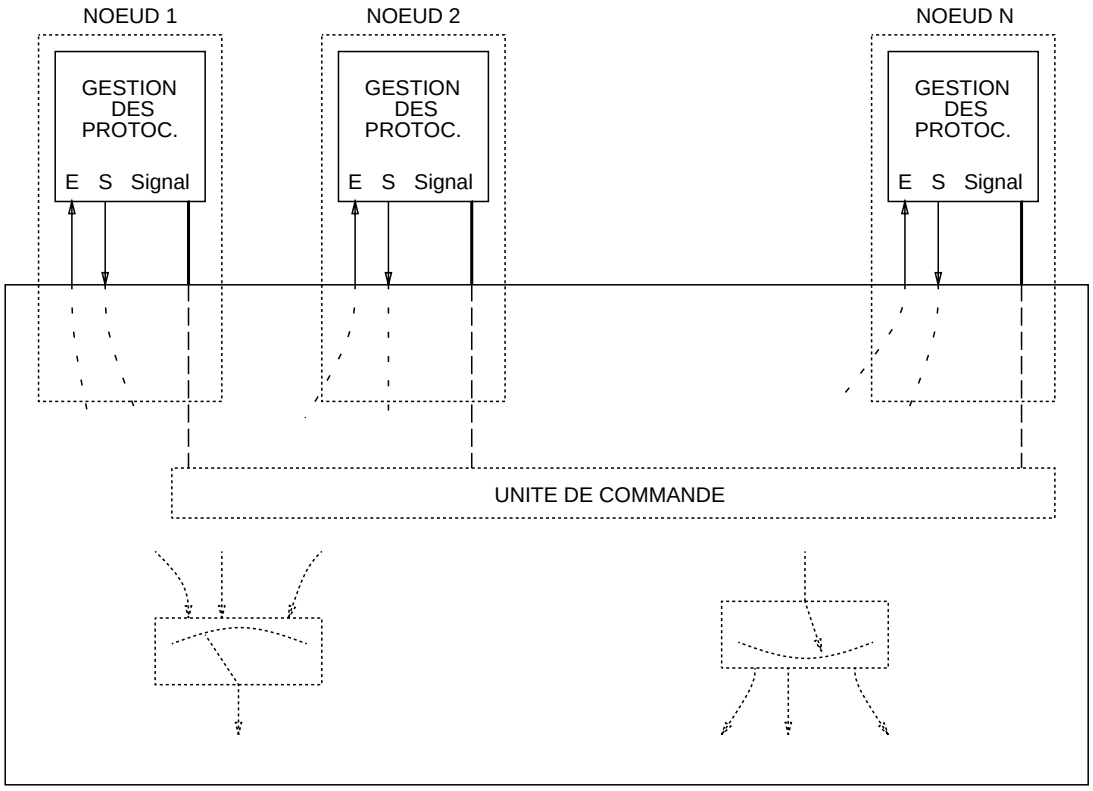


Figure 6.1: Notion d'aiguilleur

La gestion des connexions implique un dialogue entre l'UGP émettrice d'un message et l'unité de commande (UC) de l'aiguilleur :

1. L'UGP doit tout d'abord émettre une requête de création de connexion vers l'UC pour l'informer de la présence d'un message et de la destination de celui-ci.
2. L'UGP doit en retour être informée de l'instant à partir duquel la connexion a été établie (acquittement de la requête de connexion) et le transfert peut commencer.
3. L'UGP doit enfin informer l'UC de la fin du transfert (requête de destruction de connexion), qui peut alors libérer les ressources monopolisées par le transfert, à savoir :
   - le lien de sortie de l'UGP émettrice,
   - le lien d'entrée de l'UGP réceptrice et
   - le ou les liens correspondant du réseau d'interconnexion

   autorisant ainsi leur réutilisation pour d'autres transferts de messages.

L'acheminement des requêtes et des acquittements entre chacune des UGP et l'UC de l'aiguilleur suppose bien évidemment l'existence d'un support de communication.

Selon les cas, il s'agira soit d'une voie de signalisation physiquement séparée, soit des liens d'entrée et de sortie eux-mêmes, moyennant la mise en œuvre d'un protocole de multiplexage approprié.

Le coût de réalisation et la complexité de commande des réseaux de commutation utilisables pour la réalisation d'un aiguilleur croît avec le nombre d'UGP à interconnecter (c'est-à-dire avec le nombre de processeurs de la machine).

Nous envisagerons successivement l'utilisation :

- d'un réseau totalement maillé (RTM),
- d'un crossbar puis
- de réseaux de Clos à différents nombres d'étages.

Cette revue nous servira de support pour une définition plus précise de trois méthodes différentes d'acheminemenent des messages :

- routage "wormhole" déterministe,
- routage "wormhole" adaptatif et
- reconfiguration dynamique asynchrone

Nous montrerons que ces méthodes correspondent à des degrés croissant à la fois de temps de calcul de la commande et de dynamicité dans l'adaptation au traffic de messages généré par l'application ; à partir du même principe de liaison bipoint à la demande.

Dans un premier temps, nous établirons l'équivalence de l'aiguilleur à RTM avec le réseau d'interconnexion idéal dont nous dresserons auparavant le cahier des charges.

Nous montrerons ensuite en quoi le passage à un aiguilleur à crossbar ou à réseau de Clos est pénalisant, et dans quelles proportions.

6.1.1 Aiguilleur à RTM

Le critère de performance d'un réseau d'interconnexion est le délai d'acheminement d'un message, qu'il s'agit de minimiser et de rendre aussi indépendant que possible de la position relative des deux correspondants.

Considérons un message à transférer entre deux UGP respectivement source et destination (dont les voies respectives de sortie et d'entrée sont supposées libres de toute communication à l'instant considéré).

La première des deux composantes du délai d'acheminement est la durée du transfert proprement dit, soit la longueur du message divisée par la bande passante des liens empruntés.

Le reste du délai d'acheminement correspond à au délai d'établissement (puis de destruction) d'une connexion entre les deux UGP. Ce dernier recouvre à son tour :

- le temps de calcul du chemin, c'est-à-dire la détermination de la suite de liens à emprunter,
- les délais liés aux conflits d'accès à ces mêmes liens, lorsque ceux-ci ne sont pas disponibles immédiatement et
- le positionnement correspondant des dispositifs de commutation.

D'après ce critère, le réseau d'interconnexion idéal est un RTM composé de liens dont la bande passante individuelle est (supérieure ou) égale au débit de la mémoire locale de chacun des nœuds.

La commande d'un tel réseau ne nécessite en effet aucun calcul et ne souffre d'aucun conflit, puisque tous les chemins sont physiquement disponibles simultanément et en permence, et la vitesse de transfert n'est limitée que par la capacité d'entrée-sortie des nœuds et non plus par le réseau.

Transferts parallèles et séquentiels

Chaque nœud étant une machine séquentielle de type Von Neumann (dont la mémoire constitue le goulet d'étranglement), un tel réseau ne peut être que très largement sous-utilisé.

La bande passante cumulée des liens d'entrée (respectivement de sortie) d'un nœud excède en effet par définition le débit en écriture (lecture) de sa mémoire locale. Deux liens d'entrée (de sortie) ne peuvent donc pas être utilisés à la fois simultanément et chacun à sa pleine capacité.

Dans toute machine de taille significative, il est hors de question de munir les mémoires locales des nœuds d'autant de points d'accès que de processeurs, pour la seule connexion des liens.

La connexion des liens de sortie au port de lecture (autrement dit à la sortie de l'UGP) sera donc réalisée au travers d'un démultiplexeur. Réciproquement, l'entrée de l'UGP sera munie d'un multiplexeur auquel aboutiront les extrémités des liens d'entrée du nœud.

Les UGP des différents nœuds sont phyiquement indépendantes les unes des autres et sont en particulier susceptibles de générer simultanément (à elles toutes) plusieurs messages adressés à une même destination ; auquel cas l'accès à la voie d'entrée de l'UGP correspondante sera réclamé en même temps par plusieurs liens d'entrée de ce dernier.

Des précautions devront être prises pour la commande du multiplexeur de telle sorte que la politique d'allocation adoptée garantisse :

- l'exclusion mutuelle entre les liens d'entrée,

- l'absence de famine (pour garantir l'acheminement de tout message en un temps fini) et
- si possible l'équité (délai indépendant de la position relative des deux correspondants).

La plus simple et la moins coûteuse des politiques d'allocation répondant à ce cahier des charges est celle du "premier arrivé premier servi", qui n'exige que la gestion d'une file d'attente dont la taille est bornée par le nombre de processeurs de la machine.

La sortie pose quant à elle moins de problèmes, puisque chaque UGP n'émet qu'un message à la fois, dont la destination définit la commande à appliquer au démultiplexeur.

Rappelons au passage la présence indispensable d'un mécanisme d'acquittement pour le contrôle de flux, dont le rôle et le fonctionnement ont déjà été présentés au chapitre 3.

Deux stratégies de multiplexage peuvent être opposées : l'accès à la mémoire peut être attribué à un lien le temps de transférer seulement une information élémentaire (dont la taille est définie par le protocole d'acquittement) ou au contraire la totalité d'un message.

La première alternative (fonctionnement parallèle) autorise une pseudo-simultanéité des transferts sur les liens d'un même nœud ; chaque lien n'étant utilisé qu'à une fraction de sa bande passante.

Le deuxième mode de fonctionnement (séquentiel) se traduit au contraire par une exploitation des liens à leur pleine capacité, mais à tour de rôle.

La capacité d'entrée-sortie de chaque nœud peut être saturée dans les deux cas ; les deux modes de fonctionnement permettent donc d'acheminer un ensemble identique de messages dans le même délai[1].

Nous adopterons le mode de fonctionnement séquentiel.

Structure de l'aiguilleur

Nous sommes à présent en mesure de dessiner le squelette d'un aiguilleur à RTM (figure 6.2).

Celui-ci comporte deux sous-ensembles totalement symétriques chargés de l'acheminement respectivement des données et de leurs acquittements en sens inverse.

Pour chaque nœud, chacun des deux comprend un démultiplexeur (respectivement multiplexeur) sur la voie de sortie et un multiplexeur (respectivement démultiplexeur) sur la voie d'entrée de l'unité de gestion des protocoles ; le tout étant relié par un ensemble de liens formant un RTM.

[1] L'ordre d'exécution des transferts est cependant supposé choisi au mieux pour limiter les conflits entre les transferts de messages partageant la même destination.

L'unité de commande de chaque multiplexeur dialogue avec les UGP des autres nœuds et gère une file d'attente de leurs requêtes de connexions. Ce dialogue suppose, nous l'avons vu, l'existence d'une voie de signalisation.

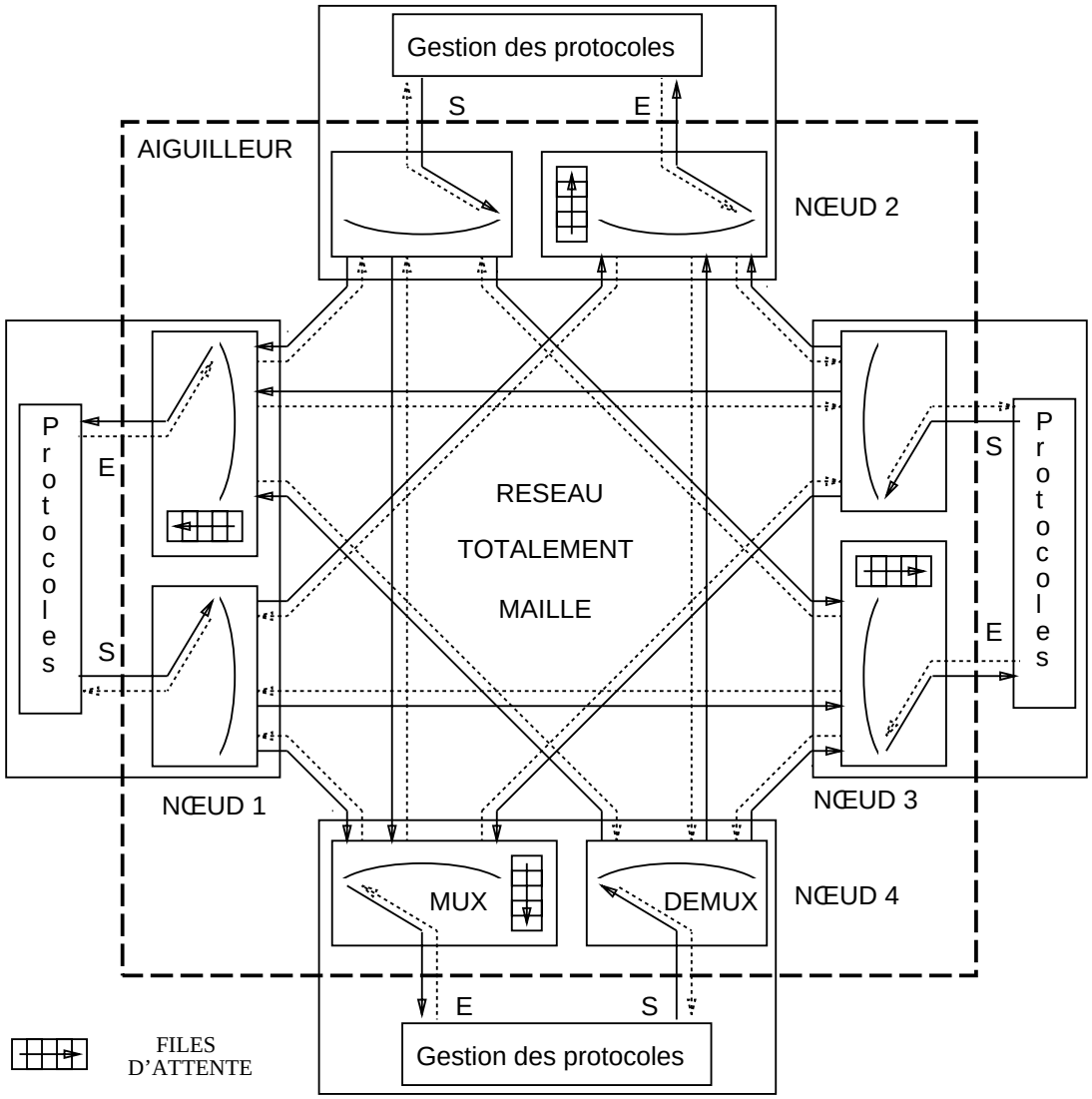


Figure 6.2: Structure d'un aiguilleur à RTM

Dans le cas présent, celle-ci peut être confondue avec le réseau de commutation :

- L'en-tête de chaque message (qui en encode la destination) jouera le rôle de requête de connexion.
- L'acquittement de celui-ci en retour sera interprété comme un acquittement de la requête, autorisant ainsi le transfert à débuter.
- La fin du message, qui peut être signalée par une marque spéciale ou

détectée par décomptage (la longueur du message étant encodée dans l'en-tête) et tiendra lieu de requête de destruction de connexion.

Pour la description de l'aiguilleur en pseudo-OCCAM, nous adopterons les conventions suivantes :

- Local désigne le nœud auquel est rattaché l'élément décrit.
- La voie de sortie de l'UGP du nœud local considéré est représentée par sortie_locale_d (voie d'émission des données) et sortie_locale_a (réception des acquittements en retour).
- Réciproquement, entrée_locale_d et entrée_locale_a correspondent à la voie d'entrée de cette même UGP.
- Enfin, les données émises par le nœud local vers un nœud i empruntent les liaisons données [i] et acquittement[i].

Unité de gestion des protocoles

Les messages délivrés à l'entrée de l'UGP locale sont consommés à mesure par celle-ci :

```
WHILE TRUE
  SEQ
    entree_locale_d ? dest;long
    entree_locale_a ! ANY
    SEQ i=0 FOR long
      entree_locale_d ? message[i]
      entree_locale_a ! ANY
    -- consommer le message
```

En émission, l'UGP peut mettre en œuvre un mécanisme de fenêtrage. La primitive d'envoi d'un message est alors la suivante :

```
sortie_locale_d ! dest;long
sortie_locale_d ? ANY
nb_emis := 0    -- compter les donnees
nb_recus := 0   -- compter les acquittements
WHILE (nb_recus < long)
   ALT
     sortie_locale_d ? ANY
       nb_recus := nb_recus + 1
     (nb_emis - nb_recus < fenetre) & SKIP
       SEQ
         sortie_locale_d ! message[nb_emis]
         nb_emis := nb_emis + 1
```

Le démultiplexeur

Le rôle du démultiplexeur est d'aiguiller le message sur le lien conduisant à sa destination :

```
WHILE TRUE
  SEQ
    sortie_locale_d ? dest;long
    donne[dest]!tampon
    PAR
      SEQ i=0 FOR long
        SEQ
          sortie_locale_d ? tampon
          donnee [dest]!tampon
      SEQ i=0 FOR long + 1
        -- dest et long a acquitter
        SEQ
          acquittement [dest] ? ANY
          sortie_locale_a ! ANY
```

Le multiplexeur

Lorsque le multiplexeur reçoit une requête de connexion et que la voie d'entrée de l'UGP locale est disponible, celle-ci est allouée au lien d'entrée et le transfert du message commence.

Pendant ce transfert, la voie d'entrée est occupée et les requêtes sont stockées dans la file d'attente.

La variable courant définit le numéro du lien d'entrée auquel est allouée la voie d'entrée de l'UGP ; et vaut NUL lorsqu'aucun transfert n'est en cours.

Les deux primitives de manipulation de la file sont les suivantes :

- **écrire_file (client)** ajoute un numéro client en queue de file et
- **lire_file ()** retire le client en tête de la file et en retourne le numéro ; retourne NUL si la file est vide.

```
SEQ
  courant := NUL
  WHILE TRUE
    ALT i=0 FOR nb_processeurs
      (i<> courant) & donnee[i] ? dest[i];long[i]
         -- traitement d'une requete
         IF
           courant = NUL
             -- libre : allouer
             SEQ
                courant := i
                initialiser_transfert()
```

```
      TRUE
        ecrire_file(i)
(courant <> NUL) & (nb_emis <=long) &
donnee[courant] ? tampon
  -- copie des donnees
  SEQ
    nb_emis := nb_emis + 1
    entree_locale_d ! tampon
entree_locale_a ? ANY
   -- comptage des acquittements
   -- et detection de fin
   acquittement[courant] ! ANY
   SEQ
     nb_recus := nb_recus + 1
     IF
       nb_recus = long[courant]
         SEQ
           courant = lire_file()
           IF
             courant <> NUL
               -- file non vide : reallouer
               initialiser_transfert()
```

La primitive initialiser_transfert recopie l'en-tete du message et réinitialise les compteurs de données et d'acquittement :

```
entree_locale_d ! dest[courant];long[courant]
nb_emis := 1  -- dest et long sont deja copies
nb_recus := 0 -- mais pas encore acquittes
```

Du RTM aux réseaux de commutation

Nous allons maintenant envisager le remplacement du RTM par un réseau de commutation et déterminer les caractéristiques idéales du réseau qui permettraient de conserver les mêmes performances.

En dehors des instants auxquels un transfert de message débute ou se termine, nous définirons le RTM comme étant dans un état stable.

Par définition du mode de fonctionnement séquentiel, tout nœud ne peut ni émettre ni recevoir plus d'un message à la fois.

Dans tout état stable, le graphe des communications en cours est donc un sous-ensemble d'une permutation de l'ensemble des nœuds et le RTM peut être remplacé par un réseau de commutation réarrangeable [2] programmé selon cette permutation.

[2]Réarrangeable : capable de réaliser toute permutation (rappel)

Cette propriété peut être aisément étendue aux transitions entre deux états stables dûes à une fin de transfert ; le nouveau graphe des communications en cours étant un sous-ensemble de l'ancien.

Il n'en va plus ainsi pour l'initialisation d'un nouveau transfert, qui rajoute au contraire un arc au graphe.

En l'absence de conflit (sources et destinations différentes deux à deux) entre le nouveau message et les messages en cours d'acheminement, un RTM est capable d'établir la connexion requise :

1. sans rompre les connexions en cours d'utilisation et
2. dans un délai limité à une consultation de la file d'attente et un accès aux registres de commande du multiplexeur et du démultiplexeur.

La première propriété définit clairement le réseau de commutation idéal destiné à remplacer le RTM comme non seulement réarrangeable, mais de plus non bloquant.

Nous envisagerons donc l'utilisation des deux réseaux de commutation entrant dans cette catégorie : le crossbar et les réseaux de Clos.

En l'absence de conflit sur les destinations des messages, le délai d'établissement de la connexion demandée résulte de la contribution de quatre phénomènes :

1. La première composante du délai est liée à la gestion des files d'attente utilisées pour résoudre les conflits.
2. La détermination du nouvel ensemble de commandes à appliquer aux dispositifs de commutation peut exiger divers calculs qui retardent d'autant l'établissement de la connexion.
3. L'application de ces commandes, c'est-à-dire leur écriture dans les registres de contrôle des (dé)multiplexeurs rajoute elle aussi un certain délai.
4. La dernière composante du délai n'est pas liée au réseau utilisé ni à son interface de commande, mais au temps de réponse intrinsèque des dispositifs de commutation. Compte tenu de la vitesse de réaction des commutateurs électroniques, ce dernier délai peut être négligé. Notons cependant qu'il peut en aller tout autrement avec certains commutateurs opto-électroniques.

Dans le cas du RTM :

1. Pour déterminer l'état (vide ou non) de la file d'attente, il suffit d'un accès mémoire (au pointeur de tête de file).
2. La destination du message définit par elle-même le numéro du lien de sortie de la source à utiliser et aucun calcul n'est nécessaire.

3. Satisfaire une requête exige seulement une écriture dans le registre de commande du démultiplexeur du nœud source (pour acheminer la requête) suivie d'une autre dans celui du multiplexeur du nœud destination (qui réalise effectivement la connexion).

Lorsque nous envisagerons la construction d'un aiguilleur à crossbar, nous montrerons que la constitution même de ce type de réseau implique un allongement des délais 1 et 3, proportionnel à la taille de la machine.

Enfin, le recours aux réseaux de Clos introduira une aggravation supplémentaire liée à la détermination des liens à utiliser pour réaliser la connexion (composante numéro deux du délai).

6.1.2 Aiguilleur à crossbar

Pour la résolution des conflits de destination entre les messages, deux caractéristiques de l'aiguilleur à RTM ci-dessus ont été mises à profit pour gérer efficacement les files d'attentes :

- Les multiplexeurs sont munis de ports de commande physiquement séparés.
- Il existe par définition une liaison bidirectionnelle (donnée et acquittement) entre chaque multiplexeur et la sortie de chaque nœud.

Ceci a permis de piloter les multiplexeurs par des unités de commande indépendantes travaillant en parallèle et gérant chacune une file d'attente. Le dialogue entre celles-ci et les UGP est alors supporté par le RTM lui-même qui cumule ainsi les fonctions de réseau de commutation pour les messages et de voie de signalisation.

Par contre, un aiguilleur à crossbar n'offre pas une telle facilité :

- En chaque nœud, les liens de sortie sont fusionnés en une unique voie de type un vers n [3] et le réseau de commutation ne peut plus jouer le rôle de voie de signalisation.
- Les ports de commande des multiplexeurs gérant les données (respectivement les acquittements) sont connectés en bus ; ce bus constituant l'interface de contrôle de l'ensemble du crossbar ainsi formé.

Du fait de l'absence d'un réseau de signalisation totalement maillé, un aiguilleur à crossbar n'est muni que d'une unité de commande centralisée gérant (séquentiellement) l'ensemble des files d'attente et des multiplexeurs (figure 6.3).

[3] Le démultiplexeur, devenu inutile, est d'autre part supprimé.

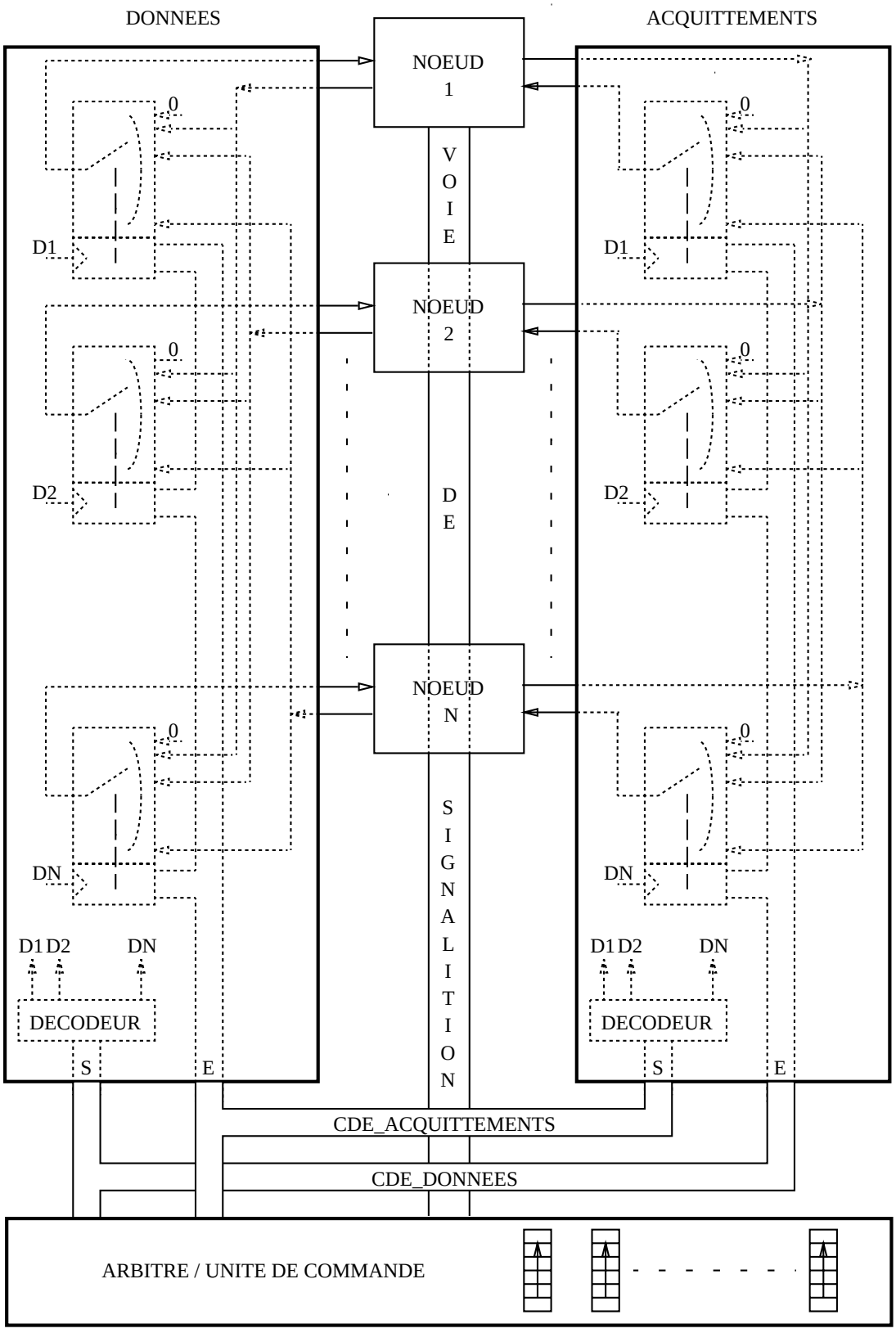


Figure 6.3: Structure d'un aiguilleur à crossbar

Cette unité de commande est connectée à l'interface de contrôle des crossbars (commutant respectivement les données et les acquittements) d'une part, et aux UGP par une voie de signalisation séparée d'autre part.

