THESE DE DOCTORAT DE L'UNIVERSITE PARIS VI

Spécialité

INFORMATIQUE

Présentée par

JEAN-LOU DESBARBIEUX

Pour obtenir le titre de

DOCTEUR DE L'UNIVERSITE PARIS VI

sujet

Conception et réalisation d'un contrôleur réseau programmable pour machine parallèle de type " grappe de PC "

Soutenue le 28 juin 2000 devant le jury composé de :

Paul Feautrier ..... Rapporteur
Bernard Tourancheau ..... Rapporteur
Alain Greiner ..... Directeur de thèse
Bernard Lecussan ..... Président et examinateur
Pierre David ..... Examinateur
Jean-Louis Pazat ..... Examinateur
Franck Wajsbürt ..... Examinateur

Résumé

Dette Thèse porte sur les réseaux de communication pour machines parallèles de type « grappe de PC ». Effectuée dans le cadre de l'équipe ASIM du LIP6, cette étude présente l'architecture d'un contrôleur de réseau programmable à hautes performances. Ce contrôleur réseau est destiné à être utilisé pour la seconde génération de la machine MPC. Son architecture est basée sur l'utilisation de deux circuits intégrés :

- un circuit spécifique NOE, constitué d'un ensemble d'opérateurs permettant de réaliser les transferts de données entre la mémoire du PC et le réseau HSL ;
- un microcontrôleur standard (RISC de la famille PowerPC) qui supervise le fonctionnement des opérateurs du composant NOE et dont l'objectif est de mettre en œuvre le protocole de communication.

La structure de cette architecture nous amène à aborder trois aspects complémentaires :

- matériel, par la réalisation d'un ASIC (NOE) assurant la fonction de contrôleur réseau ;
- système, par la définition d'un micronoyau spécifique, BEE, destiné à gérer le parallélisme entre les différentes tâches du microcontrôleur ;
- protocole, en définissant un protocole de communication de base, RWU, dont l'objectif est de rendre accessible aux processus utilisateurs les performances matérielles du réseau.

Le circuit NOE constitue le premier « system On a Chip » réalisé dans notre laboratoire, et il possède un bus interne permettant de contrôler les différents opérateurs. Ces opérateurs disposent d'une grande autonomie de fonctionnement ce qui rend leur gestion plus aisée par le microcontrôleur. Cette caractéristique nous a permis de mettre au point une architecture logicielle originale pour le microcontrôleur qui permet d'offrir une grande souplesse de programmation sans dégradation des performances.
Nous avons développé une plate-forme de simulation VNOE afin d'évaluer les performances de cette architecture et de mettre au point une première version du micronoyau BEE.
Le circuit NOE a été réalisé dans le cadre du projet européen ARCHES. Il comporte 1.3 millions de transistors et a été fabriqué un second semestre 1999 par SGS-THOMSON en technologie CMOS 0.25 µm.

Mots clés

Machines parallèles,
HSL,
MPC,
NOE,
Architecture,
VLSI

Abstract

This thesis is about communication networks for “cluster of PCs” type of parallel machine. These works have been realized in the ASIM team of the LIP6 laboratory and concern the architecture of a high performances programmable network controller. This controller will be used in the second generation of the MPC machine and its architecture exploits two integrated circuits:

- a dedicated chip NOE which performs data transfers between host memory and HSL network;
- a standard micro-controller (PowerPC RISC Family) which controls NOE chip operators. The goal of the micro-controller is to implement the communication protocol.

This architecture leads us to tackle three complementary aspects:

- Hardware : the design of the VLSI NOE chip;
- System : the specification of a dedicated micro-kernel, called BEE which is used to manage with the existing parallelism between the different NOE operators;
- Protocol : RWU is a low-level protocol which can be directly accessed by UNIX process to take advantage of the network performances.

The internal architecture of the NOE chip is “System On a Chip” type and the problems due to control of the operators by an embedded processor are successfully solved by the BEE micro-kernel.

A specific simulation test bed called VNOE has been developed for the performance evaluation of the architecture. A first version of the BEE micro-kernel has been developed. A prototype of the NOE chip has been produced in the framework of ARCHES European project by SGS-THOMSON in .25µm CMOS technology and contains 1.3 million of transistors.

Keywords

Parallel machine ,
HSL,
MPC,
NOE,
VLSI

Remerciements

Je remercie tous ceux et celles qui ont contribué à ce travail ainsi que tous ceux et celles qui l'ont rendu possible, et par avance, tous ceux qui le continueront.

Je remercie également tous les membres de mon jury, mon directeur de thèse Alain Greiner et plus particulièrement Franck Wajsbürt.

Enfin, je tiens à remercier Florence et Lara pour le soin qu'elles ont apporté à la réalisation du pot.

Merci.

Table des matières.