Pour limiter la connectique, au-delà de la dizaine de processeurs, la voie de signalisation prend la forme d'un bus (moyennant une politique équitable d'arbitrage d'accès entre les nœuds), plutôt qu'un faisceau convergent de liaisons bipoint bidirectionnelles individuelles convergeant vers l'unité de commande.

Evaluons le délai d'établissement d'un chemin en l'absence de conflit avec les transferts de messages en cours (sortie de l'UGP source et entrée de l'UGP destination simultanément libres) et comparons les deux aiguilleurs.

Nous supposerons que ce chemin n'entre en conflit avec aucun des messages en cours de transfert (sortie de l'UGP émettrice et entrée de l'UGP destinataire du message libres toutes les deux).

Dans les deux cas, la détermination des liens à emprunter est immédiate et n'exige aucun calcul : la source et la destination du message définissant par elle-même la commande à appliquer aux multiplexeurs et démultiplexeurs.

Les deux aiguilleurs sont également comparables quant au temps d'accès au registre de commande d'un (dé)multiplexeur, qui reste constant.

La différence majeure concerne donc la vitesse de résolution des conflits de destination entre les messages :

- Dans un aiguilleur à RTM, l'unité de commande du multiplexeur concerné ne gère que lui et peut donc entreprendre immédiatement le traitement de la requête de création (destruction) de connexion. Le délai se limite donc à la durée d'une consultation-mise-à-jour de la file d'attente.

- Dans un aiguilleur à crossbar, au contraire, l'unité centrale de commande traite séquentiellement les requêtes de l'ensemble des nœuds.

  Dans le cas le plus favorable où aucune autre UGP n'émet de requête simultanée, le délai sera le même que pour le RTM.

  Dans le pire des cas, les nœuds étant servis équitablement, le nœud émetteur devra attendre que l'unité de commande ait fini de traiter les requêtes de tous les autres nœuds.

Conclusion :

- Pour le RTM, le délai ne varie pas avec la distribution des requêtes au cours du temps.

- Pour tout autre réseau de commutation (en l'occurence, ici, le crossbar), le délai dépend du nombre de requêtes simultanées, donc de la fréquence de ces dernières ; et le délai maximal est proportionnel à la taille de la machine.

Interfaçage des crossbars

Les deux crossbars sont pilotés via deux bus notés E et S, sur lesquels les valeurs e et s seront écrites par la primitive connecter(e,s).

Pour le crossbar gérant les données, s désignera le multiplexeur à commander et e l'entrée sur laquelle celui-ci doit être positionné.

La valeur NUL désignera selon le cas :

- un multiplexeur fictif, ce qui revient à n'exécuter aucune commande ou
- une entrée toujours inactive, ce qui revient à déconnecter la sortie correspondante du démultiplexeur).

Les rôles de e et s sont permutés pour le crossbar commutant les acquittements.

Soit source $\neq$ NUL la sortie d'une UGP émettrice et soit destination $\neq$ NUL l'entrée d'une UGP réceptrice. Par définition :

- **connecter (NUL,NUL)** est sans effet,
- **connecter (source,destination)** réalise une connexion complète (donnée plus acquittement) entre source et destination,
- **connecter (NUL,destination)** déconnecte la voie de donnée de destination et
- **connecter (source,NUL)** déconnecte la voie d'acquittement de source.

Le bus de signalisation

Le bus de signalisation assure le transport des requêtes et de leurs acquittements entre les unités de gestion des protocoles et l'unité de commande de l'aiguilleur.

Pour chaque requête transportée, l'unité de commande reçoit les informations suivantes :

- le type de la requête (création ou destruction de connexion),
- la destination du message qui a donné lieu à cette requête et
- l'identité de l'UGP émettrice.

Les acquittements circulant en sens inverse sont de simples signaux en tout ou rien, et seule l'identité de l'UGP destinataire est transportée sur le bus.

Pour la description de l'algorithme exécuté par l'unité de commande, les conventions suivantes ont été utilisées pour modéliser l'accès au bus :

- Une UGP déclenche l'émission d'une requête par la primitive générer.
- La primitive écouter permet à l'unité de commande d'attendre qu'une requête soit émise et d'en acquérir le contenu.
- La génération et l'attente des signaux d'acquittement fait l'objet des primitives acquitter et attendre_acquittement.

La séquence exécutée par une UGP pour l'acheminement d'un message vers une destination d est alors modifiée comme suit :

```
emettre_requete (CREATION,d)
attendre_acquittement
emission du message sur sortie_locale
emettre_requete (DESTRUCTION,d)
```

Note : La requête de destruction de connexion, n'étant pas suivie d'un transfert de message, ne nécessite pas d'acquittement.

Algorithme de commande

Le nouvel algorithme de commande est assez proche de celui de l'aiguilleur à RTM. Les deux modifications principales sont les suivantes :

- La gestion du contrôle de flux (données/acquittement) pendant le transfert n'est plus du ressort de l'unité de commande.
- L'unité de commande gère l'ensemble des files d'attente via les nouvelles primitives de manipulation suivantes :
  - **écrire_file (d,client)** ajoute client en queue de la file associée à la destination d ;
  - **tête_file (d)** retourne la tête de cette même file (NUL si la file est vide) et
  - **progresser_file (d)** retire le client en tête de file et retourne la nouvelle valeur de la tête (NUL si la file ne contenait qu'un seul client).

```
WHILE TRUE
  SEQ
    ecouter (type,dest,ugp_cliente)
    IF
      type = CREATION
        SEQ
          IF (tete_file(dest)=NUL)
            PAR
              connecter (ugp_cliente,dest)
              acquitter (ugp_cliente)
```

```
    ecrire_file (dest,ugp_cliente)
type = DESTRUCTION
  SEQ
    PAR
      connecter (ugp_cliente,NUL)
      suivant = progresser_file(dest)
    PAR
      connecter (suivant,dest)
      IF suivant <> NUL
        acquitter (suivant)
TRUE
  -- erreur sur le type de requete
```

6.1.3 Au-delà du crossbar : routage et reconfiguration

Les contraintes technologiques ne permettent pas d'envisager la construction d'aiguilleurs à RTM ou à crossbar pour plus de quelques dizaines de processeurs.

Au-delà, les aiguilleurs sont donc reliés entre eux de manière à former un réseau d'interconnexion de la taille requise. Un message est alors susceptible de traverser plusieurs aiguilleurs pour rejoindre sa destination.

Tout réseau d'interconnexion de taille significative sera donc formé par association de plusieurs aiguilleurs et nous distinguerons trois stratégies différentes d'interconnexion et de commande de ceux-ci.

Routage déterministe

La première stratégie a pour objectif l'optimisation du délai d'établissement d'un chemin lié à la seule commande du réseau :

- Le calcul des chemins en cours d'exécution est remplacé par la consultation de tables précalculées ; tous les messages émis par une même source vers une même destination empruntant le même chemin.
- Le réseau est composé d'aiguilleurs à RTM ; la gestion en parallèle des files d'attente optimisant le délai de mise à jour de celles-ci.

Lorsque, de plus, les aiguilleurs :

- possèdent d+1 liens chacun et
- sont connectés de manière biunivoque chacun à une UGP,

nous retrouvons une structure classique de machine à connectique fixe, chaque nœud de degré d correspond à une paire (UGP, aiguilleur).

Ce cas de figure peut être étendu aux machines à connectique figée. Il s'agit des machines exploitant la reconfiguration synchrone, dans lesquels les aiguilleurs qui n'appartiennent pas un nœud de calcul sont commandés statiquement pour la durée de l'application ou d'une étape de calcul (reconfiguration synchrone dynamique).

Nous avons déjà étudié cette stratégie sous le nom de routage déterministe (de type "wormhole") et, à une consultation de table de routage près, un aiguilleur à RTM n'est rien d'autre qu'un routeur déterministe.

Notons cependant que le faible coût (délai) de commande du réseau a pour contrepartie l'absence de tout mécanisme de prise en compte dynamique (en cours d'exécution) de l'état d'occupation des liens ; ce qui rend l'efficacité du routage déterministe tributaire de la localité des communications et de la qualité du placement des tâches sur les processeurs.

Reconfiguration dynamique asynchrone

Conserver le principe de liaison bipoint dynamique à la demande en passant d'un crossbar à un réseau (multi-étages de Clos) non bloquant est une autre possibilité.

Cette méthode présente en fait des caractéristiques opposées à celles du routage déterministe.

Pour le choix d'un chemin, les conflits d'accès sont évités en n'utilisant que les liens libres au moment où la requête est traitée.

Le réseau étant non bloquant, le délai d'établissement d'un chemin est borné dès lors que la source et la destination du message à acheminer diffèrent de celles des messages en cours de transfert, et ce quelque soit la durée de chacun de ces transferts.

Cette propriété intéressante a une double contrepartie :

- Ainsi que l'étude de l'aiguilleur à crossbar a permis de le montrer, la gestion des files d'attente doit être centralisée, donc séquentielle.
- Pour toute paire (entrée,sortie), un réseau de Clos multi-étages offre plusieurs chemins possibles et, d'une paire à l'autre, ces chemins ne sont pas disjoints.

  Contrairement au cas de l'aiguilleur à crossbar, la détermination d'un chemin pour un message :

  - n'est plus immédiate, mais demande un certain calcul et
  - exige une vue globale du réseau : les messages doivent donc être traités séquentiellement.

Nous désignerons cette méthode sous le nom de reconfiguration :

- dynamique puisqu'intervenant en cours d'exécution et

- asynchrone, la mise à jour de la connectique du réseau étant effectuée incrémentalement pour chaque message, sans affecter les communications en cours.

Du fait du fonctionnement séquentiel, le délai moyen d'établissement d'un chemin est directement lié à la fréquence des requêtes de création ou destruction de chemin.

Du point de vue communication, les applications propices à la mise en œuvre de la reconfiguration dynamique asynchrone se caractérisent ainsi par :

- l'abcence de localité exploitable (pouvant être prise en compte pour le placement des tâches) dans les communications (susceptible d'entraîner de nombreux conflits d'accès aux liens) et par

- des échanges de messages longs et peu fréquents, ce qui réduit la fréquence des requêtes à traiter, donc le délai apparent d'établissement d'un chemin.

Le prix à payer pour l'élimination de tout conflit d'accès aux liens est un délai maximal de commande du réseau proportionnel au nombre de processeurs du MSMC (au lieu du diamètre du réseau pour le routage déterministe).

Routage adaptatif

Tout comme la première méthode, le routage adaptatif a déjà été évoqué au chapitre 5.

Rappelons que, tout comme la reconfiguration dynamique asynchrone, le routage adaptatif a pour but la réduction des conflits d'accès aux liens entre les messages.

Pour ce faire, une recherche du chemin optimal est effectuée en cours d'exécution et pour chaque message, à partir d'une vue plus ou moins partielle de l'état du réseau. Le choix est généralement contraint par une liste précalculée de chemins autorisés.

Cette recherche revient à explorer un arbre dont les premiers fils sont les liens de sortie du routeur connecté à la source du message. Le caractère plus ou moins adaptatif du routage est lié à la profondeur de l'arbre de recherche. Celle-ci est elle-même liée à la précision suivant laquelle l'état du réseau est connu (c'est-à-dire au nombre de liens examinés).

Cette méthode peut être considérée comme l'unique méthode d'acheminement des messages dont les deux précédentes constituent les deux cas extrêmes opposés :

- Le routage déterministe correspond à une profondeur nulle de l'arbre de recherche. La seule information prise en compte en cours d'exécution est alors l'état libre ou occupé du prochain lien du chemin prédéfini.

- La reconfiguration dynamique asynchrone revient au contraire à étendre l'arbre de recherche à l'ensemble du réseau et à dimensionner celui-ci de telle sorte qu'il soit non bloquant.

Le routage adaptatif offre ainsi un compromis entre les objectifs et les performances du routage déterministe et ceux (opposés) de la reconfiguration dynamique asynchrone.

Le coût relatif de la commande du réseau et des conflits d'accès aux liens est modulé en jouant sur la profondeur de l'arbre de recherche.

Le délai maximal d'établissement d'un chemin (hors conflit) est proportionnel au produit :

- du diamêtre du réseau (cas analogue au routage déterministe) et

- du délai maximal de commande de chaque aiguilleur traversé, qui dépend du nombre de liens à examiner (profondeur et degré de l'arbre).

Nous n'envisagerons que le cas le plus simple d'une exploration d'un seul niveau de l'arbre, ce qui revient à ce que chaque aiguilleur n'examine que ses propres liens de sortie.

Bien que toute l'information nécessaire au choix du lien de sortie soit disponible sur place (à l'intérieur de l'aiguilleur), cette localité ne peut être mise à profit pour accélérer la gestion des files d'attente.

La différence par rapport au routage déterministe réside en effet en ce qu'un même lien d'entrée peut réclamer plusieurs liens de sortie à la fois. Partageant des requêtes multiples communes, les files d'attente ne peuvent être gérées que de manière (centralisée et) séquentielle (et non plus en parallèle par des unités de commande indépendantes).

Ainsi, bien que répondant à des contraintes d'origines distinctes, un routeur adaptatif simple[4] ressemble beaucoup à un aiguilleur à crossbar autonome[5] exploitant la reconfiguration dynamique asynchrone :

- Au regroupement des canaux Rep et Req en une voie de signalisation près, l'organisation physique d'un aiguilleur à crossbar (figure 6.3) correspond à celle du routeur adaptatif présenté au chapitre 5 (figure 5.1).

- Dans les deux cas, le mode de fonctionement est séquentiel et se traduit par un coût maximal de commande proportionnel au nombre de liens d'entrée.

[4]Exploration à un seul niveau

[5]Non inséré dans un réseau de Clos.

Conclusion

L'analyse de la notion de liaison bipoint à la demande met en évidence l'existence d'une forme générique de routage adaptatif de type "wormhole"[6] englobant toutes les méthodes d'acheminement des messages par commutation de circuit.

Le choix d'un chemin en cours d'exécution peut tenir plus ou moins compte de l'état réel d'occupation des liens à l'instant où la requête est traitée.

Suivant la profondeur de l'arbre de recherche d'un chemin (composé de liens libres), cette forme générique se décline en différentes variantes correspondant à autant de compromis possibles entre les coûts respectifs de calcul du chemin et des conflits d'accès aux liens entre les messages :

- Le routage déterministe et le routage adaptatif limité à l'examen des seuls liens de sortie de l'aiguilleur, déjà étudiés au chapitre 5 , occupent l'extrémité du spectre qui privilégie la vitesse de commande.
- A l'autre extrémité du spectre, la reconfiguration dynamique asynchrone pousse au contraire l'adaptation jusqu'au point extrème où tout conflit est éliminé.

  La recherche des chemins dans un hypercube du paragraphe 5.5.1 en est une forme approchée ; mais le caractère bloquant de l'hypercube se traduit par des problèmes de terminaison de l'algorithme.

6.2 RECONFIGURATION ASYNCHRONE

L'analyse précédente nous a permis de définir la reconfiguration dynamique asynchrone (RDA) et de la situer parmi les autres méthodes d'acheminement des messages.

La suite logique de l'étude consiste à évaluer la RDA en tant que méthode principale (et exclusive) d'acheminement des messages.

Nous allons donc revenir sur les contraintes inhérentes à la RDA pour déterminer si celle-ci peut être appliquée avec profit à des machines de type Supernode et, le cas échéant, en délimiter le domaine d'utilisation.

6.2.1 La condition d'absence de blocage

La RDA suppose par définition le recours à un réseau de commutation non bloquant, d'où la question évidente : la construction d'un tel réseau est-elle envisageable, et jusqu'à quel nombre de processeurs ?

[6] Ou similaire, par opposition au "store-and-forward" et à la commutation de paquets.

La réponse à cette question nous ramène au chapitre 2 : nous avions conclu à une taille limite de l'ordre de quelques milliers de processeurs au-delà de laquelle le coût de construction d'un réseau de Clos non bloquant devient prohibitif.

La RDA ne peut donc constituer l'unique mécanisme d'acheminement des messages des (éventuelles) futures machines parallèles à très grande échelle.

Ce nombre correspond par contre à la taille limite des machines de type Supernode susceptibles d'être construites dans un avenir proche, pour lesquelles l'étude de la RDA garde tout son intérêt.

6.2.2 Longueur de reconfiguration

Un autre facteur affectant la taille des MSMC susceptible de bénéficier de la RDA est le coût de commande du réseau associé à cette dernière.

Le fonctionnement séquentiel résultant de la recherche d'un chemin optimal implique en effet un surcoût qui dépend de la fréquence des requêtes.

La notion de longueur de reconfiguration va nous permettre de mieux apprécier (quantitativement) ce surcoût et de délimiter le domaine d'application de la RDA.

Nous supposerons pour simplifier que chaque paire de tours de boucle de l'algorithme de commande (connexion puis déconnexion) est exécuté en un temps T par l'unité de commande (ou serveur de connexion).

Considérons un nœud E émettant une suite ininterrompue de messages de même longueur L (donnant lieu chacun à une reprogrammation de la connectique du réseau) et des liens de bande pasante B.

Le délai apparent D d'établissement d'un chemin vu par E est l'intervalle de temps séparant la fin du transfert d'un message du début de celui du suivant.

Lorsque E est le seul émetteur, ses requêtes sont traitées immédiatement par l'unité de commande, d'où $D_{optimal} = T$.

Suppsons maintenant que E soit imité par les N-1 autres nœuds du multiprocesseur.

Après établissement de la connexion, E transfère son message puis s'adresse à nouveau à l'unité de commande pour le message suivant. Pendant le transfert d'un message, celle-ci s'occupe des autres nœuds :

$$D = D_{optimal} + Max((N-1)T - \frac{L}{B}, 0) \qquad (6.1)$$

La longueur de reconfiguration L_{reconf} est la longueur minimale des messages qui permet masquer le fonctionnement séquentiel de la commande et rend la RDA comparable au routage déterministe :

$$L_{reconf} = (N-1)BT \tag{6.2}$$

La longueur de reconfiguration définit ainsi un critère quantitatif d'évaluation qui, par comparaison avec la longueur moyenne des messages à transférer, permet d'apprécier le degré d'adéquation de la reconfiguration asynchrone aux besoins des applications considérées.

6.2.3 Utilisation du crossbar

Application au nœud simple et au tandem

La mise en œuvre de la RDA sur les machines Supernode à un seul niveau de commutation à partir de l'étude de l'aiguilleur à crossbar est relativement immédiate.

En effet, à chaque élément de l'aiguilleur correspond un composant physique d'un nœud simple ou d'un tandem (aux anomalies de la structure de contrôle près) dont le réseau de commutation Est-Ouest resterait inutilisé :

Aiguilleur :	Supernode :
• Nœud ou processeur	• Transputer de travail
• Lien de sortie	• Lien Nord
• Lien d'entrée	• Lien Sud
• Crossbar gérant les données	• Crossbar Nord vers Sud
• Crossbar des acquittements	• Crossbar Sud vers Nord
• Unité de commande centrale	• Transputer de contrôle
• Voie de signalisation	• Bus de contrôle
• Bus de commande E et S	• Bus mémoire externe du transputer de contrôle

Le temps T mis par le transputer de contrôle pour effectuer deux tours de boucle (connexion + déconnexion, algorithme du 6.1.2) est approximativement égal au cumul des durées des primitives exécutées.

Ecouter correspond au mécanisme d'appel du maître offert par le bus de contrôle. Le délai correspondant a été évalué à environ 22 microsecondes pour un tandem complet (détail dans l'annexe C).

Nous avons également évalué le coût du mécanisme de réponse à l'esclave (déclenché par la primitive acquitter) à 11 microsecondes.

Sur les machines Supernode prototypes disponibles au L.G.I., la procédure connecter revient à réaliser une douzaine d'accès aux registres de

commande des crossbars, soit approximativement trois microsecondes.

Evaluons le coût de manipulation des files d'attente à partir d'un exemple de mise en œuvre. L'accès aux files par l'unité de commande présente trois particularités :

- un retrait ne peut porter que sur une requête en tête de file ;
- réciproquement, tout ajout est effectué en queue de file ;
- aucune requête n'est partagée par plusieurs files à un instant donné ;

d'où le recours à un chaînage simple des requêtes.

L'ensemble des files peut être représenté en mémoire sous la forme de trois tableaux comportant autant d'entrées que de processeurs selon les conventions suivantes :

- tête [d] et queue [d] retournent respectivement le (numéro du client occupant) début et la fin de la file associée à la destination d.
- chaînage [c] retourne le client suivant c dans la file à laquelle il appartient.

La traduction des primitives est alors la suivante :

```
ecrire_file (dest,client)
SEQ
  PAR
    -- premiere phase : lecture
    debut = tete [dest]
    fin = queue [dest]
    -- deuxieme phase : mise a jour
    PAR
      queue [dest] = client
      IF
        (debut = NUL)
          -- cas d'une file vide
          tete [dest] = client
        TRUE
          -- normal : chainer client
          chainage [fin] = client


progresser_file (dest)
  SEQ
    -- premiere phase
    debut = tete [dest]
    -- deuxieme phase
    IF (debut <> NUL)
      debut = chainage [debut]
    retourner (debut)
```

```
tete_file (dest)
  SEQ
    retourner (tete [dest])
```

Une évaluation grossière d'après le nombre de tests (1) et d'accès aux tableaux (4) nous donne un temps d'exécution par le transputer de contrôle de l'ordre de 2 microsecondes pour la primitive écrire_file. Pour les deux autres primitives, les mêmes critères conduisent à diviser ce dernier délai respectivement par 2 et 4.

La durée T d'un double tour de boucle par l'unité de commande est proche du délai d'accès au bus de contrôle, soit 80 microsecondes et pour un tandem complet, nous obtenons une longueur de reconfiguration d'environ 5 kilooctets[7].

La RDA s'avère ainsi très inefficace en temps que méthode principale d'acheminement des messages, son domaine d'utilisation se limitant aux applications effectuant des échanges de blocs mémoire de taille importante.

Pour préciser cette limite, considérons à titre d'exmple un tandem à 64 transputers placé dans les conditions du paragraphe 6.2.2[8], dans les deux modes de fonctionnement suivant :

1. reconfiguration dynamique asynchrone et
2. configuration statique en tore 8 par 8 avec routage déterministe

et déterminons la longueur minimale L_{min} des messages à partir de laquelle la reconfiguration devient plus efficace que le routage.

Calculons donc l'équivalent du délai d'établissement d'un message dans le cas du routage [9].

Dans le pire des cas, le nombre de liens traversés par un message est égal au diamètre du réseau d'interconnexion (soit $\sqrt{N}$ pour un tore).

Chaque traversée d'un routeur (programmé) déclenche l'exécution d'un algorithme de routage d'une durée R de l'ordre de la vingtaine de microsecondes (paragraphe 5.7.2).

Dans le pire des cas, cette durée doit être multipliée par le nombre l de liens taités par le routeur (4 liens physiques et un pseudo-lien vers l'unité de gestion des protocoles), à laquelle s'ajoute le temps de transfert proprement dit, soit $\frac{L}{B}$, d'où :

$D_{routage} = (\frac{L}{B} + Rl)\sqrt{N}$

soit (B = 1 Mo/s, R = 20 μs, N=64 et l=5) :

[7] T = 80 microsecondes, N = 64 processeurs, B = 1 mégaoctet / seconde

[8] Emission par chacun des nœuds d'une suite ininterropmpue de messages de même longueur L.

[9] Le transputer imposant un routage de type "Store-and-forward", il n'y a pas réellement établissement d'un chemin.

$D_{routage} = 8L + 800$

Pour la reconfiguration, la formule 6.1nous donne :

$D_{reconf} = 85 * 64 - L$

La combinaison de ces deux dernières équations nous donne une longueur de message de :

$L_{\min} \simeq 520 octets$,

soit un net avantage en faveur du routage.

Les optimisations

D'après les résultats ci-dessus, la RDA s'avère très inefficace et ne présente un intérêt que pour les transferts de gros blocs mémoires.

L'interprétation de ces résultats doit cependant tenir compte de l'histoire du projet Supernode. La structure de contrôle des machines Supernode a en effet été conçue à partir d'un cahier des charges initial prévoyant seulement une reconfiguration de type statique et un acheminement des messages par routage.

L'inefficacité de la RDA sur les machines Supernode peut ainsi être attribuée en grande partie à une inadéquation de cette structure de contrôle qui s'avère largement sous-dimensionnée et mérite une refonte complète.

Examinons donc quelles optimisations peuvent être apportées à chacun des éléments de cette structure de contrôle et évaluons les gains de performance susceptibles d'être obtenus.

Bus de contrôle :

Plus de la moitié du temps de traitement d'une requête par le transputer de contrôle est consacrée à la gestion de la voie de signalisation (c'est-à-dire du bus de contrôle).

La vitesse de fonctionnement de ce bus peut être améliorée en trois points :

1. Les contraintes de fonctionnement ne justifient pas l'emploi de circuits de type IEEE 488 qui limitent sévèrement le débit du bus (cycle de trois microsecondes). L'emploi d'une technologie classique (TTL) autoriserait au contraire une vitesse comparable à celle des bus standards (genre VME), soit un temps de cycle inférieur ou égal à la centaine de nanosecondes.

2. L'arbitrage d'accès au bus et la détermination de l'origine des requêtes sont entièrement à la charge du transputer de contrôle. Chaque signal de requête donne lieu à l'exécution d'une (coûteuse) boucle de scrutation, l'ordre de balayage définissant la politique d'arbitrage d'attribution du bus.

   Ce travail peut avantageusement être confié à un arbitre de bus câblé retournant à chaque cycle l'origine de la requête éventuelle à traiter.

3. Le bus peut enfin être décomposé en une paire de voies unidirectionnelles véhiculant respectivement les requêtes et les acquittements , ce qui en double le débit.

Une telle optimisation autorise l'acquisition d'une nouvelle requête (origine et type plus destination) tous les deux cycles de bus (soit deux cents nanosecondes) et l'émission simultanée de deux acquittements.

Pilotage des crossbars :

Le bus de pilotage des crossbars peut être modifié de manière à accéder simultanément aux registres des deux crossbars en une seule écriture.

La primitive connecter est ainsi réduite à deux cycles d'écritures, soit une centaine de nanosecondes compte tenu du temps d'accès de ces registres.

Unité de commande :

La dernière optimisation consiste à câbler la gestion des files d'attente.

Les tableaux tête, queue et suivant peuvent en particulier être stockés dans des mémoires indépendantes accessibles simultanément.

Le coût de gestion des files est ainsi réduit à deux cycles d'accès à la mémoire par tour de boucle de boucle de l'algorithme de commande.

Les trois éléments de la structure de contrôle possèdent ainsi un débit comparable d'une requête toutes les deux cents nanosecondes, soit une longueur de reconfiguration beaucoup plus attractive de 0,4 octet par processeur.

En dehors des machines Supernode, mentionnons enfin le projet NECTAR d'interconnexion optique de systèmes hétérogènes, qui a donné lieu à la construction d'un dispositif nommé HUB, très proche de notre aiguilleur à crossbar. [3] cite un temps de traitement d'une connexion par ce HUB inférieur à une microseconde.

6.2.4 Utilisation du réseau de Clos

L'expérimentation de la RDA sur les machines Supernode à deux niveaux de commutation à des fins d'évaluation est dépourvue d'intérêt, la lenteur de la hiérarchie de bus de contrôle et le caractère bloquant du réseau de Clos utilisé faussant tous les résultats.

Nous nous contenterons donc de montrer comment déterminer efficacement la commande à appliquer à un réseau de Clos non bloquant pour établir le chemin voulu sans réduire le débit de l'unité de commande (par rapport à un crossbar).

Tous les réseaux de Clos à k+1 niveaux sont construits récursivement à partir d'un réseau à deux niveaux.

Il suffit donc de ne résoudre le problème que pour ce dernier et d'appliquer récursivement k fois l'algorithme obtenu pour déterminer le chemin

dans un réseau à k + 1 niveaux. La première passe permet de choisir un sous-réseau à k niveaux pour réaliser la connexion et le problème se repose alors dans les mêmes termes pour celui-ci.

Soit une connexion à réaliser entre une entrée e et une sortie s appartenant respectivement aux crossbars notés E et S du premier des deux niveaux.

Il faut déterminer lequel des q crossbars du deuxième niveau utiliser pour relier E à S.

Par définition, celui-ci appartient simultanément :

- à l'ensemble des crossbars du deuxième niveau dont le lien de sortie vers S libre et
- à l'ensemble de ceux dont le lien d'entrée connecté à E est inoccupé.

Le calcul se ramène donc à l'application :

- d'un opérateur d'énumération des solutions possibles, c'est-à-dire d'intersection des ensembles ci-dessus et
- d'un opérateur de sélection, c'est-à-dire une règle de priorité permettant le cas échéant de trancher lorsque plusieurs choix sont possibles.

Notons état_e[i][j] (respectivement état_s[i][j]) le booléen représentant l'état libre ou occupé du $j^{ième}$ lien d'entrée (respectivement de sortie) du $i^{ième}$ crossbar du deuxième niveau (o<= i <q).

D'après ces conventions, l'opérateur d'intersection se réduit à un simple produit logique des deux booléens.

Un opérateur de priorité simple consiste à considérer les crossbars dans l'ordre décroissant des numéros.

Une solution naïve consiste à appliquer d'abord l'opérateur de priorité. Nous obtenons ainsi un premier algorithme, séquentiel, donnant le rang r du crossbar à utiliser :

```
SEQ
  pas_trouve = VRAI
  r = q
  WHILE (r>0) AND pas_trouve
    r = r-1
    IF etat_e[r][E] & etat_s[r][S]
      SEQ
        pas_trouve = FAUX
        etat_e[r][E] = 0
        etat_s[r][S] = 0
        -- marquage des liens devenus occupes
```

Lors de la destruction de la connexion, les booléens devront être remis à jour par :

```
SEQ
  etat_e[r][E] = 1
  etat_s[r][S] = 1
```

Cet algorithme est aussi inefficace que simple, puisque le calcul exige q itérations dans le pire des cas ; or il est possible de réaliser le même calcul en temps constant moyennant le recours à des opérateurs câblés spécifiques.

Représentons l'état des liens par deux tableaux Entrée et Sortie d'entiers à q bits tels que :

$Entrée[i] = \sum_{j=0}^{q-1} état\_e[i][j].2^j$

$Sortie[i] = \sum_{j=0}^{q-1} état\_s[i][j].2^j$

Notons & et $\uparrow$ les opérations de ET et de OU exclusif bit à bit sur ces entiers.

Par définition, à chaque bit à un de $t = Entrée[E]\&Sortie[S]$ correspond un crossbar utilisable pour la connexion de e à s.

Le rang r de celui-ci est celui du poids du premier de ces bits, soit :

$r = partie\_entière(Log_2(t))$

Notons que le circuit combinatoire calculant cette expression n'est rien d'autre qu'un encodeur de priorité ordinaire.

La mise à jour des tableaux correspond alors à l'itération suivante :

$t' = 2^r$

$Entrée[E] = Entrée[E] \uparrow t'$

$Sortie[E] = Sortie[E] \uparrow t'$

Cette opération est également très simple : le calcul de 2^r n'exige qu'un démutliplexeur.

Ainsi, l'ensemble des calculs n'exige que cinq circuits combinatoires :

- un opérateur ET bit à bit,
- deux opérateurs OU exclusif bit à bit
- un démultiplexeur et
- un encodeur de priorité.

Le calcul du chemin peut donc être réalisé en temps constant en trois étapes :

1. Lecture des tableaux :

```
x = Entree [E]
y = Sortie [S]
```

2. Génération de t,r,t', x' et y' par les circuits combinatoires :

```
t = x & y
r = partie_entiere (Log2 (t))
t' = 2 puissance r
x' = x ^ t'
y' = y ^ t'
```

3. Mise à jour des tableaux :

```
Entrees [E] = x'
Sorties [S] = y'
```

Le recours à des mémoires à double accès permet de plus un travail en "pipeline" en réalisant simultanément les phases 1 et 3 de deux calculs successifs.

Le temps de calcul est ainsi réduit au délai de traversée des circuits combinatoires et à la durée de deux cycles d'accès mémoire (un seul si les deux tableaux sont installés dans des mémoires indépendantes).

L'estimation précédente (paragraphe 6.2.3) reste donc valide, soit une longueur de reconfiguration de 400 octets pour mille processeurs.

6.2.5 Conclusion

L'efficacité de la reconfiguration asynchrone est largement tributaire du support matériel offert par la machine cible, comme le montre la comparaison entre les machines Supernode et une machine optimisée (à construire).

Au prix d'un câblage poussé de l'unité de commande, la vitesse de pilotage d'un réseau de Clos peut être rendue comparable à celle du crossbar.

L'examen des résultats numériques ci-dessus montre cependant que, étant donné le surcoût qu'implique la commande séquentielle du réseau, l'efficacité de la reconfiguration asynchrone est sévèrement limitée.

La R.D.A. est donc essentiellement destinée aux transferts de gros messages, dont la répartition entre les processeurs ne peut pas être estimée statiquement, qui permettent d'amortir ce surcoût.

A titre d'illustration, la fin du chapitre présente (de manière très simplifiée) un exemple d'application de la reconfiguration asynchrone à des fins d'équilibrage dynamique de charge, dans le cadre de la synthèse d'images par lancer de rayon.

6.3 Reconfiguration et équilibrage de charge

L'étude d'une application suceptible de tirer parti des techniques de parallélisme massif a permis d'asseoir ce travail de recherche sur les communications dans les machines sans mémoire commune sur un exemple concret.

C'est dans cet esprit que j'ai entrepris conjointement de paralléliser la méthode dite du lancer de rayon et la conception d'une architecture parallèle optimisée pour celle-ci.

Le lancer de rayon offrait en effet deux avantages :

- un temps de calcul élevé et un degré de parallélisme important justifiant le recours à des architectures massivement parallèles et
- dans le cadre d'un contrat, le support technique et financier apporté par le C.C.E.T.T. ([37]), spécialiste du domaine et partenaire de ce projet.

Soit une scène tridimensionnelle (décrite comme un assemblage d'objets plus ou moins nombreux) éclairée par une ou plusieurs sources de lumière.

L'objectif est de reconstruire l'image de cette scène, à partir d'un point d'obervation donné, en tenant compte de tous les phénomènes de miroir et de transparence des objets.

Le principe consiste à reconstruire (en sens inverse) le trajet des rayons lumineux des sources jusquà l'œil de l'observateur, ce qui permet d'obtenir des images de très bonne qualité.

La couleur de chaque pixel de l'image est déterminée en "lançant" un rayon partant de l'œil de l'observateur et passant par le point correspondant de l'écran virtuel sur lequel est formée l'image (figure 6.4).

Le trajet du rayon et l'atténuation correspondante de la lumière est suivi, à travers les multiples réflexions et de réfractions des objets rencontrés, jusqu'aux sources lumineuses.

Les contributions respectives des sources peuvent ainsi être calculées et cumulées pour fournir l'intensité lumineuse du pixel (selon les trois couleurs fondamentales).

La méthode engendre de très nombreux calculs d'intersection entre les rayons et les objets ; et la génération d'une image peut prendre des heures de calcul sur une machine ordinaire.

Diverses techniques d'accélération basées sur un découpage (régulier ou non) de la scène ont été proposés.

Ces techniques ne permettent pas de réduire suffisamment le temps de calcul, d'où le recours aux machines parallèles ; les rayons, tout comme les objets, pouvant être traités indépendamment les uns des autres.

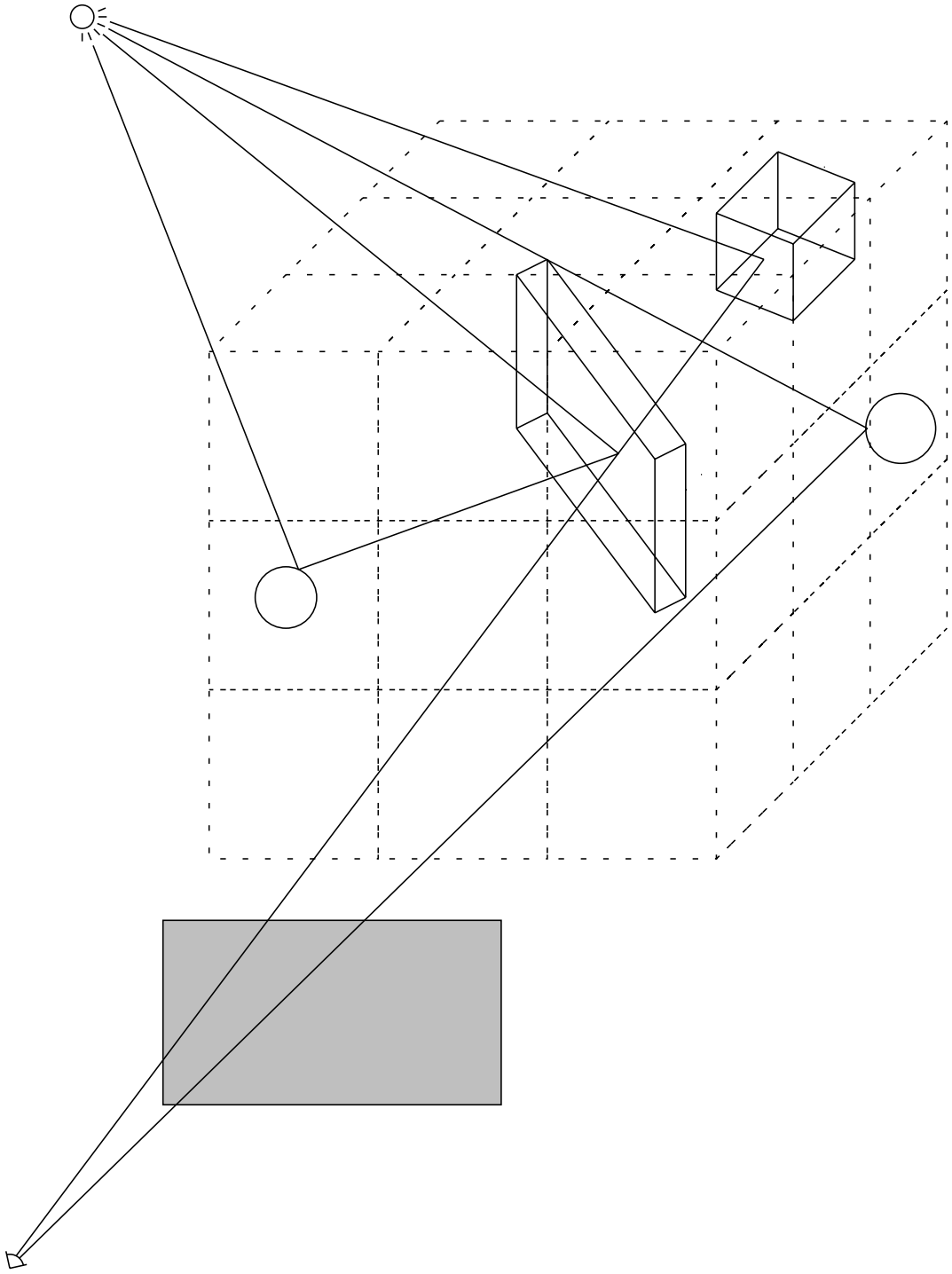


Figure 6.4: Principe du lancer de rayon

Plusieurs stratégies de parallélisation sont utilisables. L'une d'entre elles exploite un découpage régulier de la scène et une connexion des processeurs en grille à trois dimensions.

Il est alors facile d'associer chaque volume élémentaire issu du découpage au processeur occupant la position correspondante dans la grille.

Les trajets des messages entre les processeurs réflètent alors ceux des rayons dans l'espace de la scène. Les communications présentent d'évidentes propriétés de localité et peuvent être réalisées par routage déterministe.

La répartition de la charge entre les processeurs, fonction du contenu des régions, de la disposition des sources lumineuses et des phénomènes de réflexion et de réfraction, est difficile à évaluer statiquement.

L'équilibrage dynamique de charge est réalisé comme suit :

- La description d'une région traitée par un processeur saturé est temporairement dupliquée sur un processeur inactif.
- Le processeur surchargé soustraite une partie des rayons au processeur oisif et ce jusqu'à résorption du déséquilibre.

Pour éviter de perturber l'acheminement des messages (rayons) ordinaires, une connexion directe des processeurs pour la durée du couplage est des plus souhaitable, d'où le recours à la reconfiguration asynchrone.

Il s'agit en fait d'une variante de celle-ci : les destinations des messages sont déduites des évaluations périodiques de charge communiquées par les processeurs. Celles-ci remplacent les requêtes explicites de connexion.

Chaque nœud est muni d'un ou plusieurs liens additionnels dédiés à la régulation de charge.

Chapitre 7

Conclusion

Les communications entre les (unités de gestion des protocoles) des nœuds d'un multiprocesseur sans mémoire commune (MSMC) sont réalisées au travers d'un réseau de commutation, c'est-à-dire un ensemble de commutateurs (crossbars) reliés par des liaisons bipoint.

Les MSMC diffèrent en particulier par le type de réseau de commutation et par la méthode de routage (ou méthode de commande de ce réseau) utilisés.

Les MSMC les plus faciles à construire sont les machines à connectique fixe, dans lesquelles les crossbars sont reliés de manière biunivoque chacun à un nœud auquel il appartient.

Les nœuds contenant les crossbars sont reliés entre eux par des connexions (liens) fixes et directes choisies lors de la construction de la machine.

Le routage consiste à sélectionner pour chaque transfert de message la suite de liens à emprunter parmi les possibilités (chemins) offertes par le réseau et à positionner les crossbars en conséquence.

Le temps consacré à cette sélection pendant l'exécution de l'application est une caractéristique importante des méthodes de routage.

Le routage déterministe consiste à précalculer un chemin (unique) pour chaque paire de nœuds (source, destination), ce qui réduit ce temps à une simple consultation de tables de routage à une seule entrée par destination.

Dans les MSMC à connectique fixe et routage déterministe, des messages peuvent enter en conflit pour l'accès à un même lien, ce qui ralentit d'autant les communications.

Ces conflits peuvent être en particulier attribués à deux types d'utilisation non optimale des liens :

- Une topologie de réseau (fixée lors de construction de la machine) différente du graphe d'interconnexion souhaitable pour l'application se traduira par un allongement des chemins, d'où un multiplexage et une sollicitation accrus des liens.

- Le choix effectué parmi les chemins offerts par le réseau peut également être en cause. En effet, en cas de saturation (passagère ou non) des liens d'un chemin précalculé, le principe même du routage déterministe interdit l'emprunt par le message d'un autre chenin moins encombré.

  Les applications caractérisées par une répartition des flux de communication variant au cours de l'exécution et/ou difficile à estimer statiquement sont particulièrement susceptibles de pâtir de ce phénomène.

La reconfiguration synchrone remédie au premier cas de figure en remplaçant les liaisons bipoint fixes par un réseau de commutation réarrangeable dont la commande est mise à jour avant chaque étape de calcul.

L'adoption de topologies régulières qu'autorise la reconfiguration synchrone complique notablement la lutte contre l'interblocage. Ce dernier peut être prévenu par l'adoption de fonctions de routage à graphe de dépendance acyclique ; deux méthodes permettant d'exhiber de telles fonctions ont été présentées dans le chapitre 5.

Le mécanisme de rendez-vous à n participants offert par la voie de contrôle facilitant la détection des points de reconfiguration, le suppport matériel fourni par les machines Supernode, quoique largement perfectible, s'avère suffisant pour une mise en œuvre efficace de la reconfiguration synchrone.

La prise en compte de l'occupation des liens dans le choix final du chemin, reporté pour ce faire jusqu'au moment du transfert du message, est la caractéristique commune à toutes les variantes de routage adaptatif, susceptible de réduire le nombre de conflits d'accès aux liens relevant de la deuxiène catégorie.

L'optimisation des chemins est par contre obtenue au détriment de la vitesse de commande du réseau et l'importance relative accordée à ces deux paramètres est déterminée par la profondeur d'exploration de l'arbre des liens.

La reconfiguration dynamique asynchrone correspond au cas extrème de l'exploration de la totalité de l'arbre et de l'adoption d'un réseau de commutation non bloquant, ce qui permet d'élimer tout conflit d'accès aux liens.

La notion de longueur de reconfiguration permet de quantifier le coût de la recherche de chemins optimaux. Son calcul montre que la lenteur de la voie de contrôle des machines Supernode pénalise fortement le recours à la reconfiguration asynchrone sur les modules de base et lui ôte tout intérêt pour les machines hiérarchisées.

Au-delà des problèmes techniques de mise en œuvre évoqués dans les chapitres qui précédents subiste une question essentielle : la reconfiguration dans les machines sans mémoire commune est elle viable, sous quelle forme et dans quelles conditions ?

Le premier argument opposable à la reconfiguration concerne l'extensibilité des machines ; passé la dizaine de millers de processeurs le coût de

construction et la fiabilité d'un réseau de commutation réarrangeable (ou pire : non bloquant) devenant rapidement rédhibitoires.

Notons cependant que, avec ou sans reconfiguration, la construction de machines d'une taille supérieure pose de toutes façons d'épineux problèmes d'encombrement, d'alimentation et de refroidissement (pour ne citer que ceux-ci).

Outre les performances qu'elle exige du réseau de commutation (absence de blocage) et de la structure de contrôle des machines (débit du transputer et de la voie de contrôle), la reconfiguration dynamique asynchrone aboutit à une commande séquentielle du réseau totalement à l'opposé de l'esprit dans lequel les machines parallèles ont été concçues.

De par ses caractéristiques très particulières, la reconfiguration asynchrone ne peut répondre, seule ou couplée à une autre méthode de routage, qu'à des besoins spécifiques, dont l'équilibrage de la charge de calcul des processeurs est un bon exmple.

L'efficacité des autres méthodes de routage adaptatif est intimement liée au coût des conflits d'accès aux liens [1] qu'elles combattent.

Ce coût reste à évaluer par une future campagne d'analyse des communications générées par quelques applications significatives.

La viabilité de la reconfiguration synchrone reste également une question ouverte pour les mêmes raisons.

Dans l'optique d'une poursuite de ce travail, je tiens à mentionner pour terminer deux questions, quelque peu occultées dans cette étude, dont une réflexion approfondie sur les communications dans les multiprocesseurs sans mémoire commune ne saurait faire l'économie :

- La première interrogation porte sur la prévision fiable des communications engendrées par les applications : cette estimation est en effet déterminante pour la qualité du placement des tâches sur les processeurs des machines (à connectique fixe ou programmable) et d'elle dépend le succès de la reconfiguration synchrone.

- L'autre question concerne l'interaction entre les mécanismes d'équilibrage de la charge de calcul entre les processeurs d'une part et les techniques d'acheminement des messages d'autre part; dont la reconfiguration asynchrone fournit une bonne illustration.

[1] dans les machines à connectique fixe et routage déterministe

Annexe A

PARALLELISATION DU CALCUL DES TABLES DE ROUTAGE

A.1 Calcul des plus courts chemins

Le calcul des plus courts chemins se ramène au calcul de la fermeture transitive d'un graphe. L'algorithme parallèle décrit ci-dessous en pseudo-OCCAM, exécuté sur chaque nœud du réseau, calcule une table donnant donnant la longueur du plus court chemin vers chacun des autres nœuds du réseau pour chaque lien de sortie du nœud considéré.

Les chemins sont calculés par ordre croissant de longueur. MAX_INT, le plus grand entier représentable sur la machine, indique un chemin de longueur infinie, c'est-à-dire encore indéfini.

LOCAL désigne le numéro du nœud qui exécute l'algorithme et NUL est un numéro de processeur illégal indiquant la fin d'une itération.

L'algorithme est bâti autour d'un processus producteur (le récepteur chargé de la mise à jour de la table locale) et d'un consommateur (l'émetteur chargé de transmettre les nouveaux chemins aux voisins) reliés par une file premier entré, premier sorti.

L'algorithme est exécuté de manière synchrone par tous les processeurs du réseau. Le nombre minmal d'itérations à exécuter est égal au diamètre du réseau, qui est rarement connu à priori. Dans l'algorithme ci-dessous, le diamètre du réseau a été majoré par le nombre de processeurs.

Lorsque la machine est pourvue d'une voie de sigalisation (cas des machines Supernodes), chauqe nœud peut mettre à jour à la fin de chaque itération un témoin d'activité signalant l'acquisition d'un nouveau chemin. La terminaison du calcul est alors déclenchée via la diffusion d'un signal par le maître de la machine lorsqu'il détecte que tous les processeurs du réseau sont devenus inactifs.

```
-- initialisation : chemins de longueur infinie
table[i=0 FOR MAX_NOEUDS][j=0 FOR MAX_LIENS] = MAX_INT
-- le seul noeud connu est le noeud local
-- distance nulle quel se soit le lien
table[LOCAL][j=0 FOR MAX_LIENS] = 0
-- preparer la premi re etape d'emission
ajouter_file (LOCAL)
ajouter_file (NUL)
PAR
  emettre
  recevoir
```

Tout nouveau chemin est diffusé à tous les voisins :

```
PROC emettre
SEQ
  dist = 1
  -- calcul par ordre croissant des distances
  WHILE dist <= diametre
    SEQ
      dest = retirer_file()
      IF
        dest = NUL
          -- fin de l'etape
          dist = dist + 1
        TRUE
          -- diffuser
          PAR l=0 FOR Nb_liens
            sortie[l] ! dest
```

Le processus de réception est chargé de la mise à jour de la table de routage :

```
PROC recevoir
SEQ
  dist = 1
  -- chaque lien delivre sa sequence
  l_actif[i=0 FOR MAX_LIENS] = VRAI
  Nb_actifs = Nb_liens
  WHILE dist <= diametre
    -- attente sur tous les liens d'entree
    ALT le=0 FOR Nb_liens
      l_actif[l]&entree[le]?dest
        IF
          dest = NUL
            SEQ
              l_actif[le] = FAUX
              Nb_actifs = Nb_actifs - 1
              IF Nb_actifs = 0
                SEQ
```

```
            -- fin de l'etape
            ajouter_file (NUL)
            dist = dist + 1
            Nb_actifs = Nb_liens
            l_actif [] = VRAI
        TRUE
          IF table[dest][le] = MAX_INT
            SEQ
              -- mise a jour table et voisins
              table[dest][le] = dist
              ajouter_file (dest)
```

A.2 Adaptation à la méthode de l'arbre recouvrant

Le principe du calcul parallèle des plus courts chemins peut être adapté à la génération des chemins à partir de l'arbre recouvrant.

Lorsque l'arbre recouvrant a été déterminé, il suffit d'étiquetter chaque lien comme ascendant, descendant ou horizontal.