Introduction _____ *7*

1.1. Cadre de cette thèse _____ **8**

1.2. Organisation du document _____ **9**

Problématique _____ *11*

2.1. Les besoins _____ **11**

2.1.1. Les besoins de puissance de calcul _____ 11

2.1.2. Les besoins d'espace mémoire _____ 11

2.1.3. Les besoins d'entrées-sorties _____ 12

2.2. Les machines parallèles comme réponse à ces besoins _____ **12**

2.3. Les machines à mémoire partagée _____ **13**

2.3.1. Les machines à passage de messages _____ 14

2.4. Les « grappes de PC » _____ **14**

2.4.1. Le nœud de calculs _____ 14

2.4.2. Le système d'exploitation _____ 16

2.4.3. Le réseau _____ 16

2.4.4. Le contrôleur réseau _____ 18

2.4.5. Le protocole réseau _____ 19

2.5. Les Objectifs _____ **20**

2.5.1. Les performances _____ 20

2.5.2. Une plate-forme d'expérimentation _____ 20

2.6. Les critères d'appréciation des « grappes de PC » _____ **20**

2.6.1. Le débit _____ 20

2.6.2. La latence _____ 21

2.6.3. Le surcoût pour le nœud de calculs _____ 23

2.6.4. La fiabilité _____ 23

2.6.5. L'extensibilité _____ 23

2.6.6. La programmabilité _____ 23

3. Etat de l'art _____ *25*

3.1. THE HIVE _____ **25**

3.1.2. Les performances _____ 27

3.1.3. Conclusion 31

3.2. La machine BIP/Myrinet du LHPC 32

3.2.1. Architecture de la machine 32
3.2.2. Le réseau de communication 32
3.2.3. Le lien Myrinet 33
3.2.4. Le Routeur 33
3.2.5. Le contrôleur réseau 34
3.2.6. Le protocole réseau 37
3.2.7. Les performances 39
3.2.8. Analyse 41

3.3. La machine MPC0 41

3.3.1. Architecture de la machine 41
3.3.2. Le réseau de communication 42
3.3.3. Le contrôleur réseau 43
3.3.4. Le protocole réseau 45
3.3.5. Le logiciel de bas niveau 46
3.3.6. Les performances 47
3.3.7. Analyse 49

3.4. Synthèse 50

3.4.1. Le débit 50
3.4.2. La latence 50
3.4.3. Le surcoût pour le nœud de calculs 51
3.4.4. La fiabilité 51
3.4.5. L'extensibilité 51
3.4.6. La programmabilité 52
3.4.7. Récapitulatif 52

3.5. Conclusion 52

4. Le protocole RWU 55

4.1. Les objectifs 55

4.2. Les principes de fonctionnement 56

4.2.1. Composant virtuel et composant réel 56
4.2.2. Zone de contrôle du composant virtuel 57
4.2.3. Interface avec le système d'exploitation 58
4.2.4. Les Canaux 58
4.2.5. Zone mémoire de communication 59
4.2.6. Les files d'émission et de réception 59