Il suffit alors de modifier l'algorithme ci-dessus de façon à ne transmettre aux voisins que les chemins qui respectent les règles définies au 5.3 et de générer une table de routage séparée pour chaque lien d'entrée.

```
PROC emettre
SEQ
  dist = 1
  -- calcul par ordre croissant des distances
  WHILE dist <= diametre
    SEQ
      {dest,lien} = retirer_file()
      IF
        dest = NUL
          -- fin de l'etape
          dist = dist + 1
        TRUE
          sortie[lien] ! dest
```

La procédure recevoir gère deux tables : table_int est utilisée pour les messages d'origine locale et table_ext pour les messages reçus sur les liens d'entrée.

```
PROC recevoir
SEQ
  dist = 1
  -- chaque lien delivre sa sequence
```

```
l_actif[] = VRAI
Nb_actifs = Nb_liens
WHILE dist <= diametre
  -- attente sur tous les liens d'entree
  ALT le=0 FOR Nb_liens
    l_actif[l]&entree[le]?dest
      IF
        dest = NUL
          SEQ
            l_actif[le] = FAUX
            Nb_actifs = Nb_actifs - 1
            IF Nb_actifs = 0
              SEQ
                -- fin de l'etape
                ajouter_file ({NUL,NUL})
                dist = dist + 1
                Nb_actifs = Nb_liens
                l_actif [j=0 FOR MAX_LIENS] = VRAI
        TRUE
          SEQ
            IF tab_int[dest][le] = MAX_INT
              SEQ
                -- mise e jour table et voisins
                table[dest][le] = dist
                ajouter_file (dest)
              SEQ ls=0 FOR Nb_liens
                -- mise e jour individuelle
                -- pour chaque lien de sortie
                IF compatible(ls,le) &
                   tab_ext[ls][dest][le] = MAX_INT
                   SEQ
                     tab_ext[ls][dest][le] = dist
                     ajouter_file ({dest,le})
```

Annexe B

LES MACHINES SUPERNODE : LES LIENS

B.1 Les modules de base

La machine (ou module) Supernode de base est organisée autour d'un seul étage de commutation composé de crossbars "Supernode" (encore appelés crossbars RSRE).

On distingue deux modes de fonctionnement des modules de base : le fonctionnement autonome et l'association en machines hiérarchisées.

Le module de base existe en deux versions : le tandem accepte jusqu'à 64 transputers de travail (en mode autonome) et le nœud simple en est une version réduite de moitié (32 processeurs de travail).

Les liens restés libres sur les circuits RSRE permettent d'ajouter des transputers supplémentaires affectés à des fonctions de service (accès aux disques, interface réseau, etc...) à raison de deux par nœud simple et quatre par tandem.

Seront présentés dans l'ordre :

- la structure du nœud simple autonome
- la structure du tandem autonome
- et leur adaptation pour la construction de machines hiérarchisées.

B.1.1 Nœud simple ou machine 16/32

Un nœud simple se présente sous la forme d'un panier de cartes à sept emplacements (figures B.1).

Sur le fond de ce panier sont montés deux circuits RSRE.

On considère chaque circuit comme deux moitiés paire et impaire formant chacune un crossbar indépendant à 36 liens.

Les liens de numéros pairs des circuits RSRE sont reliés directement aux liens physiques 0 et 2 des transputers et forment les crossbars des réseaux Nord vers Sud et Sud vers Nord.

Les liens de numéros impairs des circuits RSRE sont reliés aux liens physiques 1 et 3 des transputers via un jeu de cavaliers sur le fond de panier et forment les crossbars des réseaux Est vers Ouest et Ouest vers Est.

On retrouve bien ainsi la structure de la figure 4.5 vue précédemment.

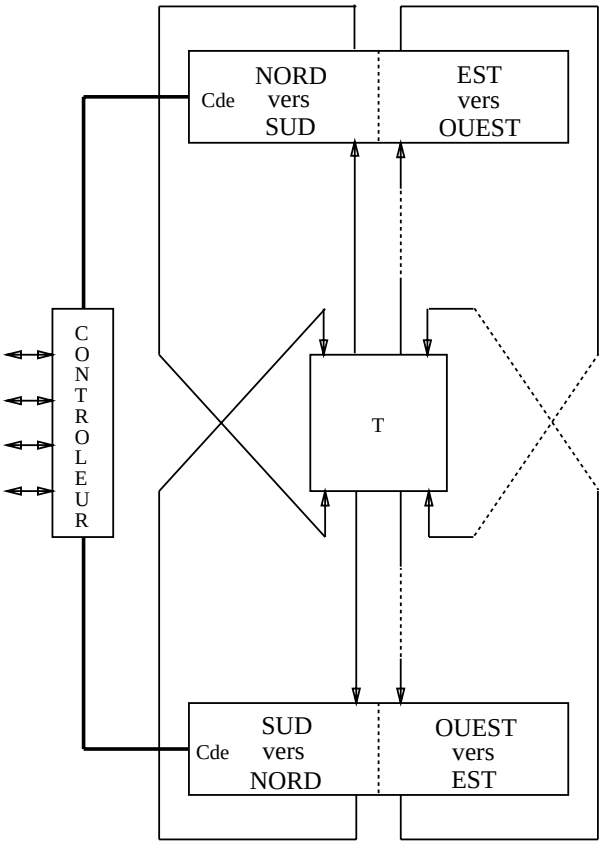


Figure B.1: Le fond de panier d'un nœud simple comprend deux crossbars

Pour faciliter leur désignation, les 144 liens gérés par les deux circuits RSRE sont structurés en neuf groupes de seize liens désignés par les lettres A à I.

Chaque groupe comprend à son tour quatre sous-groupes numérotés de 0 à 3 ; chaque sous-groupe étant lui-même composé des liens Nord, Sud, Est et Ouest connectés à un même transputer. Nous désignerons ainsi un transputer par une lettre et un chiffre identifiant le sous-ensemble de liens auquel il est connecté.

Les 32 transputers d'une configuration autonome sont regroupés dans les groupes A à H. Chacun des emplacements 3 à 6 reçoit une carte à huit processeurs de travail qui seront connectés à deux groupes de liens consécutifs (figures B.2 et C.1).

Chacun des deux emplacements 1 et 2 reçoit également un transputer serveur. Les serveurs sont connectés aux groupes I0 et I1. Les deux sous-groupes I2 et I3 ne sont pas affectés.

Un transputer chargé de gérer l'ensemble de la machine prend place sur une carte contrôleur insérée systématiquement dans l'emplacement 7.

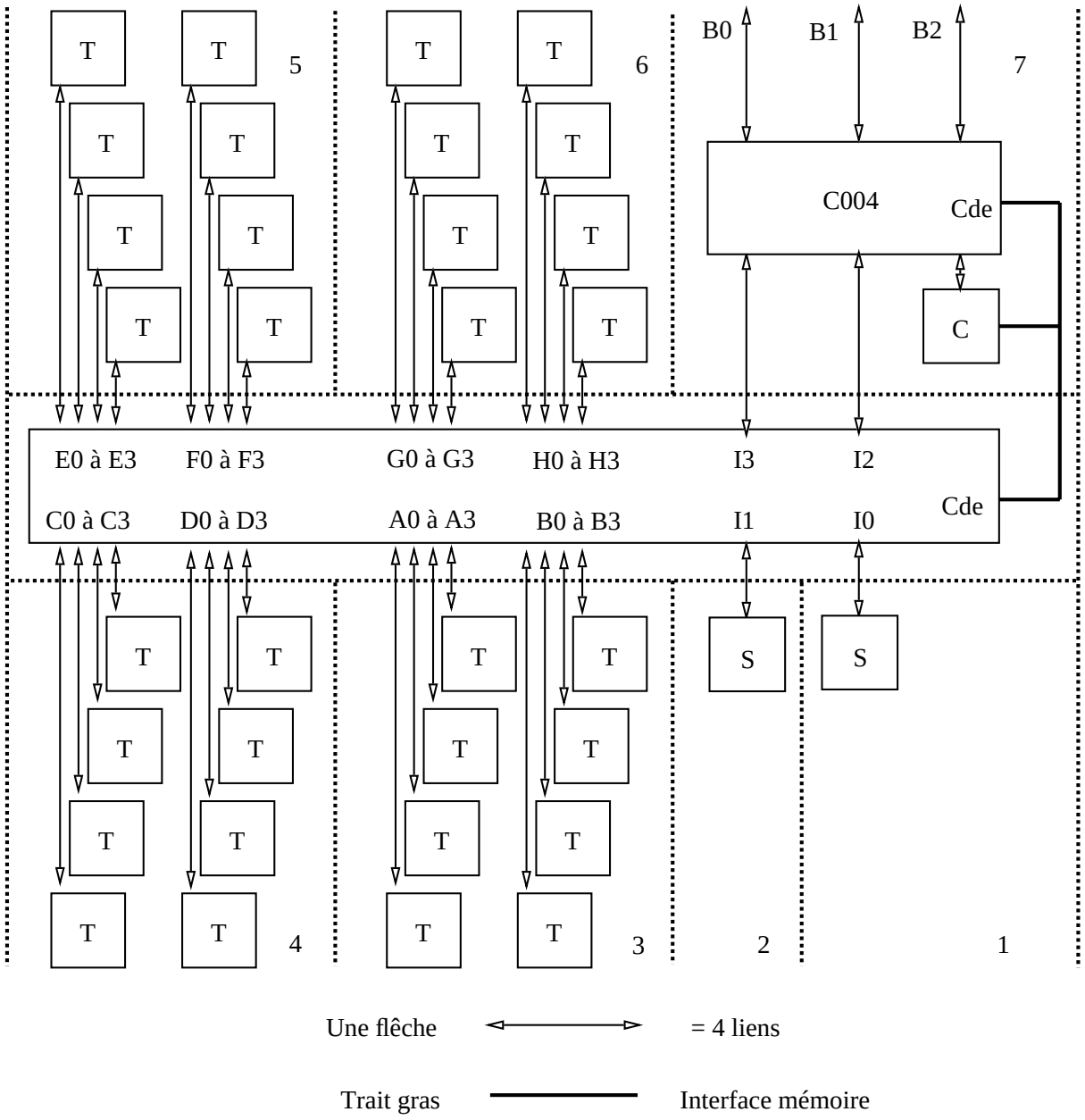


Figure B.2: Organisation d'un nœud simple

Par l'intermédiaire de son interface mémoire, ce transputer contrôleur commande les deux circuits RSRE du fond de panier ainsi que, via un adaptateur de lien, un C004 monté sur la carte contrôleur.

Ce C004 permet d'interconnecter les liens du contrôleur et les deux sous-ensembles de liens I2 et I3 restés libres sur les circuits RSRE. Les sorties restantes du C004 sont dévolues à la communication avec l'environnement extérieur. Les liaisons comprennent trois sous-ensembles de quatre liens amplifiés (ECL ou RS422) reliés aux connecteurs de sortie B0, B1 et B2 de la carte ainsi que huit liens sur les huits connecteurs ITEM (non représentés) au format TTL d'INMOS .

Les serveurs, tout comme le contrôleur sont supposés faire peu de calcul flottant. Pour cette raison, ce sont généralement des circuits de type T414, contrairement aux processeurs de travail qui sont de type T800.

En configuration autonome, le contrôleur peut ainsi être connecté au sous-ensemble I2 ou I3 et apparaître ainsi comme un serveur supplémentaire. On peut aussi relier tout ou partie de ses liens vers l'extérieur. A l'état initial, un séquenceur matériel spécial programme le C004 de façon à ce qu'au moins un des liens du contrôleur soit relié à un connecteur, B0 ou ITEM (choix par cavalier). Cette précaution permet d'accéder au contrôleur qui pourra ensuite reprogrammer le C004 à sa guise.

Les deux paragraphes suivant présentent la façon dont les liens physiques 0 à 3 des transputers sont associés aux directions Nord,Sud,Est et Ouest d'une part et aux liens des circuits RSRE d'autre part.

B.1.2 Tandem ou machine 32/64

Un tandem est constitué par jumelage de deux paniers de noeud simple et comprend donc quatre crossbars RSRE (figure B.3). Les cavaliers des fonds de panier sont positionnés de façon à connecter les liens de numéros impairs des crossbars RSRE d'un fond de panier aux liens physiques 1 et 3 des transputers appartenant cette fois à l'autre panier.

De cette manière, la structure à 4 crossbars de la figure 4.5 est bien conservée. Les circuits RSRE du panier de gauche (que nous appelerons aussi maître) gèrent alors les liens Nord et Sud des transputers des deux paniers du tandem. De la même façon, les liens Est et Ouest de tous les processeurs sont gérés par les circuits RSRE du panier de droite (que nous appelerons aussi esclave).

Un tandem au complet comportera donc deux contrôleurs, quatre serveurs et 64 processeurs de travail.

B.1.3 Utilisation dans les machines hiérarchisées

Utilisé dans une machine hiérarchisée, un nœud simple ou un tandem voit sa capacité réduite de moitié. En effet, la moitié des liens des circuits

RSRE doit être ramenée sur le deuxième étage de commutation des liens.

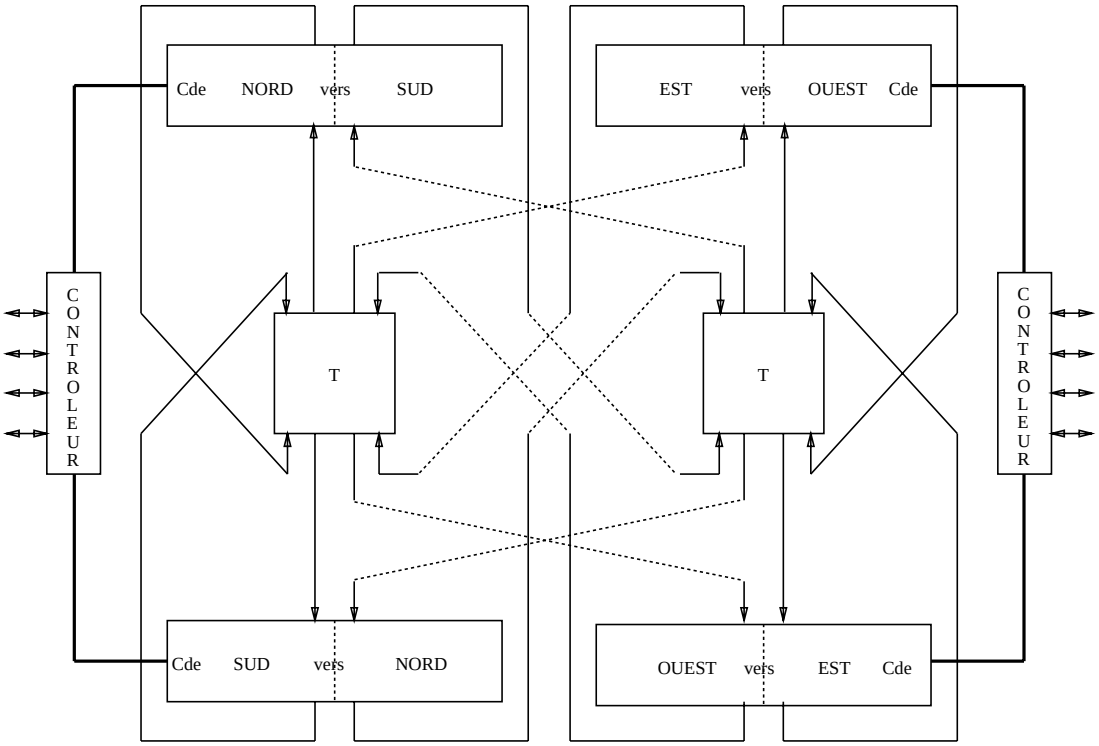


Figure B.3: Deux paniers jumelés forment un tandem

Le nombre de processeurs de travail est alors ramené à 16 pour un nœud simple et à 32 pour un tandem. C'est pourquoi ces deux machines sont également connues sous le nom de machines respectivement 16/32 et 32/64.

Compte-tenu des distances à parcourir et du nombre de circuits à traverser, les signaux électriques des liens doivent être recalibrés et amplifiés pour traverser le deuxième étage de commutation. Pour cela, les cartes de travail des emplacements 5 et 6 d'un panier sont remplacées par des cartes d'amplification (figure B.4).

Un T212 gère deux C004 dont la seule fonction est ici de recalibrer les signaux. En fonctionnement normal, le T212 démarre à partir d'une mémoire morte et initialise les C004 de façon à les rendre transparents (connexions en pointillés).

En série avec les C004 se trouve une batterie d'amplificateurs (ECL ou RS422), après quoi les liens sont regroupés quatre à quatre sur huit connecteurs de sortie.

La figure B.5 représente un tandem totalement équipé dans une machine hiérarchisée (chaque flêche représente quatre liens).

On trouve à gauche les 32 processeurs de travail, les quatre serveurs et les deux contrôleurs.

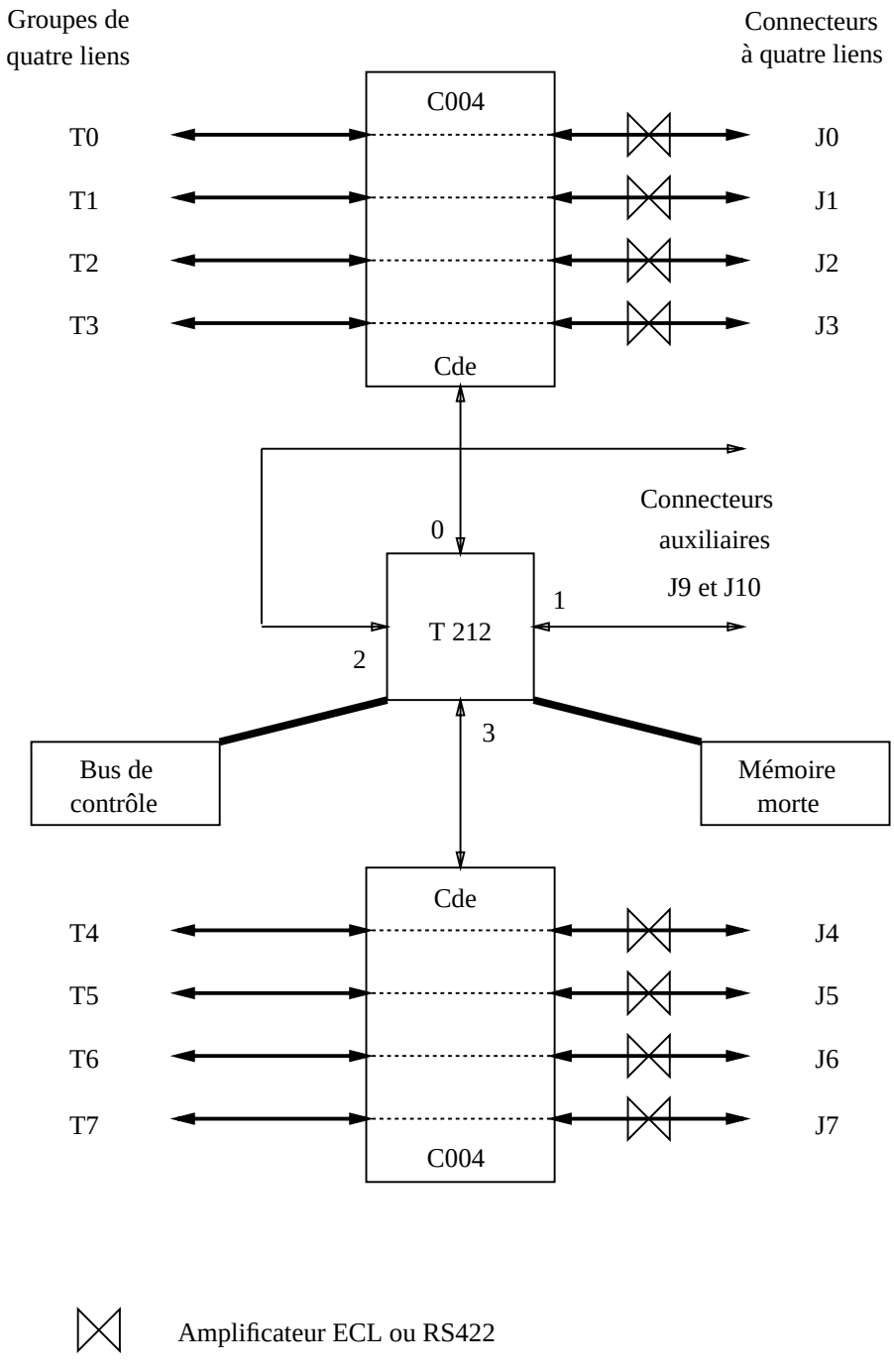


Figure B.4: La carte d'amplification

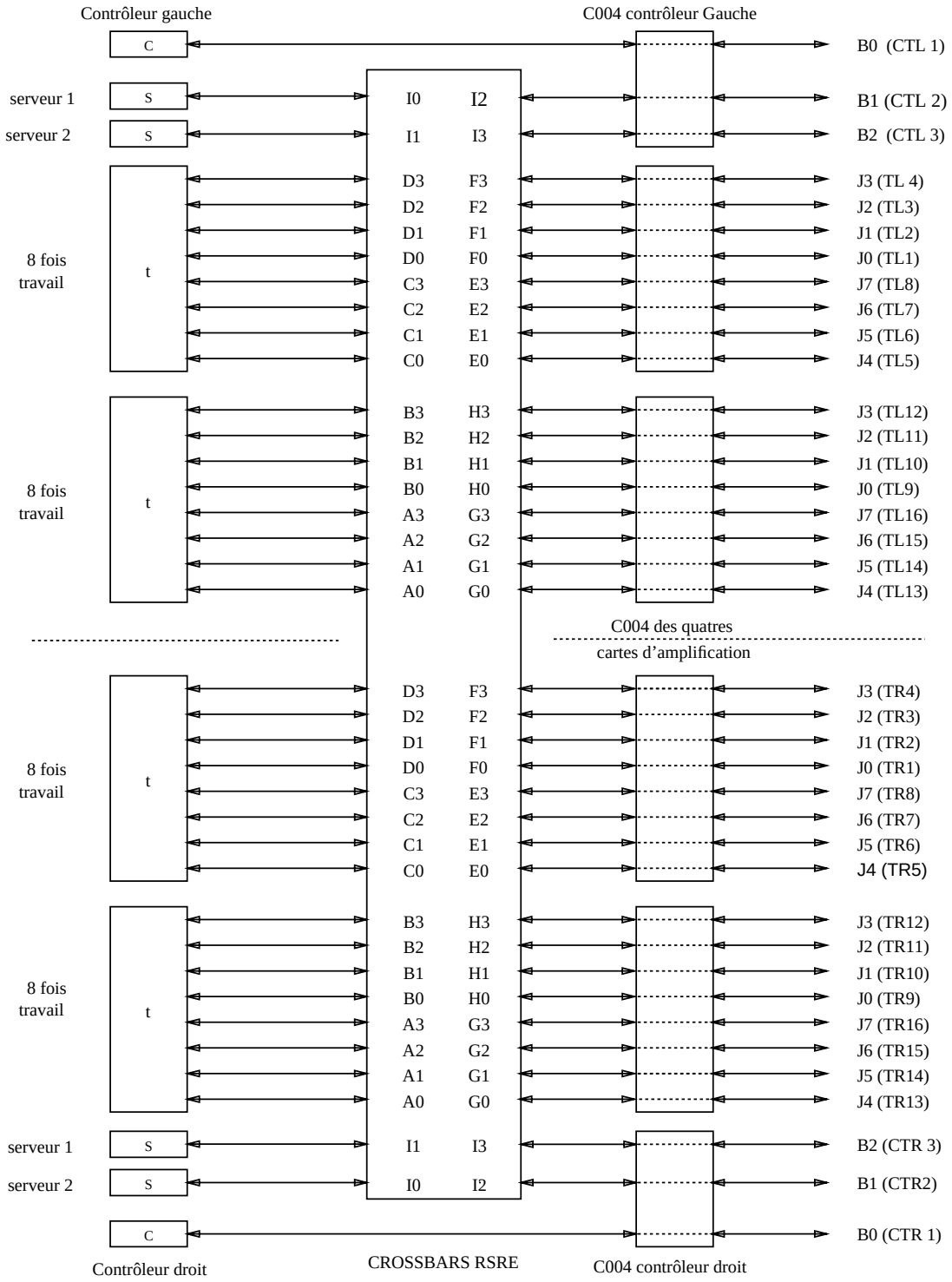


Figure B.5: Tandem inclus dans une machine hiérarchisée

En face de chaque carte de travail du côté gauche se trouve (du côté droit) une carte d'amplification qui permet d'acheminer les liens correspondant vers le deuxième niveau de commutation des liens.

Faute de place sur les cartes d'amplification, les liens correspondant aux serveurs sont recalibrés et amplifiés sur les cartes contrôleurs et sortent sur les connecteurs B1 et B2 de celles-ci. De la même manière, les liens des contrôleurs sortent sur les connecteurs B0.

La colonne de droite présente la notation géographique utilisée par le constructeur pour repérer les différents groupes de liens d'un tandem. CTL et CTR identifient les cartes contrôleur des moitiés gauche et droite du tandem respectivement ; de la même manière, TL et TR correspondent aux deux paires de cartes d'amplification.

B.1.4 Remarques

Le nœud simple et le tandem forment le bas de la gamme de machines Supernode, pour des puissances inférieures à la centaine de Mflops/s. Toutes deux sont articulées autour d'un seul niveau de commutation de liens.

Trois caractéristiques méritent toutefois d'être soulignées :

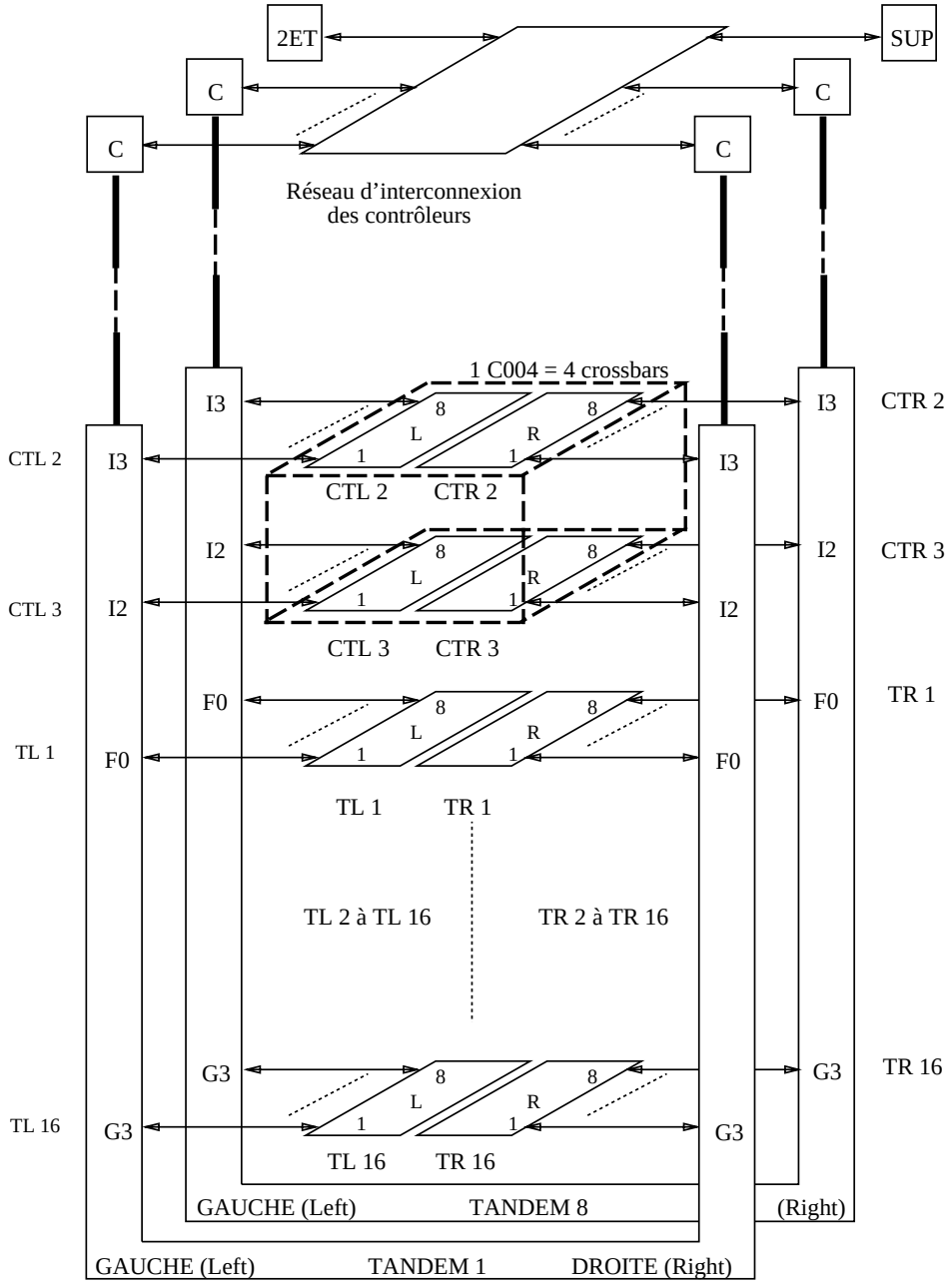
- la restriction sur l'orientation des liens mise à part, le réseau est non seulement réarrangeable, mais non bloquant puisque de type crossbar.
- le tandem présente une anomalie de contrôle. En effet, la structure du réseau d'interconnexion, est la même que celle d'un nœud simple, c'est-à-dire à un seul niveau. La commande des crossbars RSRE et des C004 des cartes contrôleur aurait donc dû être confiée à unique contrôleur. Il aurait suffit de ramener quelques signaux d'un fond de panier sur les connecteurs de la carte contrôleur de l'autre fond de panier.

  La présence de deux contrôleurs constitue donc une erreur de conception qui va rendre la commande du éseau plus complexe.

B.2 Les machines hiérarchisées

B.2.1 Principe

Nous avons vu précédemment comment un module de deuxième niveau composé de quatre jeux (un par direction) de q crossbars à p liens chacun permet d'interconnecter p modules de q transputers chacun en une machine unique à p fois q processeurs. Pour chacune des quatre directions, le deuxième étage comprendra alors q crossbars à p liens.



Reseau d'interconnexion des processeurs de travail et serveurs

Chaque flêche représente un groupe de 4 liens (N,S,E,O)

36 * 4 crossbars à 8 liens gèrent les liens des processeurs de travail

Figure B.6: Machine 256 organisée en 8 tandems

Ignorons dans un premier temps les contrôleurs.

Examinons à titre d'exemple la machine Méganode 256 composée de huit tandems (soit huit modules à 36 processeurs). Cette machine comprend 256 processeurs de travail et 32 serveurs (figure B.6). Le deuxième étage est alors composé (pour chaque direction) de 36 crossbars à huit liens.

On peut aussi construire une machine de même taille (figure B.7) à partir de 16 nœuds simples. Le deuxième étage sera dans ce cas structuré sur la base de 18 crossbars à 16 liens.

Dans la pratique, toutefois, des tandems sont systématiquement utilisés pour conserver un maximum de connexions locales au premier niveau de commutation. Etant donné la taille des C004 utilisés au deuxième niveau, le recours à des tandems est également impératif pour les machines de 512 à 1024 processeurs..

B.2.2 Le module de deuxième niveau

La carte de commutation

Une carte de commutation (figure B.8) est composée de quatre C004 couplés gérant chacun une direction dont les liens sont ramenés sur trente-deux connecteurs. Chaque connecteur regroupe un sous-ensemble de quatre liens Nord, Sud, Est , Ouest.

Deux T212s commandent chacun une paire (Nord-Sud ou Est-Ouest) de C004 via leurs liens zéro et trois. Les liens 1 et 2 sont utilisés pour chaîner les T212 des différentes cartes pilotant une même direction .

Le panier du deuxième étage

Un panier spécial à dix emplacements permet de connecter en guirlande les T212 des cartes de commutation (figure B.9). Ces liaisons sont réalisées de façon rigide par le circuit imprimé du fond de panier. Un cavalier mobile permet de reboucler les deux morceaux de chaque guirlande quelque soit le nombre de cartes de commutation présentes.

L'ajout d'un panier supplémentaire de prolongation, permet, le cas échéant, d'agrandir de façon modulaire le deuxième étage pour loger le nombre de cartes de commutation voulu.

Une carte contrôleur C2ET (semblable à celles de nœuds simples et tandems) pilote l'ensemble du deuxième étage, les T212 servant uniquement de relai de connexion. Le panier de deuxième étage étant dépourvu de circuits RSRE, le sous-groupe de siens I3 du C004 est réaffecté aux extrémités des guirlandes. Le sous-groupe I2 n'est pas utilisé.

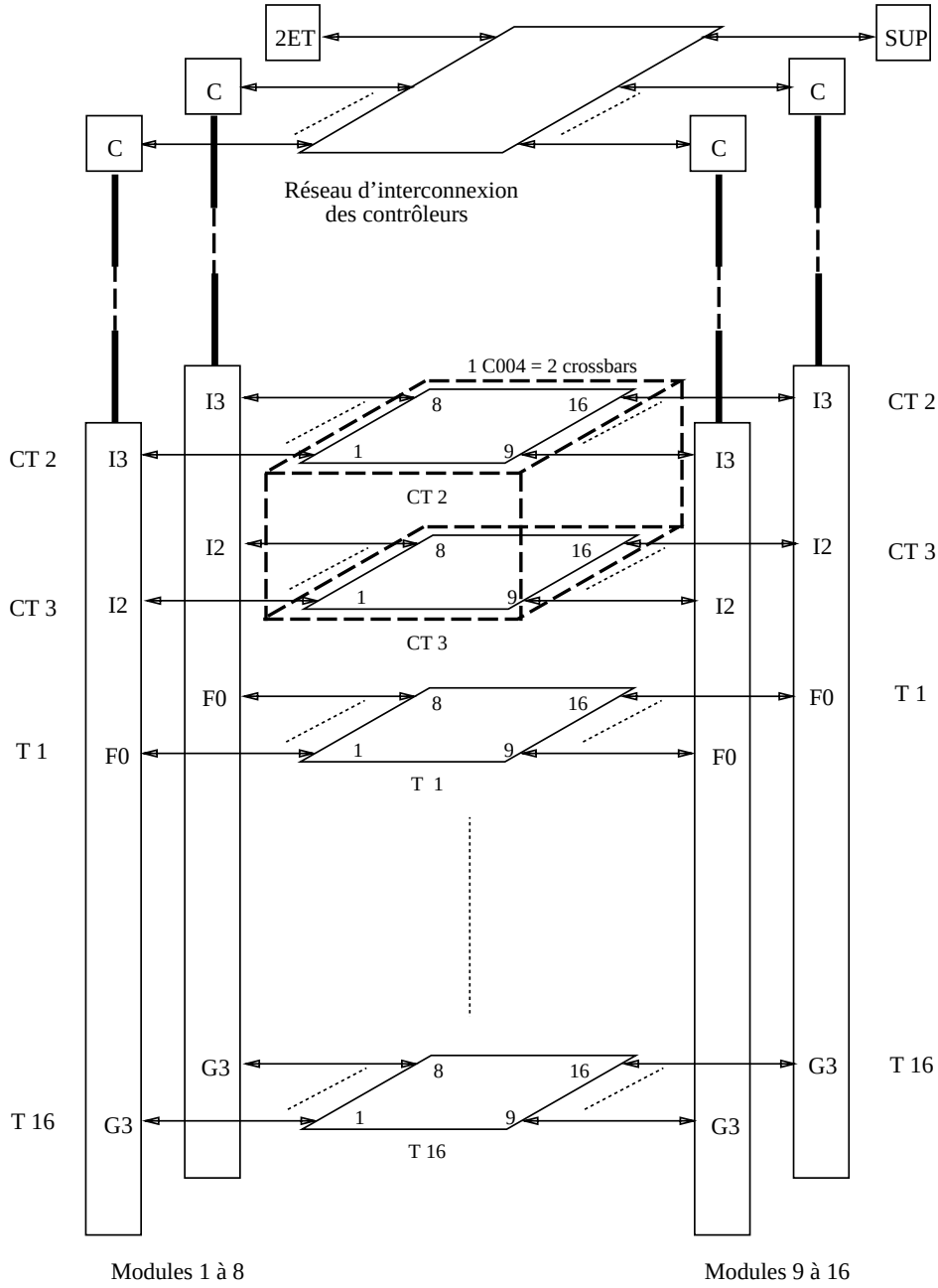
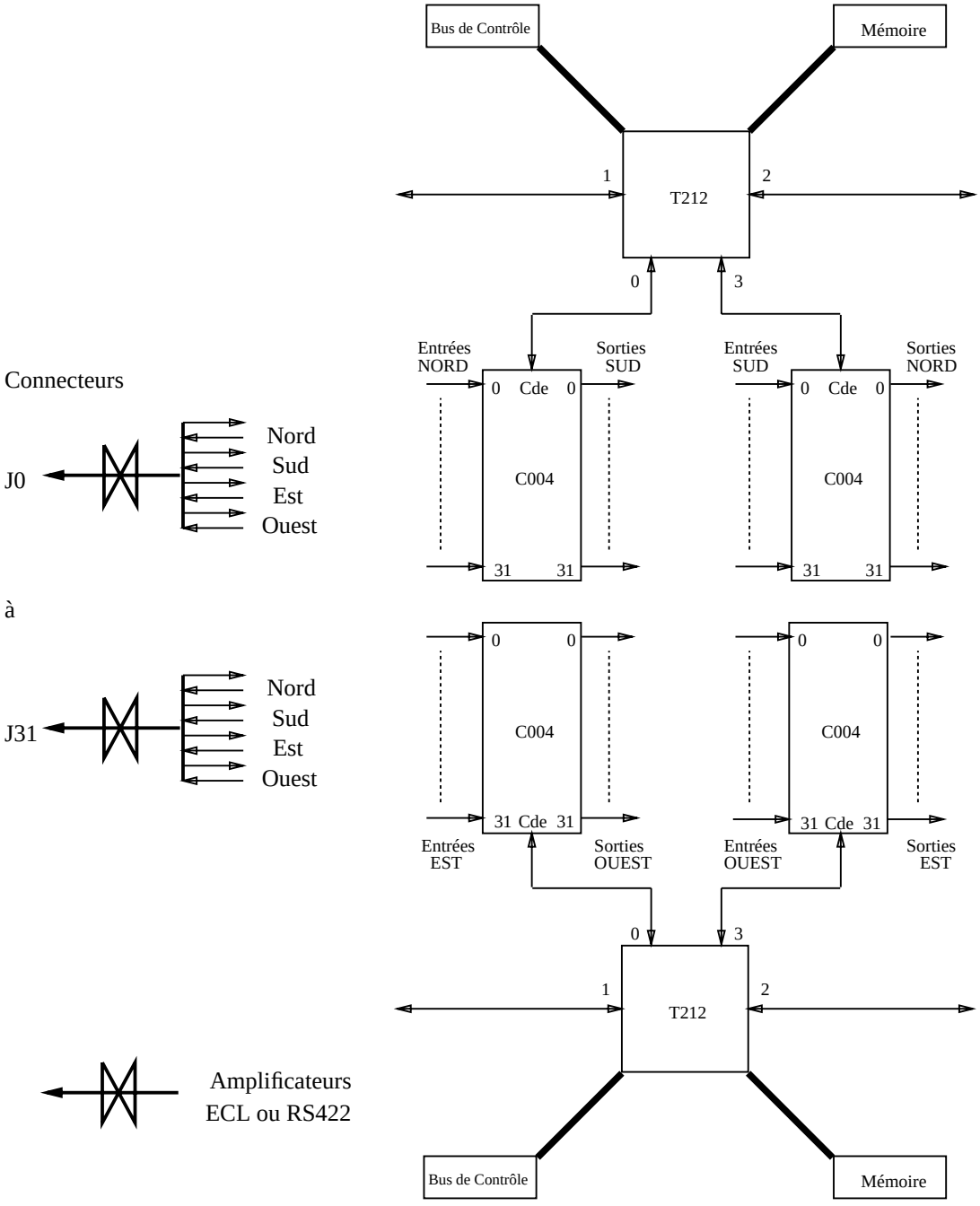


Figure B.7: Machine 256 organisée en 16 nœuds simples



Sur les connecteurs de la carte de commutation arrivent 32 groupes de quatre liens N,S,E,O

Figure B.8: Synoptique d'une carte de commutation

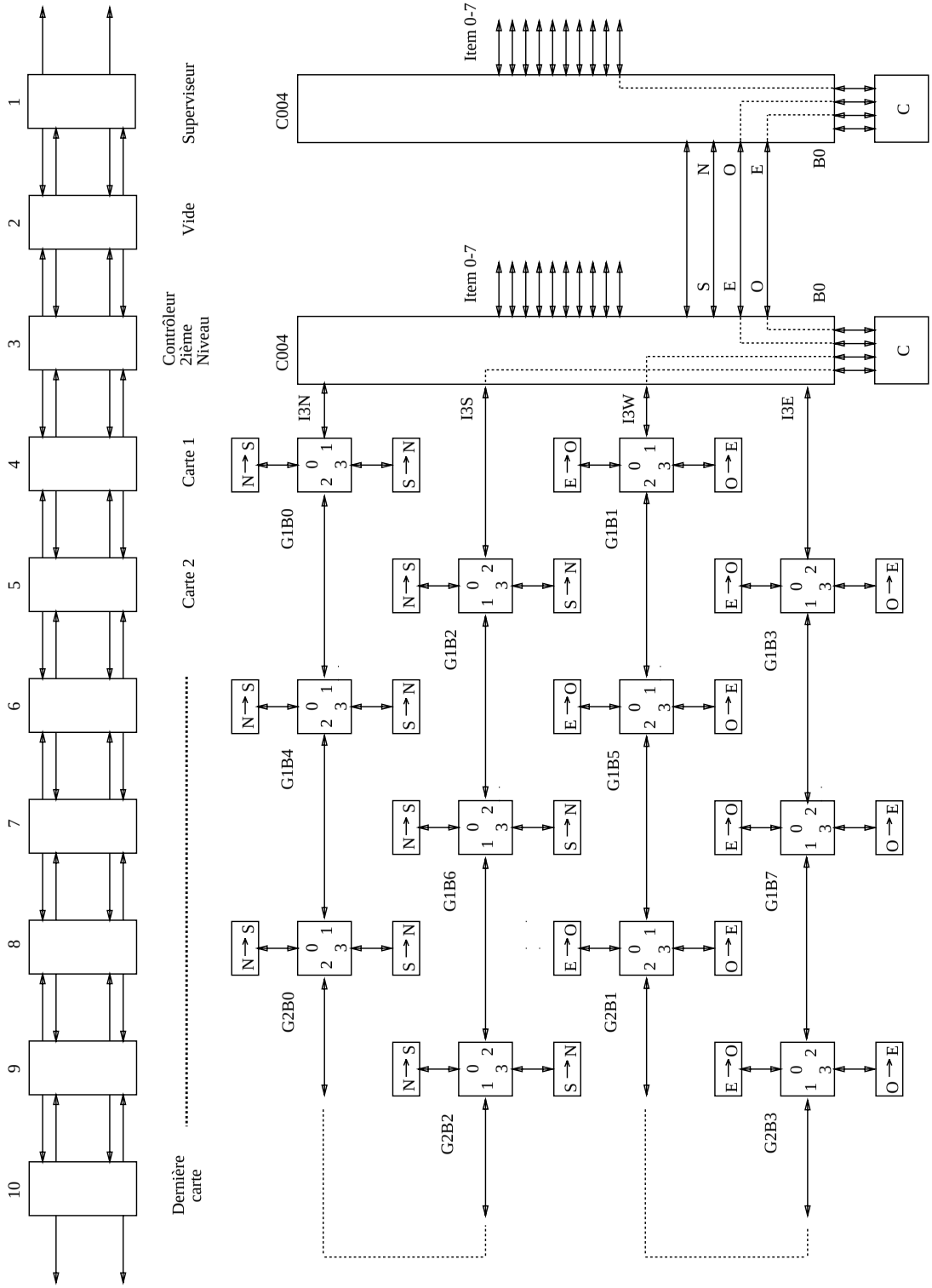


Figure B.9: Le panier du deuxième étage

On trouve aussi, bien que ne faisant pas partie du deuxième étage proprement dit, une carte contrôleur erperviseur chargée de commander la voie de contrôle.

A l'initialisation, les C004 de ces deux cartes sont programmés (connexions en pointillé) de façon à ce que le contrôleur de deuxième étage soit accessible depuis l'extérieur via le port ITEM 0 du erperviseur et un câble additionnel reliant directement les ports B0 des deux cartes.

En haut et à gauche de chaque T212 se trouve son adresse (E,P) sur le bus de contrôle du deuxième étage (voir chapitre suivant).

Regroupement des crossbars

Lorsque cela est possible (nombre de modules inférieur à 16), on regroupe plusieurs crossbars du deuxième niveau sur un même C004.

Par exemple, les 36 (quadruples) crossbars à huit siens d'une machine 256 (composée de huit tandems) seront regroupés quatre à quatre dans 9 cartes de commutation.

La complexité de l'algorithme de pilotage du deuxième étage dépend du nombre de crossbars que ce dernier comporte.

Pour cette raison, les crossbars gérant les siens de même nom des moitiés gauches et droites des tandems sont toujours regroupés dans une même carte de commutation et peuvent alors être fusionnés en un seul crossbar de taille double. Tout se passe alors exactement comme si la machine était construite à partir de nœud simples et le nombre de crossbars à gérer est divisé par deux.

Le deuxième étage d'une machine 512 sera donc réorganisé en 18 crossbars à 32 siens interconnectant 32 moitiés de tandems plutôt qu'en 36 crossbars à 16 siens interconnectant 16 tandems.

Pour les machines de taille inférieure, on peut même imaginer de considérer des quarts de tandems de façon à réduire encore le nombre de crossbars à considérer.

Dans les descriptions qui suivent, le deuxième étage est toujours présenté sur la base d'une structuration en nœuds simples.

B.2.3 Connexion des contrôleurs

Il existe trois manières de connecter les contrôleurs :

1. Suppression des serveurs.

   Les contrôleurs peuvent simplement remplacer les serveurs (qui sont éliminés), les connexions étant réalisées via les groupes de siens I2 et I3. Si chaque contrôleur n'est relié qu'à des processeurs de travail

de son tandem ou à d'autres contrôleurs, il n'est plus nécessaire de supprimer qu'une des deux paires de serveurs par tandem.

2. Une autre méthode consiste à associer les contrôleurs en un ou plusieurs modules de la taille d'un nœud simple qui seront alors traités comme des modules ordinaires supplémentaires (figure B.10). On a ainsi toute liberté de connexion entre les contrôleurs et les processeurs de travail (et serveurs).

   Le revers de la médaille est l'accroissement de la taille des crossbars, ce qui conduit généralement à une augmentation sensible du nombre de cartes de commutation. Ainsi le deuxième étage d'une machine 256 reconfigurant totalement les contrôleurs suivant ce principe comprend 18 crossbars à 17 liens (16 demi-tandems et un nœud de contrôleurs) et occupera donc 18 cartes de commutation au lieu de 9 lorsque les contrôleurs sont gérés à part (ci-dessous).

3. Un compromis intéressant consiste à interconnecter les contrôleurs uniquement entre eux dans un réseau séparé. C'est la structure adoptée dans la machine Méganode livrée à l'IMAG. Le deuxième étage comprend donc 9 cartes de commutation pour les processeurs de travail et les serveurs et une dixième carte qui gère à part les contrôleurs.

B.2.4 Conclusion

On peut faire quelques remarques en guise de conclusion :

- Les machines Supernode de grande taille sont réalisées de façon modulaire et le deuxième niveau de commutation de liens également. L'objectif de modularité est donc satisfait.

- Il est toujours possible de rétablir la condition d'absence de blocage q >= 2m -1 en réduisant de nouveau de moitié le nombre de transputers de travail dans les modules de base (soit le quart de la configuration autonome), mais ceci réduit également la taille maximale des machines de moitié.

- Les T212 des cartes de commutation du deuxième niveau résolvent essentiellement un problème de connectique entre le contrôleur de deuxième niveau et les C004. Du fait des possibles conflits d'accès à un même C004 entre les différentes liaisons à réaliser, l'algorithme de commande du deuxième niveau sera exécuté de façon centralisée sur ce contrôleur.

- Le pilotage du deuxième niveau souffre d'une double handicap : la complexité intrinsèque de l'algorithme de pilotage des réseaux de Clos et la nécéssité de router les messages de commande le long des deux guirlandes de T212.

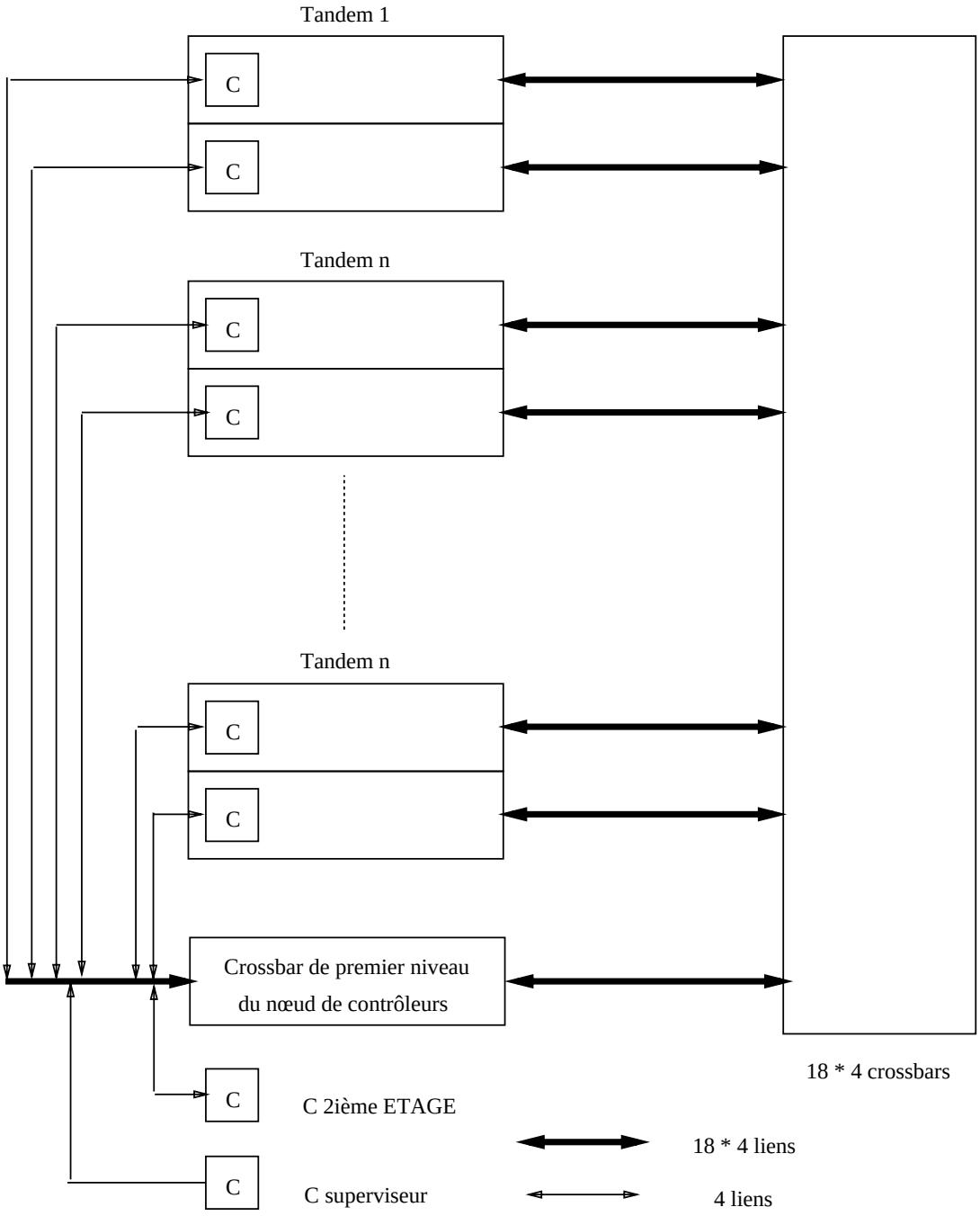


Figure B.10: Regroupement des contrôleurs en un nœud supplémentaire

Annexe C

LES MACHINES SUPERNODE : VOIE DE CONTROLE

C.1 Cahier des charges et principe

Le réseau de commutation des machines Supernode est accompagné d'une voie de signalisation qui doit permettre la supervision de l'ensemble de la machine et supporter le dialogue lié à la reconfiguration dynamique entre le(s) contrôleur(s) et transputers de travail.