4.2.7. Gestion des adresses 59
4.2.8. Les Tables 59
4.2.9. Emission d'un message 60
4.2.10. Réception d'un message 61

4.3. La sécurité 63

4.4. VIA versus RWU 63

4.5. Conclusion 64

5. Le contrôleur de réseau NOÉ 67

5.1. Les objectifs 67

5.2. Architecture générale 68

5.2.1. Le bus d'entrée/sortie 68
5.2.2. Le réseau HSL 68
5.2.3. Le composant NOÉ 69
5.2.4. Le microcontrôleur 69

5.3. Le découpage en opérateurs 70

5.3.1. MEMRW 71
5.3.2. DMAR 74
5.3.3. DMAW 76
5.3.4. PB 77
5.3.5. PX 81
5.3.6. ITCTL 83
5.3.7. HSLM 85

5.4. Utilisation de NOÉ pour les transferts de paquets 85

5.4.1. Emission 85
5.4.2. Réception 85
5.4.3. Conclusion 86

5.5. Architecture interne 86

5.5.1. Résultats 88

5.6. Conclusion 89

5.6.1. L'architecture 89
5.6.2. Le composant NOÉ 90

6. Le micronoyau BEE 91

6.1. Le problème 91

6.2. Les solutions existantes 92

6.2.1. Le contrôleur gigabit Ethernet Alteon 92

6.2.2. Le LANAI de la carte myrinet 92

6.3. Présentation de BEE 93

6.3.1. Les Tâches 94

6.3.2. Les sémaphores 94

6.3.3. La commutation de tâche 97

6.3.4. L'ordonnanceur de tâche 99

6.3.5. Une API 100

6.3.6. Un exemple de tâche 101

6.4. Conclusion 102

7. Les Performances 103

7.1. Les objectifs 103

7.2. Architecture de la plate-forme VNOÉ 103

7.2.1. Modélisation du composant NOÉ 104

7.2.2. Modélisation des opérateurs 106

7.2.3. Modélisation des fifos 106

7.2.4. Mécanisme général de simulation 107

7.2.5. Modélisation de l'environnement 108

7.2.6. Le processeur de la station de travail 108

7.2.7. Le réseau HSL 108

7.2.8. La mémoire 109

7.2.9. Exemple de modélisation d'un opérateur 109

7.3. DDSLR, un protocole élémentaire 110

7.3.1. La tâche PCI 110

7.3.2. La tâche de transmission 110

7.3.3. La tâche de réception 110

7.4. Les Performances 111

7.4.1. Le débit 111

7.4.2. La latence 112

7.4.3. Incidence de la charge sur le contrôleur de réseau NOÉ 112

7.4.4. Taille minimale des Paquets 113

7.5. Les limites de la méthode de mesure 113

7.6. Conclusion 114

8. Conclusion _____ *115*

Bibliographie _____ *119*

CHAPITRE 1

Introduction

Les ordinateurs sont partout, ils ont envahi tous les secteurs de l'activité humaine. Ce sont certainement les premiers instruments de la recherche tous domaines confondus. Comment envisager la production industrielle sans ordinateur, ils prennent une place de plus en plus importante dans les communications et les loisirs. Ils sont même devenus un outil majeur de la création artistique. L'ordinateur est devenu l'outil et le symbole du développement des pays industrialisés. Leur nombre dépassera bientôt celui des habitants de la planète. A n'en pas douter, la place de l'informatique dans nos sociétés ne cessera de croître dans le futur et, avec elle, l'ensemble des besoins qui y sont liés.

Au cours des cinq dernières, années, la puissance de calcul des micro-ordinateurs s'est accrue d'un bon facteur dix. Les capacités en mémoire dont disposent ces micro-ordinateurs ont suivi la même progression. Les réseaux qui permettent la communication entre les ordinateurs on vu leur débits multipliés par cent.

Aujourd'hui, nous voyons apparaître deux grandes directions dans le domaine des architectures informatiques. L'une vise au développement de circuits très complexes destinés à des applications spécifiques « System On a Chip » ou il s'agit de regrouper dans un même circuit un ensemble d'opérateurs, processeur de contrôle et co-processeurs spécialisés pour réaliser une unique fonction. Ces puces sont destinées à être enfouies dans les objets de la vie quotidienne (téléphone ou voiture). Une autre tendance vise à tirer le meilleur parti des micro-ordinateurs disponibles en les reliant par un réseau performant pour en faire des machines parallèles qualifiées de « grappes de PC »[Grap]. Ces architectures sont destinées à des applications nouvelles nécessitant de grandes puissances de calcul ou de stockage (infographie, simulation, serveur de bases de données, etc…). Ces applications, qui

jusqu'alors, étaient réservées aux « main frames », peuvent maintenant être exécutées sur des « grappes de PC » [Cap98] [Mark96].

L'élément clé de ce type de machine est la gestion des communications entre les nœuds de calculs. Cette thèse porte sur l'architecture du réseau d'interconnexion d'une machine parallèle de type « grappes de PC ». Nous abordons trois aspects complémentaires :

1. Matériel : par la réalisation d'un contrôleur réseau réalisé comme un circuit VLSI, NOÉ[1].
2. Système : par la définition d'un micronoyau, BEE, destiné à gérer le parallélisme entre les différentes tâches du contrôleur réseau.
3. Protocole : en définissant un protocole de communication de base, RWU, essayant de tirer parti des performances matérielles du réseau.

A travers l'architecture interne du contrôleur réseau NOÉ, nous serons amenés à traiter des aspects spécifiques aux « System On a Chip », tel que le contrôle par un processeur embarqué de co-processeurs fonctionnant en parallèle.

1.1. Cadre de cette thèse

Le travail a été réalisé au sein de l'équipe ASIM[2] du laboratoire LIP6[3] dans le cadre du projet MPC[4] visant à la réalisation d'une machine parallèle de type « grappe de PC ». Le développement du circuit NOÉ a été supporté par le projet Européen ARCHES[Arches].

Une première génération de la machine MPC est aujourd'hui opérationnelle, elle est basée sur l'utilisation de micro-ordinateurs de type PC reliés à l'aide d'un réseau spécifique HSL[5] développé au LIP6[6] dans le cadre des projets européens HIC et EUROPRO[Euro].

[1] New Open-Ended network controler.

[2] Architecture des Systèmes Intégrés et Micro-électronique

[3] Laboratoire d'Informatique de Paris 6

[4] Multi-PC

[5] High Speed Link [HSL]

[6] Laboratoire d'Informatique de Paris 6.

1.2. Organisation du document

Le chapitre 2 présente les objectifs de notre travail. Nous présentons les machines parallèles de type « grappes de PC ». Nous précisons le cadre de notre étude, et nous présentons la problématique des communications entre nœuds de calculs d'une machine du type « grappe de PC ».

Le chapitre 3 dresse un *état de l'art* concernant les « grappes de PC ». Nous détaillerons les réseaux d'interconnexion de différentes machines parallèles ayant une problématique similaire.

Le chapitre 4 expose le protocole de communication RWU[7] comme une alternative aux protocoles présentés dans l'état de l'art. La définition de ce protocole va nous permettre de préciser ce qui est attendu d'un contrôleur réseau adapté aux « grappes de PC ».

Le chapitre 5 est consacré à la présentation de l'architecture du contrôleur réseau NOÉ.

Le chapitre 6 présente l'architecture du micro-logiciel embarqué sur le microcontrôleur associé à NOÉ. Ce chapitre met un accent particulier sur la présentation du micronoyau BEE, ce dernier servant de support aux tâches réalisant le protocole réseau.

Le chapitre 7 présente la plate-forme de simulation VNOÉ et les résultats expérimentaux obtenus.