C.1.1 Gestion des transputers de travail

Quelque soit le mode de fonctionnement choisi pour les machines Supernode, la voie de signalisation doit au minimum permettre au transputer contrôleur d'accèder aux fonctions de service des transputers dont il assure la supervision.

Ces fonctions de service concernent essentiellement :

- la gestion des paramètres physiques : dans le cas du transputer, il s'agit de choisir la vitesse des liens ;
- la réinitialisation : déclenchement et choix du mode de réinitialisation et
- la détection d'erreur : erreur du transputer lui-même et/ou de parité de la mémoire externe.

C.1.2 Reconfiguration synchrone

En cas de reconfiguration synchrone, la voie de signalisation sera utilisée lors des transitions d'étape (de calcul) à étape, qui met en jeu trois phases :

1. Au cours de la phase de détection de fin d'étape, un signal doit être envoyé au contrôleur lorsque le dernier des processeurs atteint le point de reconfiguration.
2. L'étape suivante est la reconfiguration proprement dite, c'est-à-dire la reprogrammation du réseau de commutation par le contrôleur selon la topologie correspondant à la prochaine étape.
3. Le contrôleur doit ensuite déclencher la poursuite du calcul en diffusant un signal de reprise à l'ensemble des transputers de travail. Ce signal peut éventuellement être accompagné du numéro de cette nouvelle étape.

C.1.3 Reconfiguration asynchrone

Le scénario de communication-synchronisation associé à la reconfiguration asynchrone est conceptuellement plus simple. Ce scénario ne comprend que deux phases :

- une requête d'un client vers le serveur. Cette requête émise vers le serveur comprend un signal et un message spécifiant la nature de la requête (établissement ou clôture de connexion)
- une réponse de même type (signal et message) du serveur à son client.

Les mécanismes de communication/synchronisation associés à la reconfiguration asynchrone sont donc :

- l'émission d'un signal vers le serveur par un processeur quelconque de la grappe.
- le transfert d'un message de ce même client vers le serveur.
- l'émission en retour d'un signal du serveur vers son client.
- associé au signal ci-dessus, le transfert d'un message contenant la réponse du serveur vers le client.

C.1.4 Cas des machines hiérarchisées

Pour simplifier, nous supposerons que les machines hiérarchisées sont composées uniquement de nœuds simples.

Dans une machine hiérarchisée, la voie de contrôle doit être organisée de façon à satisfaire les contraintes spécifiques de la gestion du deuxième niveau de commutation tout en restant compatible avec la structure adoptée dans les nœuds simples autonomes.

Le principe de gestion des connexions au travers du réseau d'interconnexion d'un nœud simple autonome peut être transposé au deuxième niveau de commutation des liens. Le contrôleur de deuxième étage joue alors le rôle de serveur de connexions dont les contrôleurs deviennent les clients.

De même, les signaux de service des contrôleurs de nœud simple doivent être gérés par un maître. Le transputer qui supervise le deuxième niveau de commutation constitue le condidat idéal pour cette fonction.

C.2 Principe

La structure de la voie de contrôle a été calquée sur celle du réseau d'interconnexion.

Dans un nœud simple, la voie de contrôle se présente sous la forme d'un bus parallèle de même nom. Ce bus est de type maître-esclave, le maître étant bien sûr le contrôleur qui pilote le réseau d'interconnexion et les esclaves les processeurs de travail (serveurs inclus).

Dans une machine hiérarchisée, la voie de contrôle l'est également, suivant une structure à deux niveaux.

Le premier niveau correspond au bus de contrôle local de chaque module.

Le deuxième niveau comprend un bus de contrôle de même type par lequel le contrôleur de deuxième niveau (maître) pilote les contrôleurs de modules (esclaves) qui assurent le relai vers les processeurs de travail de leurs modules respectifs.

Nous allons maintenant présenter les services offerts par le bus de contrôle ainsi que son fonctionnement, dans le cas d'un nœud simple autonome.

L'organisation détaillée de la voie de contrôle dans les tandems et les machines hiérarchisées fera l'objet du dernier chapitre.

C.3 Fonctionnement du bus

Le bus de contrôle est un bus parallèle huit bits de type maître-esclave.

Chaque processeur, esclave ou maître, est connecté au bus par une interface qu'il voit comme une zone mémoire débanalisée.

L'interface de bus peut générer un signal sur l'entrée "évènement" des transputers.

Le bus de contrôle offre deux primitives de communication :

- le transfert d'un octet de l'interface d'un esclave donné vers celle du maître, réalisé via un cyle de lecture sur le bus et
- la diffusion d'un octet de l'interface maître vers les interfaces d'une grappe d'esclaves obtenue par un cycle d'écriture sur le bus.

et deux primitives de synchronisation :

- La requête OU permet à un esclave quelconque d'un ensemble donné d'émettre un signal de requête vers le maître. Le maître a la possibilité, pour acquitter cette requête, de générer en retour un signal vers cet esclave.
- L'acquittement ET permet au maître de diffuser un signal de requête vers un ensemble donné d'esclaves. Le maître recevra alors en retour un signal lorsque tous les esclaves de l'ensemble auront acquitté cette requête.

C.3.1 Protocole de synchronisation

Nous allons maintenant aborder le protocole de communication-synchronisation entre le maître et le(s) esclave(s), dans les deux modes de reconfiguration.

Reconfiguration synchrone

Dans la phase de détection de fin d'étape, le maître émet une requête de transition d'étape que les esclaves acquittent lorsqu'ils atteignent le point de reconfiguration. Soit E la grappe à reconfigurer.

Le maître :

- Sélectionner E pour la synchronisation ET et la diffusion d'octet.
- Sur E, activer la synchronisation ET, qui va émettre un signal de requête vers les esclaves.
- Attendre la réception de l'acquittement pour commencer à reconfigurer le réseau d'interconnexion.

Chaque esclave de E :

- Code de l'étape précédente
- Attendre un signal de requête
- Acquitter la requête

La reconfiguration terminée, le contrôleur va diffuser un signal de reprise que nous supposerons accompagné d'un message d'un octet.

Le maître :

- Sur E, diffuser l'octet du message
- Sur E, activer la synchronisation ET
- Attendre l'acquittement
- Retirer la sélection.

Chaque esclave de E :

- Attendre un signal de requête
- Consommer l'octet du message
- Acquitter la requête.

Reconfiguration asynchrone

Rappel : lors de la première phase, l'esclave envoie une requête (signal et message) vers le maître, via la synchronisation OU. Nous supposerons que le message est composé d'un seul octet.

Le maître :

- Sélectionner E pour la requête de type OU
- Attendre une requête d'un esclave
- Déterminer par scrutation l'esclave R à l'origine de la requête
- Restreindre la sélection à R pour le transfert d'un octet dans le sens esclave vers maître et pour la génération d'un acquittement
- Transférer l'octet par un cycle de lecture (de l'interface de l'esclave par le maître.
- Emettre un acquittement vers R.
- Retirer la sélection de R.

Chaque esclave de E :

- Déposer le message d'un octet dans l'interface esclave
- Emettre une requête de type OU vers le maître
- Attendre un acquittement indiquant la prise en compte de la requête.

Plusieurs esclaves peuvent émettre simultanément chacun une requête et le maître ne recevoir qu'un seul signal. Il pourra dans ce cas traiter séquentiellement les requêtes tout en combinant la phase de scrutation et la phase de réponse à chaque requête.

Lors de la deuxième phase, le maître renvoie à un l'esclave R le résultat de sa requête. Cette communication constitue un cas particulier de la diffusion, l'ensemble des esclaves destinaires se réduisant au seul processeur R. Le protocole utilisé a déjà été présenté dans le cadre de la reconfiguration synchrone.

Remarque : il est bien sûr possible de transférer des messages de plusieurs octets. Il suffit pour cela de répéter le cycle de communication et synchronisation (sauf la sélection qui est déjà effectuée).

C.3.2 Adressage des esclaves

Le mécanisme d'adressage des esclaves s'inspire fortement de la structure physique du nœud simple.

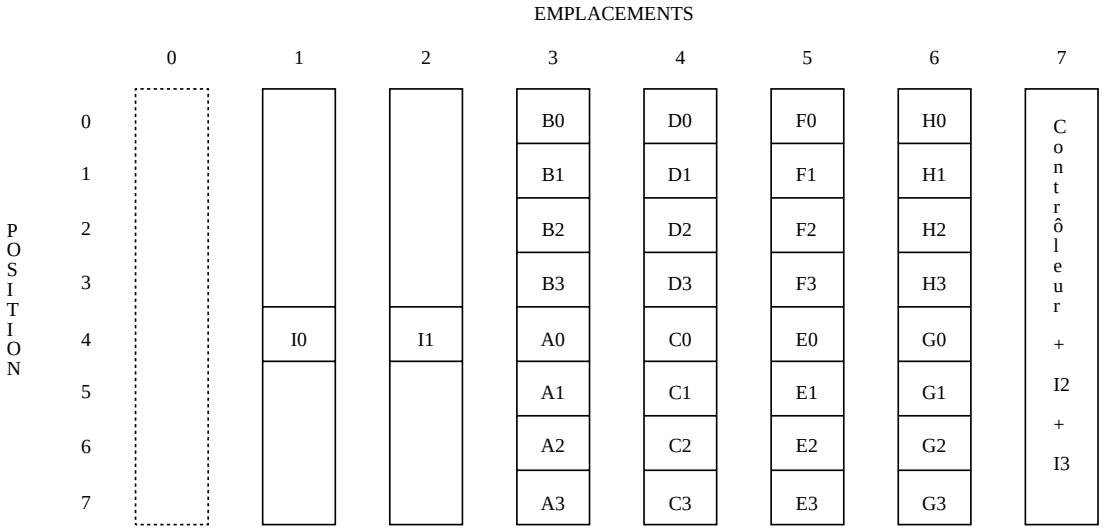


Figure C.1: Adressage géographique des esclaves

Le bus de contrôle considère un panier virtuel à sept emplacements (numérotés de 0 à 6) de huit esclaves chacun. L'adresse d'un esclave est donc formée d'un couple (emplacement, position dans l'emplacement).

Dans un nœud simple, les emplacements 3 à 6 du panier virtuel correspondent aux emplacements de mêmes numéros du panier de cartes physique. Il en va de même pour les emplacements 1 et 2, mais seule la position quatre peut être occupée (par un serveur).

L'emplacement 0 est purement virtuel, mais définit une adresse d'esclave légale. L'emplacement 7 correspond à l'emplacement du contrôleur et l'adresse associée est illégale et sera traitée comme définissant l'ensemble de la machine.

Tout cycle de bus comprend deux phases durant chacune desquelles un octet est placé sur le bus.

Un octet d'adresse est émis par l'interface maître lors de la première phase. Cet octet comprend deux champs :

- un numéro d'emplacement sur trois bits qui définit les interfaces esclaves concernées par le cycle
- un code de fonction sur cinq bits qui définit, à l'intérieur de chacune des interfaces esclaves, le dispositif accédé.

Selon la fonction exécutée, lors de la deuxième phase, dite de donnée, un octet peut ou non être transféré sur le bus. Le sens de transfert et la signification de cet octet sont données par le code de fonction.

Une interface esclave comprend trois types de dispositifs :

- des tampons d'un octet pour les transferts de donnée entre maître et esclave(s)
- des bascules booléennes de présélection pour une primitive de communication ou de synchronisation donnée.
- des bascules booléennes d'état, associées à des signaux (Reset, Analyse, etc...) ou à une synchronisation (présence ou absence d'une requête).

Les fonctions de lecture avec adressage permettent de consulter les bascules booléennes d'état des huits esclaves d'un emplacement donné. L'octet transféré des esclaves vers le maître lors de la phase de donnée donne alors la valeur de la bascule consultée, chaque bit étant associé à l'esclave de position correspondante dans l'emplacement.

Exemple : un cycle de lecture (adresse = 011.01100, donnée = 00110100) indique que, dans l'emplacement trois (code 011), la bascule maître vers esclave BMVE (code fonction 01100) est active chez les esclaves B2, A0 et A1 (positions 2, 4 et 5) et inactive chez les esclaves B0, B1, B3, A2 et A3 (positions 0, 1, 3, 6 et 7).

Les fonctions d'écriture avec adressage pilotent les bascules booléennes de présélection. Lors de la phase de donnée, l'octet est alors transféré du maître vers les esclaves ; la fonction étant activée sur les seuls esclaves dont le bit correspondant est à un. Cet octet n'est donc pas une donnée mais une adresse d'esclave(s) dans l'emplacement.

Exemple : un cyle d'écriture (adresse = 011.10001, donnée = 00110000) active la présélection pour la diffusion maître vers esclaves (code fonction 10001) chez les esclaves A0 et A1 de l'emplacement 3. La bascule de présélection n'est pas modifiée chez les autres esclaves.

Le code d'emplacement 7 (111) active les fonctions conditionnelles. Une fonction conditionnelle porte par défaut sur l'ensemble de la machine. Son exécution sur chaque esclave est conditionnée par la valeur de la bascule de présélection locale de l'interface esclave considéré. La phase de donnée peut alors être utilisée pour transférer un octet de donnée entre le maître et les esclaves.

Nous allons maintenant passer en revue les différents blocs fonctionnels d'une interface de type esclave. Les fonctions repérées par une lettre suivie du code (décimal) de la fonction. La lettre A correspond aux fonctions avec adressage et la lettre C aux fonctions conditionnelles.

C.3.3 Le bloc de service

Le signal BootFromRom (bascule BFR) est géré directement par les fonctions avec adressage A7 et A23.

Symbole	FONCTION	Valider Activer	Invalider Inhiber	Relire
PFS	PRESELECTION SIGNAUX DE SERVICE	A25	A9	-
RES	SIGNAL RESET	C1	C2	-
ANA	SIGNAL ANALYSE	C3	C4	-
L0	VITESSE LIEN 0	C16	C17	-
L123	VITESSE LIENS 1, 2 ET 3	C18	C19	-
LSPE	VITESSE SPECIALE DES LIENS	C20	C21	-
BFR	SIGNAL BOOT FROM ROM	A23	A7	-

Table C.1: Table des fonctions de service

L'accès aux autres fonctions de service est conditionné par une bascule commune de présélection PFS manipulée par les fonctions A9 et A25 (figure C.2).

Les signaux Reset et Analyse sont manipulés par les fonctions conditionnelles C1 à C4 et les signaux de sélection de vitesse des liens par les fonctions conditionnelles C16 à C21.

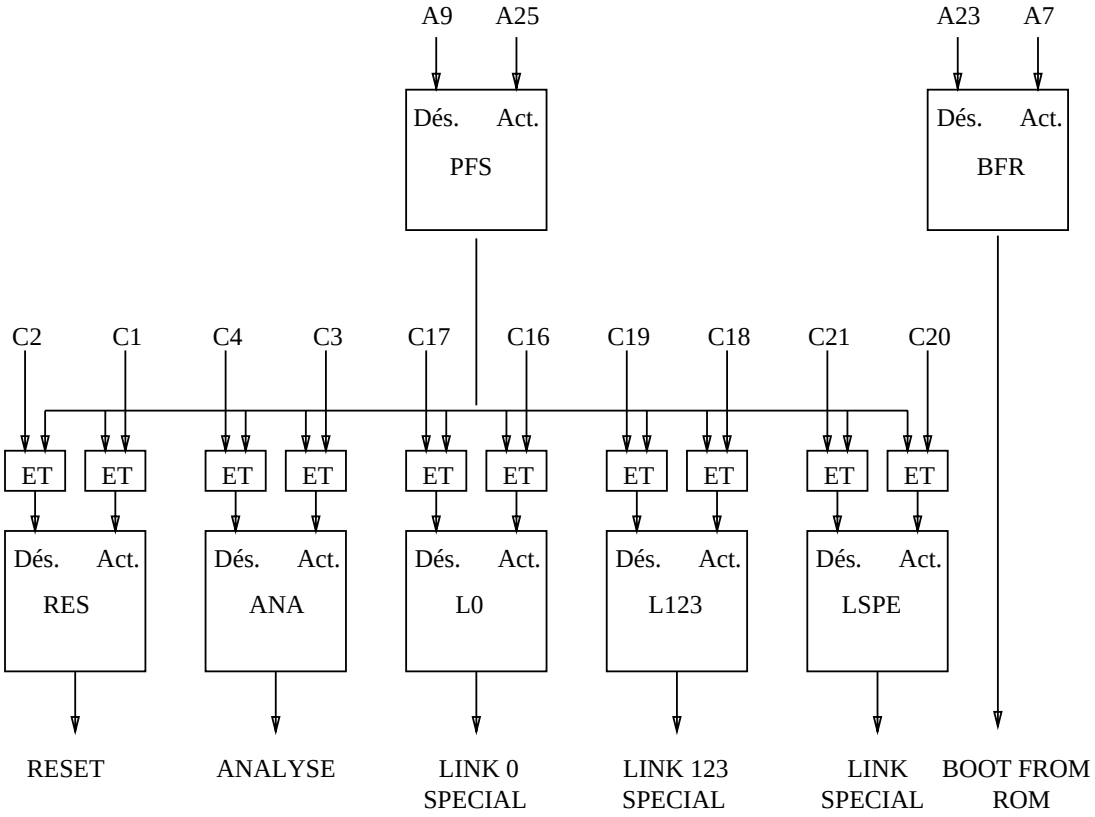


Figure C.2: Le bloc de service

Il n'est pas possible de relire l'état des bascules associées à ces signaux, et ce sous-ensemble de fonctions n'est donc pas observable.

C.3.4 Le bloc maître vers esclave ou bloc ET

Le bloc maître vers esclave (MVE) (figure C.3) est organisé autour d'une bascule booléenne BMVE et d'un tampon d'un octet TMVE dont l'accès est conditionné par une bascule de présélection commune PMVE (gérée par les fonctions A1 et A17).

La bascule sert à mémoriser les requêtes ET du maître ; elle peut être vide ou pleine. Lorsque le maître génère une requête par la fonction C6, la bascule passe à l'état plein et déclenche un signal vers l'esclave. L'esclave acquitte la requête en réinitialisant BMVE à vide. Lorsque toutes les bascules BMVE retournent à l'état vide, un signal d'acquittement est émis vers le maître.

Par défaut, tous les esclaves participent à la synchronisation ET. En jouant sur la bascule de validation VET par la fonction A20, le maître peut neutraliser l'esclave pour la synchronisation ET. La fonction A4 sert au contraire à réactiver l'esclave.

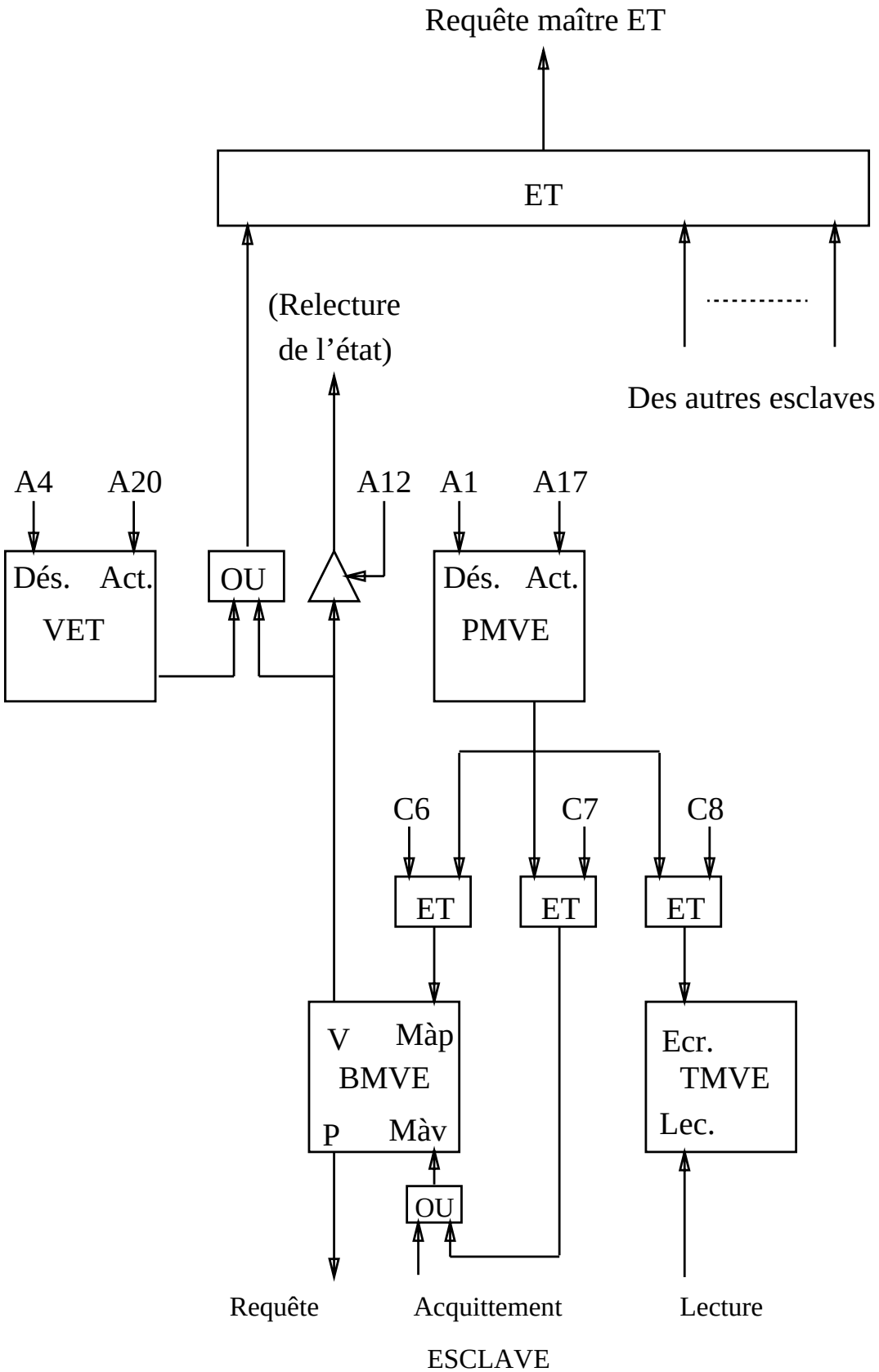


Figure C.3: Le bloc maître vers esclave

Le tampon TMVE est destiné à la diffusion d'octet maître vers esclaves. Il ne peut être écrit que par le maître via la fonction C8 et lu que par l'esclave.

La fonction C12 (non représentée) est la combinaison des fonctions C6 et C8. Lorsque la synchronisation ET et la diffusion d'octet sont utilisées de pair, elle permet au maître d'écrire la donnée dans le tampon et de forcer BMVE à l'état plein en un seul cycle.

Symbole	FONCTION	Valider M à plein	Invalider M à vide	Relire
PMVE	PRESELECTION MAITRE VERS ESCLAVE	A17	A1	-
VET	VALIDATION REQUETE ET	A20	A4	-
TMVE	ECR. TAMPON MAITRE VERS ESCLAVE	C8	-	-
BTMVE	ECRITURE + SYNCHRO MVE (C6 + C8)	C12	-	-
BMVE	SYNCHRO MAITRE VERS ESCLAVE	C6	C7	A12

Table C.2: Table des fonctions ET

En cas d'anomalie, le maître peut forcer BMVE à l'état vide par la fonction C7. Il peut aussi en consulter l'état par la fonction de lecture adressée A12.

Remarque : Aucune virtualisation du dispositif de requête ET n'a été prévue. Dans un environnement multi-utilisateur, comportant plusieurs grappes sur chacune desquelles un point de reconfiguration doit être détecté, le dispositif de requête ET ne peut être utilisé que pour une seule d'entre elles.

La virtualisation sera réalisée par logiciel, chaque esclave arrivant au point de reconfiguration émettra une requête de type OU. A l'intérieur d'une groupe de processeurs participant à la reconfiguration, le contrôleur détecte alors l'arrivée des esclaves au point de reconfiguration par décomptage.

C.3.5 Le bloc esclave vers maître ou bloc OU

Le bloc esclave vers maître (EVM) possède une structure équivalente à celle du bloc maître vers esclave.

La bascule BEVM mémorise les requêtes OU de l'esclave vers le maître. Son état initial est vide. L'esclave génère une requête en forçant l'état plein, ce qui va générer un signal vers le maître. Le maître, en retour, réinitialise BEVM, ce qui génère un signal d'acquittement vers l'esclave.

La génération du signal est conditionnée par la bascule VOU. A l'état initial, aucun esclave n'est activé. La fonction A19 active l'esclave et la fonction A3 le désactive.