Enfin, le dernier chapitre apporte une conclusion. Nous identifions les améliorations qu'il faudrait apporter à l'architecture NOÉ ainsi que les perspectives ouvertes par notre travail.

[7] Remote Write User mode.

CHAPITRE 2

Problématique

Dans ce chapitre, nous rappelons de façon générale les différentes architectures de machines parallèles. Nous détaillons les architectures de type « grappe de PC » en analysant les principaux éléments constitutifs de celles-ci. Nous définissons les points clefs qui nous serviront plus tard de critères de comparaison.

2.1. Les besoins

2.1.1. Les besoins de puissance de calcul

En première place, des applications nécessitant une grande puissance de calcul, nous retrouvons la simulation de systèmes physiques (météorologie, physique nucléaire, exploration pétrolière). D'une façon générale la plupart des applications scientifiques sont très gourmandes en puissance de calcul. D'autres domaines tels que la génération d'images virtuelles (en deux voire trois dimensions) apparaissent et nécessitent également de très grandes puissances de calcul. Un autre gros consommateur de puissance de calcul est la conception assistée par ordinateur. Dans le domaine de la conception micro-électronique, les besoins sont très importants, que se soit pour la simulation de systèmes complet ou le routage de circuits de plusieurs dizaines de millions de transistors.

Pour tous ces domaines nous constatons que la puissance de calcul des ordinateurs actuels est un facteur limitant.

2.1.2. Les besoins d'espace mémoire

Nombre d'applications précédemment citées manipulent de très gros volumes de données. Le calcul formel utilise des matrices de plusieurs milliards d'éléments. Une image à deux

dimensions comporte plusieurs millions de pixels. Toutes ces applications nécessitent de très grandes capacités de mémoire.

Aujourd'hui, un ordinateur conventionnel ne dispose que de quelques centaines de MegaOctets de mémoire et c'est insuffisant pour les applications précédemment citées.

2.1.3. Les besoins d'entrées-sorties

Avec l'utilisation de plus en plus importante du stockage des informations sous forme numérique, c'est la capacité des disques durs qui explose. Le développement d'Internet fait apparaître la nécessité de très gros serveurs de données pour permettre à un très grand nombre d'utilisateurs d'accéder simultanément à des bases de données. Le tri de ces informations nécessite une grande puissance de calcul, leur stockage nécessite de très grandes capacités de disque et leur transfert sollicite tout particulièrement les entrées-sorties conventionnelles ainsi que les réseaux d'interconnexions.

Là encore, ces applications sont limitées par les caractéristiques des ordinateurs conventionnels.

2.2. Les machines parallèles comme réponse à ces besoins

Bien que les caractéristiques des ordinateurs conventionnels soient en constante progression, les puissances de ces machines sont toujours des facteurs limitants pour les applications les plus exigeantes.

La réponse à ces besoins consiste en une parallélisation des architectures des ordinateurs classiques : on va remplacer l'unique processeur par un certain nombre de processeurs, en cela on va multiplier la puissance de calcul théorique par le nombre de processeurs. On pourra en faire de même pour la mémoire ainsi que pour les disques et cela dans l'objectif de répondre aux différents besoins énoncés précédemment.

L'analyse du monde qui nous entoure nous conforte dans cette approche, nos sociétés sont elles mêmes basées sur la répartition du travail entre les individus. Nous pouvons prendre l'exemple d'une usine appliquant les principes du Taylorisme : le travail est décomposé en un ensemble de tâches plus ou moins élémentaires qui sont confiées à différents ouvriers spécialisés.

Prenons l'exemple de la construction d'une voiture : si 20 jours sont nécessaires à un ouvrier pluridisciplinaire pour assembler une voiture (en supposant que cette assemblage puisse être décomposé en 10 sous-assemblages indépendants), en utilisant 10 ouvriers, une voiture pourra

être produite en 1,5 jours. Le premier jour, les 10 ouvriers spécialisés (donc plus efficaces) vont réaliser leurs sous-assemblages indépendamment. A la fin de la première journée, il pourront se synchroniser et utiliser la troisième demi-journée pour assembler entre eux les 10 éléments déjà produits. Cette collaboration permet de réduire le temps de production d'une voiture et la spécialisation des ouvriers permet d'accroître la productivité globale de l'usine.

A travers cet exemple, nous voyons déjà apparaître les écueils qui guettent l'ordinateur parallèle :

- la possibilité de découper la tâche en plusieurs sous-tâches indépendantes de même taille ;
- la localisation géographique des processeurs qui complique les transferts de données et la synchronisation entre les tâches (par analogie, les ouvriers travailleraient dans des usines différentes, ce qui compliquerait la synchronisation et qui obligerait le transport des matières premières et des sous-assemblages) ;

Du point de vu des modes de communication entre les processeurs, le matériel peut offrir au programmeur deux modèles de gestion de la mémoire.

2.3. Les machines à mémoire partagée

Les machines dites « à mémoire partagée » ne partagent pas nécessairement une mémoire centralisée mais toutes les cellules processeur partagent le même espace d'adressage [Li89]. Tous les échanges entre cellules processeurs passent par des lectures et des écritures mémoires.