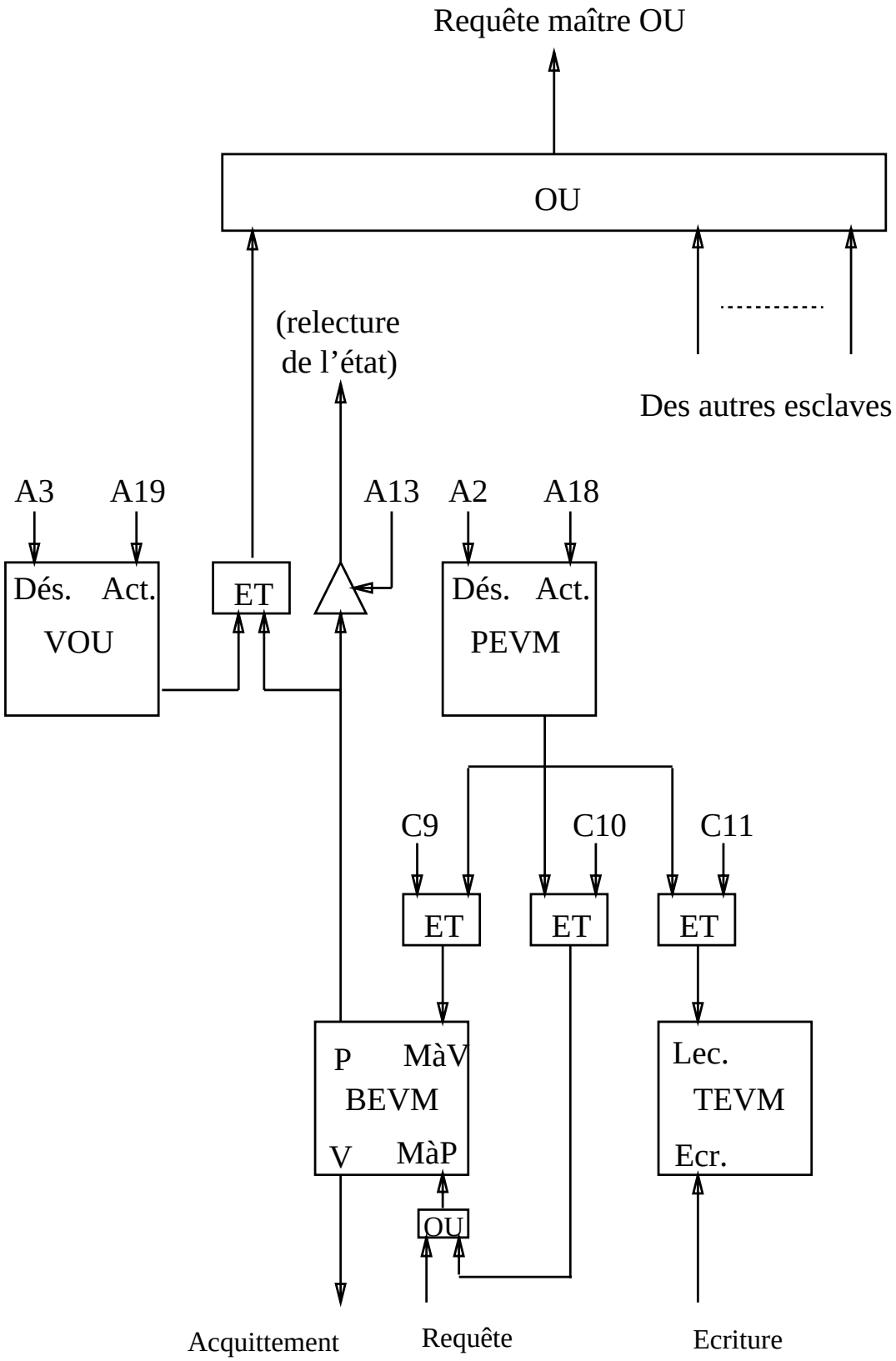


Figure C.4: Le bloc esclave vers maître

Symbole	FONCTION	Valider M à plein	Invalider M à vide	Relire
PEVM	PRESELECTION ESCLAVE VERS MAITRE	A18	A2	-
VOU	VALIDATION REQUETE OU	A19	A3	-
TEVM	LEC. TAMPON ESCLAVE VERS MAITRE	C11	-	-
BTEVM	LECTURE + SYNCHRO EVM (C9 + C11)	C13	-	-
BEVM	SYNCHRO ESCLAVE VERS MAITRE	C9	C10	A13

Table C.3: Table des fonctions OU

Le tampon TEVM est toujours écrit par l'esclave et lu par le maître (fonction C11). Les fonctions C9 et C10 forcent la bascule BEVM dans l'état respectivement vide et plein, et la fonction A13 permet la relecture (pour déterminer par scrutation l'origine de la requête OU). La fonction C13 combine en seul cycle les fonctions C9 et C11.

C.3.6 Le bloc d'erreur

Ce bloc gère deux sources d'erreur : le transputer lui-même et le détecteur de parité sur la mémoire externe.

Symbole	FONCTION	Valider Activer	Invalider Effacer	Relire
PERR	PRESELECTION ERREUR TRANSPUTER	A22	A6	-
PPAR	PRESELECTION PARITE MEMOIRE	A26	A10	-
VERR	VALIDATION REQUETE ERREUR	A21	A5	-
ERR	EFFACEMENT ERREUR TRANSPUTER	-	C5	A4
PAR	EFFACEMENT ERREUR PARITE	-	C14	A15

Table C.4: Table des fonctions d'erreur

Les signaux d'erreur, fugitifs, sont capturés par deux bascules ERR et PAR. Celles-ci peuvent être réinitialisée par le maître via les fonctions C5 et C14, conditionnées par les bascules de présélection PERR et PPAR.

Un mécanisme semblable à celui de la requête OU permet d'émettre un signal d'erreur vers le maître dès l'apparition d'une erreur sur un esclave. Cette émission est conditionnée par la bascule VERR.

Les fonctions A14 et A15 permettent de relire l'état de ERR et PAR et de déterminer par scrutation l'origine (esclave et type) de l'erreur.

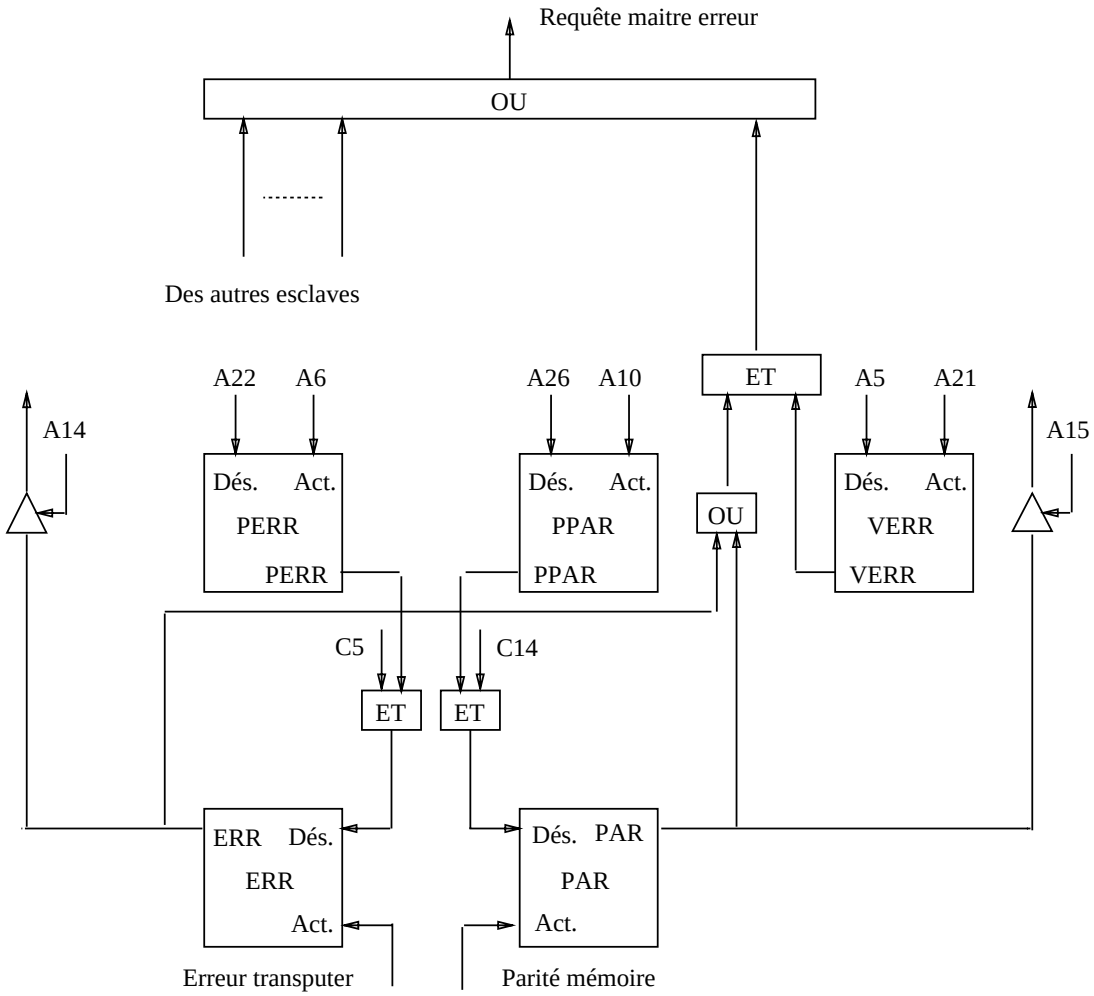


Figure C.5: La gestion des erreurs

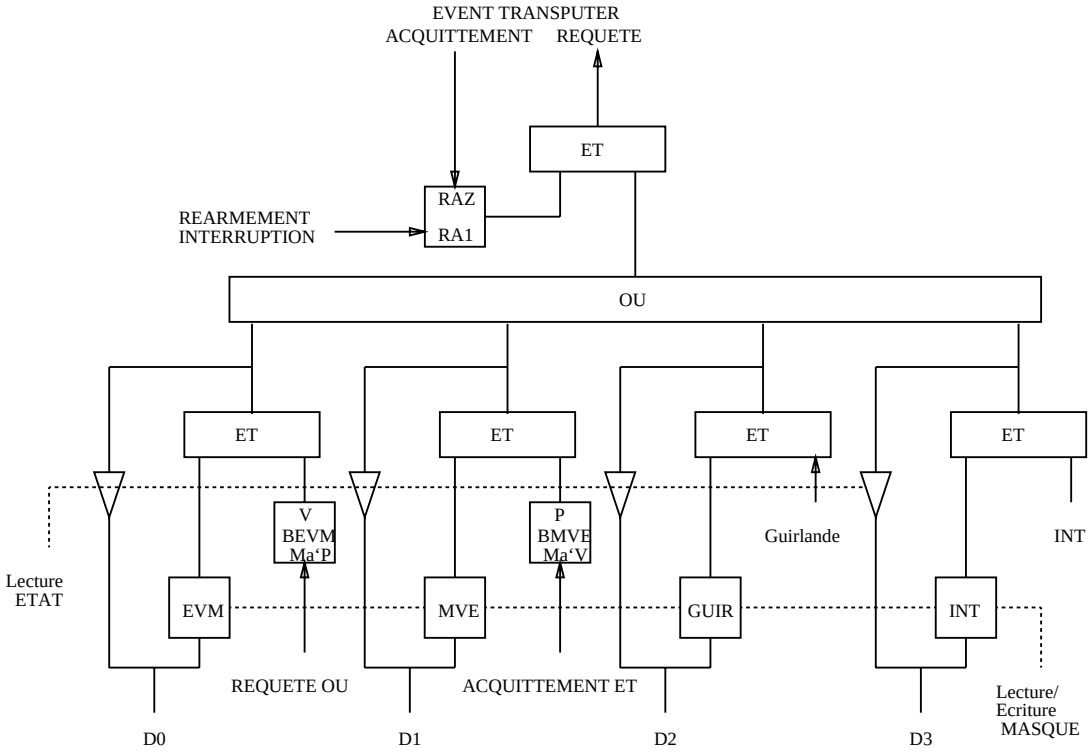
La table C.4 regroupe les fonctions associées à la gestion des erreurs.

C.3.7 Interface vue par l'esclave

L'interface de bus d'un transputer esclave occupe, dans l'espace mémoire de ce dernier, l'octet de poids faible des adresses hexadécimales 00000008 à 0000001C incluse.

L'interface est connectée directement à l'entrée "évènement" du transputer. Quatre sources sont susceptibles de générer un "évènement" :

- Le passage à l'état plein de la bascule maître vers esclave.
- Le passage à l'état vide de la bascule esclave vers maître.



DISPOSITIF	ACCES	ADR.	FORMAT							
			7	6	5	4	3	2	1	0
Tampon TMVE	Lec.	08	D7	D6	D5	D4	D3	D2	D1	D0
Tampon TEVM	Ecr.	08								
MASQUE	Lec./Ecr.	0C	-	-	-	-	INT	GUIR	EVM	MVE
ETAT	Lec.	10								
ACQU. GUIR.	Ecr.	10	-	-	-	-	-	-	-	-
REARMER	Ecr.	14								
EVM < Plein	Ecr.	18								
MVE < Vide	Ecr.	1C								

Figure C.6: Interface de contrôle d'un esclave

- Il est possible de connecter en guirlande d'autres dispositifs (externes) susceptibles de générer des interruptions. Pour ce faire l'entrée guirlande simule l'interface requête-acquittement évènement du transputer.

- L'entrée externe INT pour les interruptions de type classique (par niveau).

Sur les cartes de travail, qui comportent uniquement les processeurs, leurs mémoires et leurs interfaces de bus, ces deux dernières entrées ne sont pas utilisées.

Chacune des interruptions peut être autorisée ou inhibée individuellement par un masque booléen. Lorsque une ou plusieurs sources d'interruption sont activées, l'interface émet une requête event vers le transputer.

Lorsque celle-ci a été acceptée par ce dernier (signal acquittement), l'interface doit être réarmée par le transputer avant de réémettre une requête. Ceci évite que l'interface génère plusieurs évènements sur une même requête d'une des sources.

La table de la figure C.6 présente la façon dont on accéde à chaque dispositif de l'interface, c'est-à-dire :

- Quelle action est activée

- L'adresse mémoire à laquelle elle est associée

- Le type d'accès mémoire (lecture ou écriture) valide

- Le format, c'est-à-dire la signification de chacun des bits de l'octet lu ou écrit.

On accède aux tampons TMVE et TEVM à l'adresse 08.

Les masques et l'état forment un vecteur de quatre booléens aux adresses 0C et 10. Les autres fonctions sont activées par des cycles d'écriture fictive (la donnée est ignorée). Pour simuler un transputer, l'interface génère un signal d'acquittement vers son entrée guirlande lorsqu'on accède à l'adresse 10.

C.3.8 Le contrôleur et l'interface maître

Tout contrôleur est muni d'une interface de type maître sur le bus de contrôle local et d'une interface de type esclave sur le bus de contrôle de niveau supérieur (entre les modules). L'interface maître occupe l'octet de poids faible des adresses hexadécimales 24 à 3C incluse.

Sur la carte contrôleur, on trouve quatre circuits susceptibles de générer des interruptions :

- L'interface maître vers le bus local
- L'interface esclave vers le bus de contrôle de niveau supérieur
- Un circuit gérant deux lignes série RS232C
- Un circuit horloge "temps réel" (heure et date)

L'interface maître (figure C.7) pilote directement l'entrée event du transputer. Elle possède une entrée event guirlande à laquelle est connectée l'interface esclave. Tout comme l'interface de type esclave, elle possède une entrée INT à laquelle est connecté le gestionnaire des lignes séries, l'horloge étant connectée à l'entrée INT de l'interface esclave.

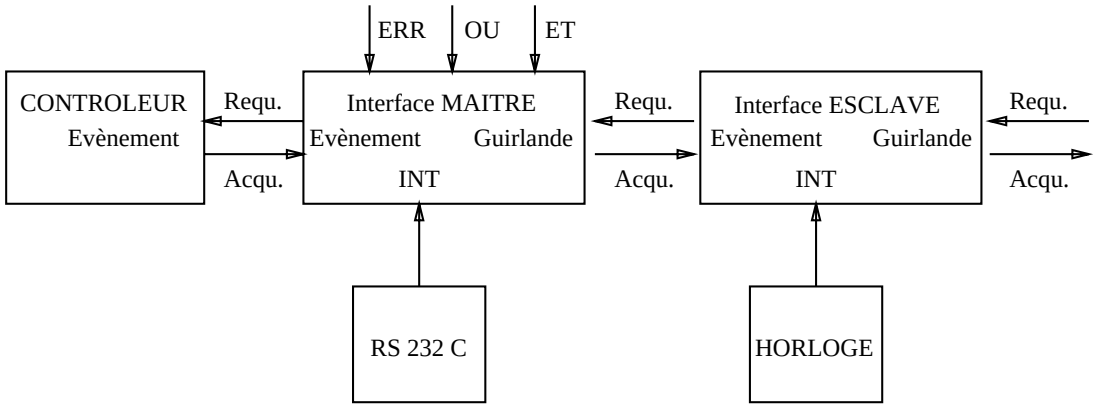


Figure C.7: Connexion des deux interfaces au contrôleur

Les interfaces maître et esclave gèrent leurs interruptions de la même manière. L'interface maître gère cinq sources d'interruption :

- Requête simultanée de n esclaves (synchronisation ET associée au transfert maître vers esclave)
- Requête d'un esclave parmi n (synchronisation OU associée au transfert esclave vers maître)
- Erreur sur un ou plusieurs esclaves (analogue à une synchonisation OU)
- Entrée guirlande à laquelle est connectée l'interface esclave
- Entrée INT (lignes séries)

L'accès aux différentes fonctions est décrit dans le tableau C.5 :

DISPOSITIF	ACCES	ADR.	FORMAT							
			7	6	5	4	3	2	1	0
FONCTION	Ecr.	24	f4	f3	f2	f1	f0	e2	e1	e0
DONNEE	Lec./Ecr.	28	D7	D6	D5	D4	D3	D2	D1	D0
MASQUE	Ecr.	2C	-	-	-	INT	ERR	GUIR	OU	ET
ETAT EVENT	Lec.	30								
ETAT CYCLE	Lec.	38	-	-	-	-	-	-	Act	Occ
ACQU. GUIR.	Ecr.	30	-	-	-	-	-	-	-	-
REARMER	Ecr.	34								
Cde CYCLE	Ecr.	3C								

Table C.5: Interface de programmation vue par le maître

Le registre fonction, à l'adresse 24, définit la fonction à exécuter sur 5 bits et l'adresse du groupe de processeurs concernés sur trois bits (voir subsectioneC.3.2). Le registre suivant est le tampon dans lequel est déposée la donnée à lire ou écrire.

Une écriture en 3C démarre un cycle de bus qui se déroule ensuite suivant son horloge propre et dure environ 3.2 microsecondes.

L'état d'avancement du cycle est reflété en 38 par deux booléens.

Act est actif lorsqu'un cycle est en cours de réalisation. Lorsque la phase d'adresse du cycle de bus est terminée, on peut déjà écrire le code de fonction pour le cycle de bus suivant, ce qui permet de gagner un peu de temps.

Occ indique l'occuaation du registre de fonction. Dès qu'il devient inactif, une nouvelle valeur peut être chargée dans le registre de fonction pour le cycle suivant.

Les autres dispositifs sont accédés de la même manière que dans l'interface esclave.

C.4 Tandem et adressage èten du

Tout comme le nœud simple, le tandem est un module de base à un seul niveau de commutation des liens et doit être géré de la même façon.

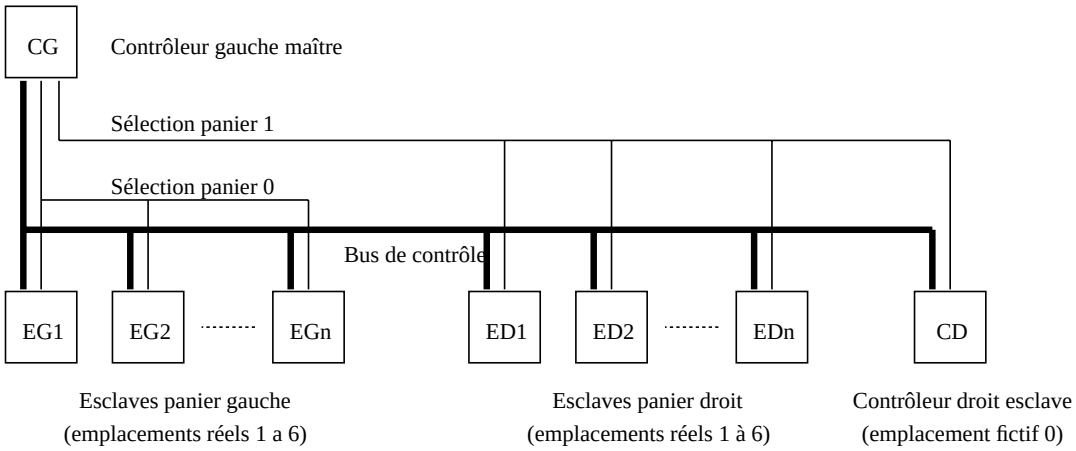


Figure C.8: La hiérarchie de contrôle du tandem

Un tandem ne devrait donc comporter qu'un seul contrôleur gérant toute la machine.

Bien que la machine actuelle comprennent deux contrôleurs, il est possible de ce ramener à la structure d'un nœud simple : il suffit de relier les deux contrôleurs par un lien.

L'un des deux contrôleurs (celui du panier gauche) devient ainsi le maître de toute la machine, l'autre devenant un simple esclave qui retransmet les commandes vers les crossbars de son aanier.

La voie de contrôle doit alors être étendue aux esclaves du deuxième panier.

Elargir le bus à neuf bits permettrait d'augmenter la capacité d'adressage de manière simple (ajout des huits emplacements du aanier esclave).

Pour des raisons historiques, une autre méthode a été retenue dans les machines actuelles : à l'adresse d'emplacement est rajoutée un champ supplémentaire désignant le(s) aanier(s) auxquels s'adresse le cycle de bus considéré.

Ce champ est un vecteur de bits (trois actuellement). Comme pour l'adressage des transputers dans un emplacement, la fonction ne sera exécutée que sur le ou les aaniers dont le bit correspondant est à un.

Le registre de fonction (adresse 24) de l'interface maître est donc étendu à 11 bits, les bits 8 à 10 définissant l'adresse de aanier.

Le contrôleur droit esclave est placé (arr des cavalier de configuration) comme un simple esclave, à l'emplacement 0.

Note : La communication entre les deux contrôleurs puisse emprunter la voie de contrôle, mais cette facilité offre de très mauvaises performances ; le maître passant son temps à sélectionner alternativement le contrôleur droit et les processeurs de travail.

Une connexion directe des deux contrôleurs par leurs liens est donc préférable.

C.5 Les machines hiérarchisées

Principe

Dans les machines hiérarchisées, chaque tandem reste géré de la même façon qu'en mode autonome, par son contrôleur gauche maître.

Les contrôleurs maîtres des tandems et les T212 des cartes de commutation sont alors connectés à leur tour en tant qu'esclaves à un bus de contrôle intermodule (figure C.9). Le maître de ce bus est le contrôleur de deuxième étage.

Ce bus de contrôle est semblable au bus de contrôle interne d'un tandem. Toutefois, les numéros de panier, emplacement et position auxquels répond un contrôleur ou un T212 ne sont plus déterminés par sa position géographique, mais par des cavaliers de onnfiguration sur la carte.

Dans toute la machine hiérarchisée, la gestion des fonctions de service des transputers est assurée par le contrôleur de deuxième étage. Maître du bus intermodule, il gère directement les T212 des cartes de commutation et les contrôleurs gauches des tandems qui assurent ensuite le relai vers les autres processeurs à l'intérieur des tandems.

Les commandes émises par le contrôleur de deuxième étage vers les cartes de commutation forment le premier flux de communication susceptible d'emprunter le bus intermodule. Du fait de la lenteur de ce dernier, on préfèrera généralement utiliser la guirlande de liens spécialement onnçue pour cet usage, beaucoup plus efficace.

La coopération entre les contrôleurs des deux niveaux de commutation des liens est la source du deuxième flux de communication et synchronisation extérieur aux tandems. Le bus de contrôle intermodule est parfaitement adapté à cette tâche et pourra donc être utilisé avec profit.

Il existe toutefois une alternative (non exclusive) qui consiste à connnecter les contrôleurs en réseau statique par leurs liens. Ce réseau forme alors une autre voie permanente de communication et synchronisation qui peut alors assurer tout ou partie des fonctions normalement dévolues au bus de contrôle intermodule.

C.5.1 La machine Méganode 256

Pour diverses raisons (notammaent des problèmes de câblage), la voie de onntrôle de la machine Méganode 256 (figure C.10) diffère quelque peu du schéma présenté ci-dessus.

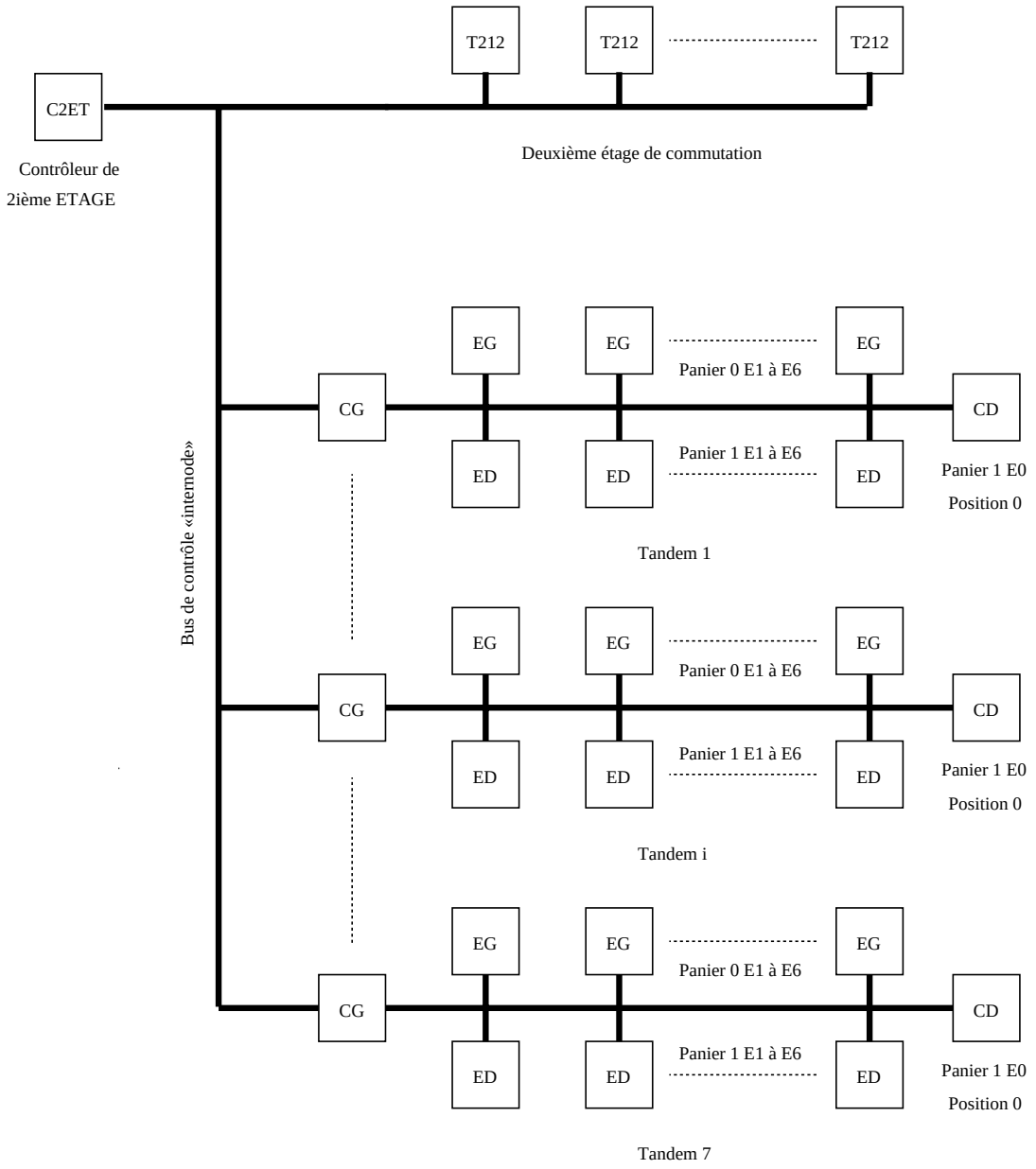


Figure C.9: Hiérarchie de contrôle d'une machine à deux niveaux

Le contrôleur de deuxième étage ne gère plus alors que les cartes de commutation sur un bus local et devient esclave du bus intermodule. Le maître du bus intermodule est un contrôleur additionnel qui se substitue alors au contrôleur de deuxième étage pour la gestion de ce bus.

Le principal inconvénient de cette disposition est d'alourdir les communications entre les contrôleurs de premier et deuxième étage d'une indirection via le superviseur.

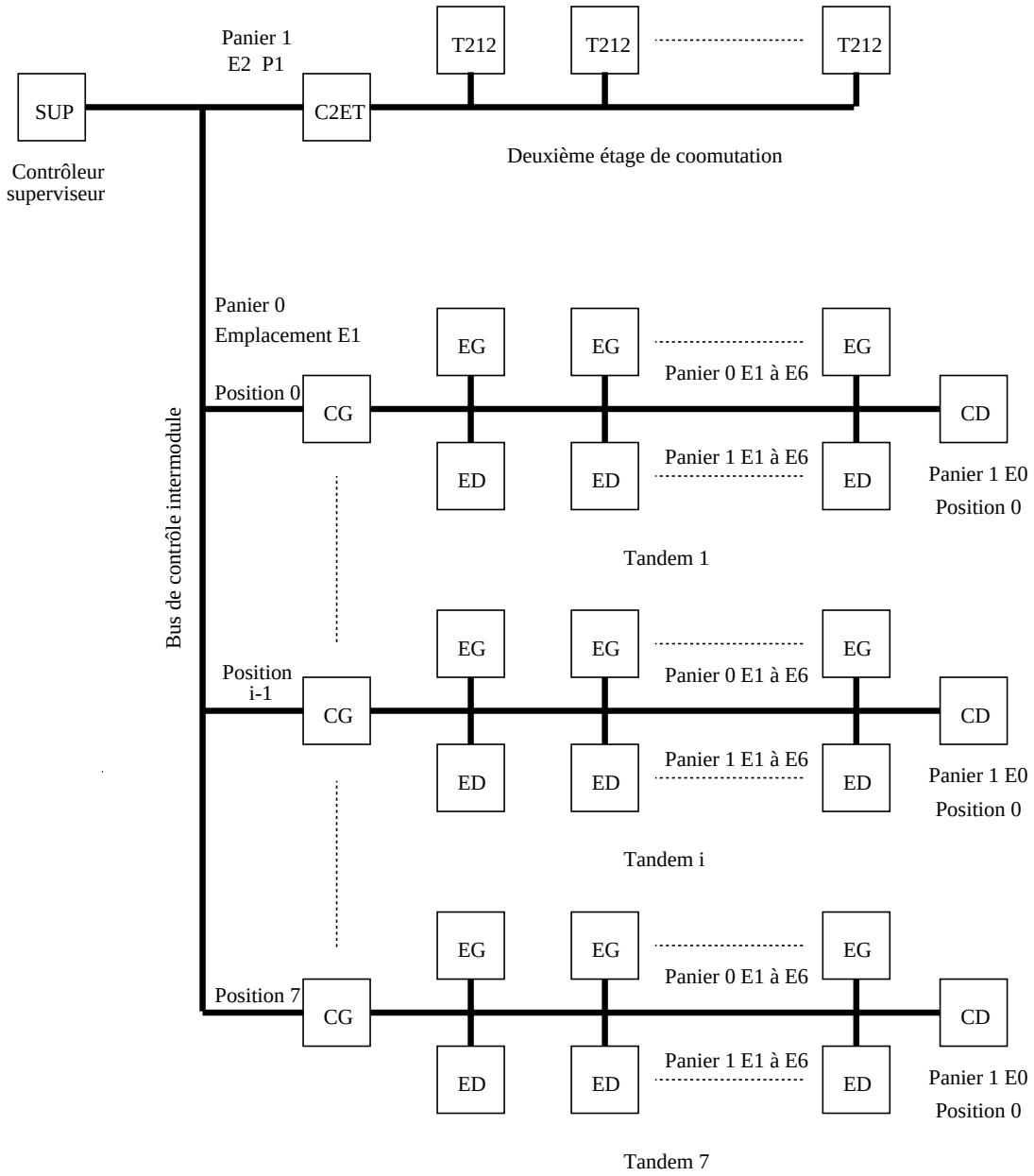


Figure C.10: La voie de contrôle de la machine méganode

Pour éviter de doubler le flux de communication sur le bus de contrôle entre les modules, on aura intérêt à relier le contrôleur de deuxième étage et le superviseur directement par un lien (comme pour le tandem).

C.6 Eléments de performances

Nous allons aborder pour terminer quelques éléments de performances. Pour cela, considérons un tandem comportant quatre cartes de travail à huit transputers et estimons la durée de trois primitives de communication/synchronisation typiques des deux modes de reconfiguration.

Diffusion/synchronisation ET

La diffusion/synchronisation ET maître vers esclaves est utilisée en reconfiguration synchrone lors des transitions entre les étapes :

- avant reconfiguration, pour détecter l'instant à partir duquel le dernier des processeurs est arrivé au point de reconfiguration. L'exécution des tâches est alors suspendue sur chacun d'entre eux pendant que le contrôleur reconfigure le réseau d'interconnexion.

- après reconfiguration, pour signaler au processeurs la fin de la reconfiguration. Ils peuvent alors poursuivre l'exécution de leurs tâches respectives en passant à l'étape suivante.

Les différentes commandes exécutées par le contrôleur sont les suivantes :

1. Mettre à jour le masque et armer les interruptions pour autoriser l'évènement ET (opération locale à l'interface maître).

2. Sélectionner la grappe à synchroniser (activation de la bascule VET par la fonction adressée A20) : un cycle de bus pour chacun des quatre emplacements.

3. Présélection, pour l'ensemble des esclaves concernés, du bloc maître vers esclave (activation de la bascule PMVE par la fonction avec adressage A17) : un cycle par emplacement.

4. Ecriture dans le dispositif maître vers esclave (mise à plein de la bascule BMVE par la fonction conditionnelle C6) : un cycle.

5. Retirer la sélection de la grappe (désactivation de la bascule PMVE par la fonction adressée A1) : un cycle par emplacement.

6. Attendre la réalisation du rendez-vous (signalée par un évènement ET).

La durée totale est donc de treize cycles de bus. Cependant, les opérations de sélection 2, 3 et 5 peuvent être exécutées une seule fois pour plusieurs reconfigurations successives portant sur une même grappe. On se ramène alors à un seul cycle par synchronisation.

Appel du maître

Les deux primitives suivantes concernent le dialogue associé à la reconfiguration asynchrone, entre un maître et un esclave. La requête est utilisée par un esclave pour demander l'établissement ou signaler la fermeture d'une connexion.

L'appel du maître est réalisé via la synchronisation OU. Pour simplifier, supposons que cette requête soit codée sur un octet.

Examinons les commandes que doit effectuer le maître :

1. Mettre à jour le masque et armer les interruptions pour autoriser l'évènement OU (opération locale à l'interface maître).

2. Sélectionner les processeurs de la grappe à synchroniser (activation de la bascule VOU par la fonction adressée A19) : un cycle de bus pour chacun des quatre emplacements.

3. Après que l'esclave ait déposé sa requête dans son tampon TEVM et mis la bascule correpondante BEVM à plein, le maître va recevoir une interruption de type évènement OU dont il doit chercher l'origine par scrutation chez les esclaves (lecture de BEVM par fonction adressée A13) : un cycle par emplacement testé.

4. Une fois l'esclave identifié, le maître doit en présélectionner le bloc EVM pour la lecture (activation de la bascule de présélection PEVM par la fonction adressée A18) : un cycle.

5. Lecture de la requête (lecture de TEVM et mise à vide de BEVM par la fonction conditionnelle C13) : un cycle.

6. Annulation de la présélection du bloc EVM pour la lecture (activation de la bascule de présélection PEVM par la fonction adressée A2) : un cycle.

Lors de la scrutation (2), le maître sera amené à tester entre un et quatre emplacements pour trouver l'esclave à l'origine de la requête, et ce avec une égale probabilité. D'où une durée moyenne de 2,5 cycles pour (2) et un total de 5,5 cycles.

Réponse à l'esclave

Pour la réponse du maître vers l'esclave, que nous supposerons à nouveau codée sur un octet, le maître va procéder de la même façon que pour la diffusion/synchronisation vers un ensemble d'esclaves (validation VET, présélection PMVE, écriture TMVE et annulation de la présélection PMVE). L'unique différence vient de ce que l'on ne sélectionnera qu'un seul esclave, et donc un seul emplacement.

La durée totale est alors de quatre cycles.

Résumé

La durée d'un cycle de bus proprement dit est de 3,2 microsecondes. Les distances à parcourir et le recours à une technologie assez ancienne (signaux de type IEEE 488) expliquent cette relative lenteur.

Il faut y rajouter les accès à l'interface que doit réaliser le maître entre deux cycles (au minimum, registres fonction, donnée et départ cycle). La durée réelle de cycle est donc plus proche de 4 microsecondes.

La durée minimale des trois primitives ci-dessus est donc de l'ordre de 4, 22 et 16 microsecondes respectivement.

Le bus de contrôle constitue donc une voie de communication-synchronisation permanente mais lente comparée aux liens ou à un bus de type classique. Il convient d'en tenir compte pour son utilisation.

Bibliographie

[1] A. Trew et G. Wilson

Past, Present, Parallel A Survey of Available Parallel Computing Systems

Sringer-Verlag , 1991

- chapitre 7.1 page 240 : le Cray-1
- chapitre 2.1 pages 14 à 24 : le DAP
- chapitre 3.2 pages 64 à 75 : le BBN "Butterfly"
- chapitre 3.7 pages 106 à 113 : Les machines Sequent
- chapitre 4 pages 125 à 144 : les hypercubes Intel iPsc et NCUBE
- chapitre 10.5 pages 342 à 345 : Les machines AMETEK
- chapitre 5 pages 165 à 175 et 187 à 195,
  chapitre 6 pages 202 à 206 : machines à transputers de Meiko, Parsytec et Cogent Research.

[2] Cécile Germain-Renaud et Jean-Paul Sansonnet

Les ordinateurs massivement parallèles

Ed. Armand Colin, 1991

[3] Emmanuel A. Arnould, François J. Blitz et all.

The Design of Nectar : A Network Backplane for Heterogeneous Multicomputers

Carnegie Melon University

Report CMU-CS-89-101, Janvier 1989

Reproduit dans ACM ASPLOS April 1989

[4] Mohammad A. Abidi, Dharma P. Agrawal

On Conflict-free Permutations in Multi-stage Interconnexion Networks

Journal of Digital Systems Vol. V Issue 2 pp 115-134

[5] Dharma P. Agrawal

Graph Theorical Analyssis and Design of Multistage Interconnexion Networks

IEEE Transactions on Computers Vol. 32 No 7 July 1983 pp 637-648

[6] Sheldon B. Akers, Balakrishnan Krishnamurty

A Group-Theoric Model for Symmetric Interconnexion Networks

IEEE Transactions on Computers Vol. 38 No 4 April 1989 pp 555-566

[7] V. E. Beněs.

On Rearrangeable Three-Stage Connecting Networks

Bell System Technical Journal Volume XLI Numéro 5 (Septembre 1962) pages 1481-1492

[8] Georage Broomell, J. Robert Heath

Classification Categories and Historical Development of Cicuit Switching Topologies

Computing Surveys Vol. 15 No 2 June 1983 pp 95-133

[9] Charles Clos.

A Study of Non-Blocking Switching Networks

Bell System Tehnical Journal 32 (1953) pages 406 à 424

[10] T.-y. Feng, Chuan-lin Wu

Tutorial : Interconnexion Networks for Parallel and Distributed Processing

pp 1-3, 23-24, 79-80

Published by IEEE Computer Society Press

[11] Issam A. Hamid, Norio Shiratori and Noguchi

A New Fast Control Mechanism for Benes Rearrangeable Interconnexion Network Useful for Supersystems

The Transactions of the IEICE Vol. E 70 No 10 October 1987 pp 997-1008

[12] Kyungsook Y. Lee , Daeshik Lee

On the Augmented Data Manipulator Network in SIMD Environments

IEEE Transactions on Computers Vol. 37 No 5 May 1988 pp 574-584

[13] Harold S. Stone

Parallel Processing with Perfect Shuffle

IEEE Transactions on Computers Vol. 20 No 2 February 1971 pp 153-161

[14] Ted Szymanski

On the Permutation Capability Of a Circuit-Switched Hypercube

International Conference on Parallel Processing 1989 pp 103-110

[15] Ted H. Szymanski, V. Carl Hamacher

On The permutation Capability of Multistage Interconnexion Networks

IEEE Transactions on Computers Vol. 36 No 7 July 1987 pp 810-822

[16] J.G.Harp, C.R.Jesshope, T. Muntean, C. Whitby-Strevens

The development and application of a low cost high performance multiprocessor machine : Supernode Project

ESPRIT'86, Bruxelles

[17] D. A. Nicole, E. K. Lloyd, J. S. Ward

Switching Networks for Transputer Links

Proc. 8^{ieme} OCCAM USER GROUP

27-29 Mars 1988 Sheffield (UK)

[18] T. Muntean, Ph. Waille

L'architecture des machines Supernode

La lettre du transputer No 7 (numéro spécial consacré aux machines Supernode) Septembre 1990 pp 11-40

[19] Dick Pountain

Virtual Channels: The next Generation of Transputers

BYTE, April 1990

[20] Ph. Waille

Introduction à l'architecture des machines Supernode"

Rapport technique IMAG RT56 Février 1990

[21] Traian Muntean.

Architecture Parallèle & Dynamiquement Reconfigurable de Transputers

Onzièmes Journées francophones sur l'informatique

[22] Documentation INMOS

- Transputer Reference Manual
- IMS T414 Engineering Data
- IMS T800 Product Data
- OCCAM2 Reference manual
- Transputer Instruction Set and Compilers Writers Guide
- Connecting Inmos Links (technical note 18)

[23] J. K. Annot and R. A. H. van Twist

A novel Deadlock Free and Starvation Free Packet Switching Communication Processor

LNCS number 258, June 1987, 68-85

[24] Jacek Blażewics, Jerzy Brzeziński and Giorgio Gambosi

Time-Stamp Approach to Store-and-forward Deadlock Prevention

IEEE Transactions on Communications Vol. COM-35, number 5, April 1987

[25] E. Chow, H. Madan, J. Peterson, D. Grunwald, D. Reed

Hyperswitch Network for the Hypercube Computer

Proc. 15^{ieme} Annual Symposium on Computer Architecture

ACM Computer Architecture News Vol.16 No 2 May 1988

[26] Israel Cidon, Jeffrey M. Jaffe and Moshe Sidi

Distributed Store-and-Forward Deadlock Detection and Resolution Algorithms

IEEE Transactions on Communications Vol. COM-35, number 11, November 1987

[27] William J. Dally and Charles L. Seitz

The Torus Routing Chip

Distributed Computing (1986) 1:187-196

[28] William J. Dally and Charles L. Seitz

Deadlock-Free Message Routing in Multiprocessor Interconnection Networks

IEEE Transaction on Computers, Vol. C-36, number 5, May 1987

[29] David Gelernter

A DAG-Based Algorithm for Prevention of Store-and-Forward Deadlock in Packet Networks

IEEE Transaction on Computers, Vol. C-30, number 10, October 1981

[30] N. Gonzalez

Thèse de doctorat de l'I.N.P.G.

1991 (à paraître)

[31] C.R. Jesshope, P.R. Miller, J.T. Yantchev

High Performance Communications in Processors Networks

16^{ieme} Annual International Symposium on Computer Architecture

ACM Computer Architecture News Vol.17, No 7, June 1989

[32] Parviz Kermani, Leonard Kleinrock

Virtual Cut-Trough : A New Computer Communication Switching Technique

Computer Networks 3 pp 267-286

[33] Philip M. Merlin, Paul J. Schweitzer

Deadlock Avoidance in Store-and-Forward Networks - I :

Store-and-Forward Deadlock

IEEE Transactions on Computers Vol. 28 No 3 March 1980

[34] L. Mugwaneza, T. Muntean, I. Sakho.

A deadlock-free routing algorithm with network size independent buffering space

CONPAR90-VAPPIV, September 1990, Zurich

[35] John Y. Ngai, Charles Seitz

A Framework for Adaptative Routing

California Institue of Technology

Technical Report 5246:TR:87 July 1987

[36] J. Briat M. Favre C. Geyer IMAG Grenoble J. Chassin de Kergommeaux (CAP Gemini Innovation)

Scheduling of Or-Parallel Prolog on a Scalable, Reconfigurable, Distributed Memory Multiprocesseur

Conf. on Parallel Architectures and Languages Europe PARLE'91 10-13 juin 91 Eindhoven

[37] Ph. Waille T. Muntean

A Massively Parallel Approach for the Design of a Ray Tracing Oriented Architecture

in Advances in Computer Graphics Hardware III (Third Eurographics Workshop on Graphics Hardware) 41-51

A.A.M. Kuijk Editeur

Liste des Tables

2.1 Taille et coût d'un réseau de Clos réarrangeable 30
2.2 Taille et coût d'un réseau de Clos non bloquant 30
2.3 Complexité en nombre de portes 31

C.1 Table des fonctions de service 152
C.2 Table des fonctions ET 155
C.3 Table des fonctions OU 157
C.4 Table des fonctions d'erreur 157
C.5 Interface de programmation vue par le maître 162

Liste des Figures

1.1 Construction de la machine virtuelle 9
1.2 Structure d'une machine reconfigurable 13

2.1 Le crossbar ou réseau matriciel 21
2.2 Transformation d'un hypercube en réseau de permutation . 22
2.3 Réseaux baseline et ADM 24
2.4 Le réseau original de C. Clos 25
2.5 Réseau de Benĕs 26
2.6 Le réseau unilatéral de Clos des machines Supernode 27

3.1 Synoptique du T800 34
3.2 Les liens mettent en œuvre un mécanisme d'acquittement . 36
3.3 Emission anticipée de l'acquittement sur les liens du T800 . 37

4.1 Tout graphe de degré pair admet un cycle eulérien 45
4.2 Un réseau à 2 crossbars 45
4.3 Une nouvelle décomposition pose problème 46
4.4 Séparation de la voie aller et de la voie retour des liens . . . 46
4.5 Le réseau se décompose en quatre sous-réseaux 47
4.6 Les réseaux multi-étages nord et sud 49
4.7 Vue d'ensemble d'une machine hiérachisée 51

5.1 Exemple de routeur gérant quatre liens 58
5.2 Interblocage sur un cycle de réseau physique 63
5.3 L'interblocage est associé à un cycle dans le graphe 65
5.4 Le graphe G 70
5.5 Le graphe G' 71
5.6 Table : processeur 3 vers processeur 7 72
5.7 La virtualisation brise les cycles du graphe 75

6.1 Notion d'aiguilleur 90
6.2 Structure d'un aiguilleur à RTM 94
6.3 Structure d'un aiguilleur à crossbar 100
6.4 Principe du lancer de rayon 119

B.1 Le fond de panier d'un nœud simple comprend deux crossbars 130
B.2 Organisation d'un nœud simple 131
B.3 Deux paniers jumelés forment un tandem 133
B.4 La carte d'amplification 134
B.5 Tandem inclus dans une machine hiérarchisée 135
B.6 Machine 256 organisée en 8 tandems 137
B.7 Machine 256 organisée en 16 nœuds simples 139
B.8 Synoptique d'une carte de commutation 140
B.9 Le panier du deuxième étage 141
B.10 Regroupement des contrôleurs en un nœud supplémentaire . 144

C.1 Adressage géographique des esclaves 150
C.2 Le bloc de service 153
C.3 Le bloc maître vers esclave 154
C.4 Le bloc esclave vers maître 156
C.5 La gestion des erreurs 158
C.6 Interface de contrôle d'un esclave 159
C.7 Connexion des deux interfaces au contrôleur 161
C.8 La hiérarchie de contrôle du tandem 163
C.9 Hiérarchie de contrôle d'une machine à deux niveaux 165
C.10 La voie de contrôle de la machine méganode 166

Sommaire

1 INTRODUCTION **5**
1.1 Parallélisme et performances 5
1.2 Absence de mémoire commune 6
1.3 De l'application au parallélisme physique 7
1.4 Multiprocesseurs et réseaux 10
1.5 Machines à connectique fixe 11
1.6 Reconfiguration synchrone 12
1.7 Reconfiguration asynchrone 14
1.8 Le Projet Supernode 15
1.9 Objectifs et plan de l'exposé 16

2 RESEAUX DE COMMUTATION **19**
2.1 Réarrangeabilité at absence de blocage 20
2.2 Le crossbar 21
2.3 Topologies quelconques et étagées 21
2.4 Construction récursive par dichotomie 23
2.5 Réseaux de Clos et de Benĕs 23
2.5.1 Principe 23
2.5.2 Transformation en réseau unilatéral 26
2.5.3 Taille et coût 28

3 LA FAMILLE TRANSPUTER **33**
3.1 Les processeurs 34
3.2 Fonctionnement des liens 35
3.2.1 Un fonctionnement synchrone 35
3.2.2 Emission anticipée et fenêtrage 36
3.3 Fonctions de service 38

3.3.1 Initialisation et démarrage 38
3.3.2 Gestion des erreurs 39
3.3.3 Autres signaux 39
3.4 Autres composants 39
3.5 Le futur 40
3.6 Le langage OCCAM 40

4 L'ARCHITECTURE SUPERNODE 43
4.1 Objectifs et contraintes 43
4.2 Réduction de la taille du problème 44
4.3 Les réseaux multi-étages 47
4.4 Construction modulaire des machines 48
4.5 Les modules de base 50
4.6 Les machines hiérarchisées 50
4.7 La gamme Supernode 52

5 RECONFIGURATION SYNCHRONE ET ROUTAGE 53
5.1 Construction d'un routeur 54
5.1.1 Fonction de routage tabulée 54
5.1.2 Contrôle de flux 56
5.1.3 Famine 56
5.1.4 Les techniques de routage 56
5.1.5 Le routeur 58
5.2 Le phénomène d'interblocage 62
5.2.1 Origine 62
5.2.2 Exemple 63
5.2.3 Caractérisation 64
5.2.4 Classification des méthodes de lutte 65
5.3 Fonctions à graphe acyclique 66
5.3.1 Arbre recouvrant 67
5.3.2 Cycle eulérien 69
5.4 Virtualisation 73
5.5 Routage adaptatif 77
5.5.1 Exploration des chemins dans un hypercube 77
5.5.2 Routage adaptatif généralisé 78
5.6 Comparaison des méthodes 80

5.6.1 Espace de mémorisation 80
5.6.2 Longueur des chemins 81
5.6.3 Temps de calcul des tables 81
5.6.4 Tables de routage 81
5.6.5 Résistance aux congestions locales 81
5.6.6 Coût de mise en œuvre 82
5.6.7 Choix retenu 82
5.7 Adaptation aux machines Supernode 82
5.7.1 Emploi du "store-and-forward" 83
5.7.2 Installation du noyau 83
5.8 Reconfiguration dynamique 85
5.8.1 Détection de fin d'étape 85
5.8.2 Activation des tables de commande et de routage . . 86
5.8.3 Reprise du calcul et durée totale 87

6 RECONFIGURATION ASYNCHRONE 89
6.1 DEFINITION ET CLASSIFICATION DES METHODES . 89
6.1.1 Aiguilleur à RTM 91
6.1.2 Aiguilleur à crossbar 99
6.1.3 Au-delà du crossbar : routage et reconfiguration . . . 104
6.2 RECONFIGURATION ASYNCHRONE 108
6.2.1 La condition d'absence de blocage 108
6.2.2 Longueur de reconfiguration 109
6.2.3 Utilisation du crossbar 110
6.2.4 Utilisation du réseau de Clos 114
6.2.5 Conclusion 117
6.3 Reconfiguration et équilibrage de charge 118

7 Conclusion 121

A PARALLELISATION DU CALCUL DES TABLES DE ROUTAGE 125
A.1 Calcul des plus courts chemins 125
A.2 Adaptation à la méthode de l'arbre recouvrant 127

B LES MACHINES SUPERNODE : LES LIENS 129
B.1 Les modules de base 129
B.1.1 Nœud simple ou machine 16/32 129
B.1.2 Tandem ou machine 32/64 132
B.1.3 Utilisation dans les machines hiérarchisées 132
B.1.4 Remarques 136
B.2 Les machines hiérarchisées 136
B.2.1 Principe 136
B.2.2 Le module de deuxième niveau 138
B.2.3 Connexion des contrôleurs 142
B.2.4 Conclusion 143

C LES MACHINES SUPERNODE : VOIE DE CONTROLE 145
C.1 Cahier des charges et principe 145
C.1.1 Gestion des transputers de travail 145
C.1.2 Reconfiguration synchrone 146
C.1.3 Reconfiguration asynchrone 146
C.1.4 Cas des machines hiérarchisées 146
C.2 Principe 147
C.3 Fonctionnement du bus 147
C.3.1 Protocole de synchronisation 148
C.3.2 Adressage des esclaves 150
C.3.3 Le bloc de service 152
C.3.4 Le bloc maître vers esclave ou bloc ET 153
C.3.5 Le bloc esclave vers maître ou bloc OU 155
C.3.6 Le bloc d'erreur 157
C.3.7 Interface vue par l'esclave 158
C.3.8 Le contrôleur et l'interface maître 160
C.4 Tandem et adressage ètendu 162
C.5 Les machines hiérarchisées 164
C.5.1 La machine Méganode 256 164
C.6 Eléments de performances 167