Dès que le nombre de processeurs devient important, la mémoire ne peut être unique et centralisée. Les seules machines à mémoire partagées existantes sont des NUMA[8]. Ces machines nécessitent un mécanisme matériel spécifique qui permet, suivant les adresses, de diriger les écritures et les lectures, soit vers la mémoire associée à la cellule processeur initiatrice de la transaction, soit vers une autre mémoire. Suivant les cas (local ou distant) l'accès nécessitera un temps différent. Pour des raisons évidentes de performances, les cellules processeur doivent pouvoir bénéficier de mécanismes de cache afin, de diminuer les temps d'accès aux mémoires distantes. La gestion de la cohérence de ces caches doit être mise en œuvre par des éléments matériels spécifiques très complexes à réaliser.

[8] Non Uniform Memory Access

2.3.1. Les machines à passage de messages

Dans les machines dites « à passage de message », chaque cellule processeur dispose de son propre espace d'adressage. Pour réaliser une communication entre deux cellules processeurs, un programme devra faire appel à une bibliothèque spécialisée permettant l'envoi et la réception de messages. Cette bibliothèque doit fournir au minimum des fonctions permettant de commander le transfert de données entre deux nœuds.

Il existe de nombreuses bibliothèques de passages de messages telles que PVM et MPI [Méh98] qui sont plus particulièrement dédiées à des utilisations du type machines parallèles. Les « Active Messages » [Eick92] ont pour objectif de permettre à un programme s'exécutant sur une cellule processeur de déclencher l'exécution de routines spécifiques sur des cellules processeurs distantes.

2.4. Les « grappes de PC »

Comme en fait la constatation David Patterson [And95], il existe toujours un décalage entre la cellule processeur d'une machine parallèle et celle d'une station de travail. On peut facilement évaluer à deux ans le délai qui sépare l'apparition d'une cellule processeur dans une station de travail de son apparition dans une machine parallèle. Cette durée correspondant approximativement au doublement des performances des processeurs. Ce décalage est simplement imputable aux différences de volumes des marchés respectifs des stations de travail et des machines parallèles.

Les faibles volumes de production des machines parallèles expliquent également les coûts nettement supérieurs à ceux de n stations de travail de type PC pour une même puissance de calcul.

L'idée des « grappes de PC » découle simplement de ces constatations et vise à réaliser une machine parallèle en reliant entre elles des stations de travail de type PC. Pour bénéficier le plus rapidement des améliorations technologiques, il sera nécessaire de réutiliser le maximum de composants dits « standard ».

Nous allons étudier les différents composants tant matériels que logiciels d'une machine parallèle du type « grappe de PC ».

2.4.1. Le nœud de calculs

Toutes les cartes mère de PC présentent la même structure :

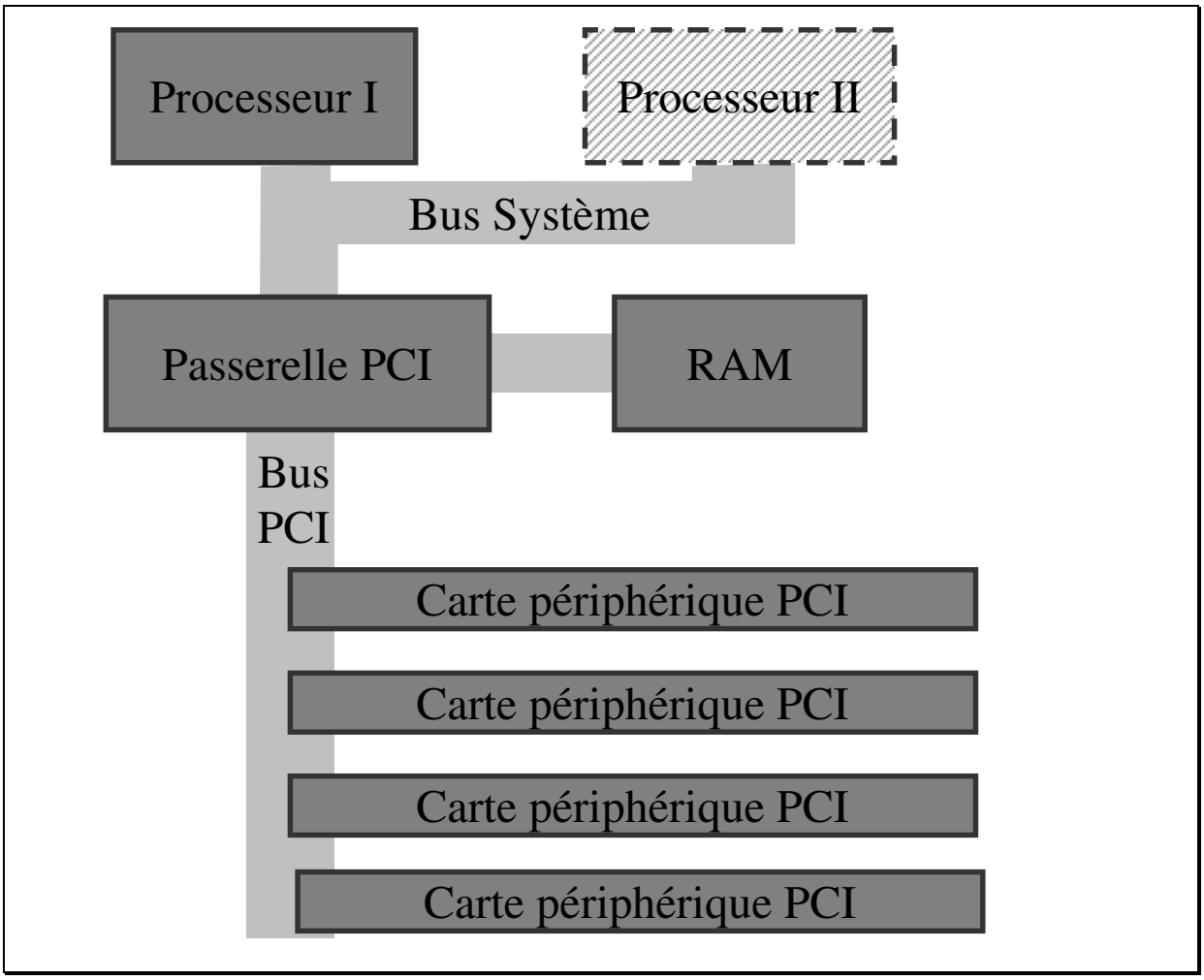


Figure 2.1 Schéma d'une carte mère.

Cette structure est déterminante dans le choix du type d'architecture qu'il est possible de réaliser avec une carte mère de station de travail de type PC. Cette carte est bâtie autour de deux bus distincts :

- le bus système : il permet de relier le ou les processeurs à la mémoire RAM, par l'intermédiaire de la passerelle PCI (avec un débit de 800 Mo/s) ;
- le bus PCI est un bus d'entrée-sortie (il est possible de connecter sur ce bus des cartes périphériques, telles que les cartes réseau avec un débit de 132 Mo/s).

Le rôle joué par la passerelle PCI est essentiel : elle va se comporter comme un aiguillage du point de vue des accès mémoires. Suivant l'adresse émise par le processeur, les requêtes vont être dirigées soit vers la mémoire RAM, soit vers un périphérique PCI. Cette aptitude à rediriger les requêtes mémoire vers un périphérique qui pourrait être un contrôleur réseau,

peut sembler intéressante pour mettre en œuvre des mécanismes de communications de type mémoire partagée. Malheureusement, comme nous l'avons vu précédemment, pour être performante ce type de machine doit utiliser la mémoire cache. Pour qu'un contrôleur réseau puisse assurer la cohérence de ce cache, il est nécessaire de pouvoir observer tous les échanges entre le ou les processeurs et ce cache. Une caractéristique de cette architecture est que le contrôleur réseau est connecté au bus PCI, ce qui ne lui permet pas d'observer les transactions entre la mémoire et les processeurs.

Sans modification de cette carte mère, il est donc impossible au contrôleur réseau d'assurer une cohérence entre les mémoires situées sur les différentes cartes mères [Cran00]. Il est donc difficile de fournir un modèle de programmation de type mémoire partagée, sur une « grappe de PC ».

2.4.2. Le système d'exploitation

Les raisons qui s'appliquent au choix d'une carte mère comme cellule processeur peuvent tout aussi bien s'appliquer au choix du système d'exploitation.

En choisissant un système d'exploitation standard de station de travail, nous bénéficions des évolutions de celui-ci (la compatibilité binaire est assurée, ce qui nous permet d'utiliser tous les logiciels existants et enfin, nous sommes assurés de bénéficier des pilotes disponibles pour les périphériques que nous souhaitons utiliser).

Un système d'exploitation comme un UNIX, dont les sources sont disponibles gratuitement, nous permet d'envisager les quelques modifications qui pourraient apparaître nécessaire pour tirer le meilleur parti du matériel, dans le cadre de son utilisation pour bâtir une machine parallèle.

Comme nous l'avons vu précédemment, les « grappes de PC » permettent seulement de réaliser des machines à passage de messages, et, les UNIX standard offrent déjà des bibliothèques permettant la communication par passage de messages.

2.4.3. Le réseau

Si nous reprenons l'analogie faite avec la production de voitures, nos ouvriers correspondants aux nœuds de calculs travaillent dans des usines différentes. Il va donc être nécessaire de mettre à leur disposition des moyens de transports performants : c'est le réseau.

Pour réaliser une machine parallèle basée sur des cartes mères de station de travail, il nous faut un réseau de communication. Ce réseau doit permettre aux cellules processeurs de communiquer entre elles avec des performances satisfaisantes.

L'objectif est d'équilibrer la capacité d'un processeur à produire des données, avec la capacité qu'un réseau peut avoir à les transporter.

Les deux caractéristiques importantes du réseau sont le débit et la latence.

Bien que la quantité de communication entre nœuds de calculs soit fortement dépendant du type d'application, nous pouvons évaluer un ordre de grandeur des débits réseau nécessaires.

En observant l'exécution d'un programme standard, nous constatons que 10 à 20% des instructions exécutées correspondent à des accès mémoire. Si nous considérons un processeur de type 32 bits fonctionnant a 1Ginstructions par seconde, la bande passante mémoire pour les données devrait être de 4*1000*0,2=800 Mo/s. En faisant l'hypothèse que seulement 25% des données produites ou consommées viennent ou sont à destinations d'autres nœuds de calculs, la bande passante réseau nécessaire à chaque nœud serait de l'ordre de 200 Mo/s.

Bien que ces estimations soient très approximatives, il apparaît que pour réaliser une machine parallèle, il est nécessaire de disposer d'un réseau permettant un débit très important (supérieur à 100 Mo/s).

Contrairement à la production de voitures, il ne nous est pas seulement nécessaire de produire beaucoup de voitures, mais il faut être capable de produire un résultat rapidement. Mieux vaut prévoir le temps de demain pour une région donnée que celui d'hier pour plusieurs régions.

La latence du réseau intervient directement dans la vitesse de production des résultats d'une application, dans le cas de dépendance immédiate de données : un nœud de calculs a besoin pour commencer son travail d'une donnée produite par un autre nœud. Dans ce cas, il ne peut y avoir recouvrement du calcul et des transferts de données et, donc, tout délai introduit par le réseau se traduira en temps perdu pour l'application.

Le réseau devra donc introduire la latence la plus faible possible.

En plus du débit et de la latence, la fiabilité du réseau est essentielle. De l'existence et du nombre des erreurs va dépendre la performance, voire même la possibilité d'utiliser la machine pour des applications parallèles dont la durée d'exécution dépasse fréquemment plusieurs heures.

2.4.4. Le contrôleur réseau

Le contrôleur réseau se trouve sur une carte périphérique, il est connecté d'un coté au bus PCI [PCI], de l'autre au réseau.

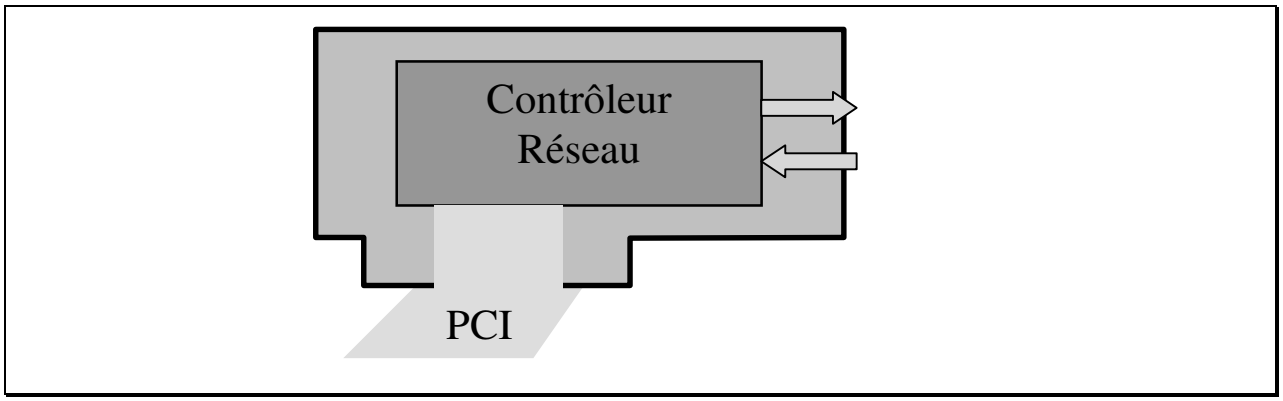


Figure 2.2 Carte réseau.

Les objets manipulés par le réseau sont des paquets. Ils sont constitués par les données à transférer et ils contiennent des informations concernant le transfert à effectuer. Ces informations, telles que le numéro du nœud destinataire ou l'adresse dans la mémoire du destinataire, sont généralement contenues dans l'en-tête du paquet.

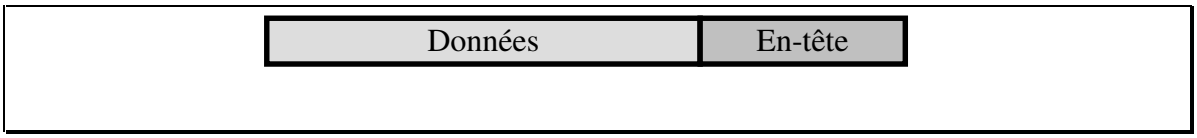


Figure 2.3 Paquet standard.

Le contrôleur réseau va recevoir les ordres venant du processeur. Ces ordres peuvent prendre la forme d'écritures aussi bien dans l'espace des entrées-sorties (IO), que dans l'espace mémoire.

Il va assurer le transfert des données à destination ou venant du réseau. Il faut donc qu'il soit capable de lire et d'écrire dans la mémoire du nœud de calculs. Il peut éventuellement assurer la mise en forme et l'extraction des paquets. En émission, il réalisera la fabrication des entêtes, en réception, il extraira les données des paquets reçus.

Lorsqu'un paquet aura été reçu ou qu'il aura été transmis, il pourra assurer la notification au processeur sous la forme d'une interruption, ou seulement par la mise à jours de liste de paquets reçus ou transmis.

Les contrôleurs réseau peuvent prendre différentes formes et être plus ou moins complexes. Leur efficacité dans le traitement des paquets a une incidence directe sur la latence du réseau : ils sont fortement liés au réseau qui leur est connecté, mais ils peuvent également être dédiés à un protocole de communication spécifique.

2.4.5. Le protocole réseau

Le protocole réseau englobe plusieurs éléments. D'une façon générale, il réalise l'interface entre l'application et le réseau.

Du point de vue de l'application, il définit un ensemble de primitives de communications. Nous pouvons distinguer trois principaux types de primitives :

- les primitives de transfert de données : elles réalisent la recopie d'une certaine zone de la mémoire du nœud local, vers (ou en provenance) une certaine zone de mémoire d'un nœud distant dont on a spécifié le numéro ;
- les primitives de synchronisation : elles permettent de mettre en place des barrières pour les différentes tâches s'exécutant sur des nœuds différents de la machine. Il s'agit de points de rendez-vous pour une application. Les différents ouvriers qui réalisent les différents éléments d'une voiture se fixent un rendez-vous et s'attendent mutuellement avant d'assembler l'ensemble de la voiture ;
- les primitives déclenchant l'exécution d'une tâche sur un nœud distant.

Toutes ces primitives peuvent être bloquantes ou non bloquantes du point de vue des applications. Une primitive bloquante interrompt l'exécution de l'application jusqu'à ce qu'elle soit terminée. Une primitive non bloquante n'interrompt pas l'exécution de l'application.

Un protocole définit également souvent un mode de notification. La terminaison d'une primitive non bloquante peut être signalée par une interruption matérielle, ou bien la mise à jour d'une liste particulière.

L'accès à ces primitives peut prendre la forme soit d'appels au système d'exploitation, soit d'appel à des fonctions appartenant à une bibliothèque de communication spécifique, telle que PVM ou MPI. Ces bibliothèques peuvent elles-mêmes faire appel au système d'exploitation ou commander directement le contrôleur réseau.

Le protocole dit de « bas niveau » définit le ou les différents formats de paquets. Il est généralement mis en œuvre par le contrôleur réseau.

La mise en œuvre d'un protocole de communication sollicite nécessairement deux ressources : le processeur du nœud de calculs et le contrôleur réseau. Il s'agit d'un point critique des architectures des machines parallèles. En effet, les traitements effectués par le processeur de calculs seront doublement coûteux, ils interviendront directement dans la latence de la communication, et le temps passé à mettre en œuvre le protocole sera autant de temps perdu pour les calculs.

2.5. Les Objectifs

Notre travail porte sur la conception d'un contrôleur réseau adapté aux machines parallèles du type "« grappe de PC » et il sera guidé par deux objectifs.

2.5.1. Les performances

L'objectif est de concevoir et de réaliser un contrôleur réseau capable de mettre en œuvre un protocole de communication qui permette à une application de tirer le meilleur parti d'un réseau de communication à haut débit (supérieur à 100 Mo/s) et faible latence (inférieure à 10 µs).

Comme nous le montrera l'étude des machines existantes, il s'agira principalement de minimiser la latence globale, logicielle et matérielle, et de rendre négligeable le surcoût pour le processeur de calculs.

2.5.2. Une plate-forme d'expérimentation

Les protocoles de communications et leur mise en œuvre sont aujourd'hui les principaux obstacles à l'utilisation performante des « grappes de PC ». Pour tester les différentes réponses possibles à ce problème, le contrôleur de réseau NOÉ devra être programmable afin de pouvoir être utilisé comme une plate-forme d'expérimentation pour différents types de protocoles réseau.

2.6. Les critères d'appréciation des « grappes de PC »

Nous pouvons définir un ensemble de critères nous permettant d'apprécier dans quelle mesure les différentes machines de type « grappe de PC » répondent à nos objectifs.

2.6.1. Le débit

Il ne s'agit pas du débit du réseau mais du débit de l'ensemble des éléments qui interviennent dans les transferts de données.

Le débit des communications doit pouvoir être comparable aux capacités à produire des données des applications s'exécutant sur la machine.

2.6.2. La latence

La latence est définie comme le temps écoulé entre le début d'une émission et la fin de la réception correspondante.

Ce qui nous intéresse n'est pas uniquement la latence du réseau, mais la latence vue par une application qui est la seule grandeur réellement pertinente pour mesurer la performance d'une machine parallèle.

Minimiser l'impact de cette latence sur la performance de la machine suppose un modèle de programmation spécifique, consistant à anticiper les transmissions, afin de permettre le recouvrement du temps de transfert des données par des calculs. Cette méthode est très difficile à mettre en œuvre voir impossible pour certaines applications.

Suivant le taux de dépendances mutuelles des différentes actions réalisées par un programme, il est plus ou moins aisé de découper un programme en plusieurs tâches pouvant s'exécuter en parallèle. Le niveau de ce découpage définit le grain de parallélisme d'une application. Dans le cas des machines à passage de messages une application est découpée en plusieurs processus ou thread, il s'agit d'un parallélisme dit à gros grain. Plus le grain de parallélisme est important moins fréquentes seront les communications.

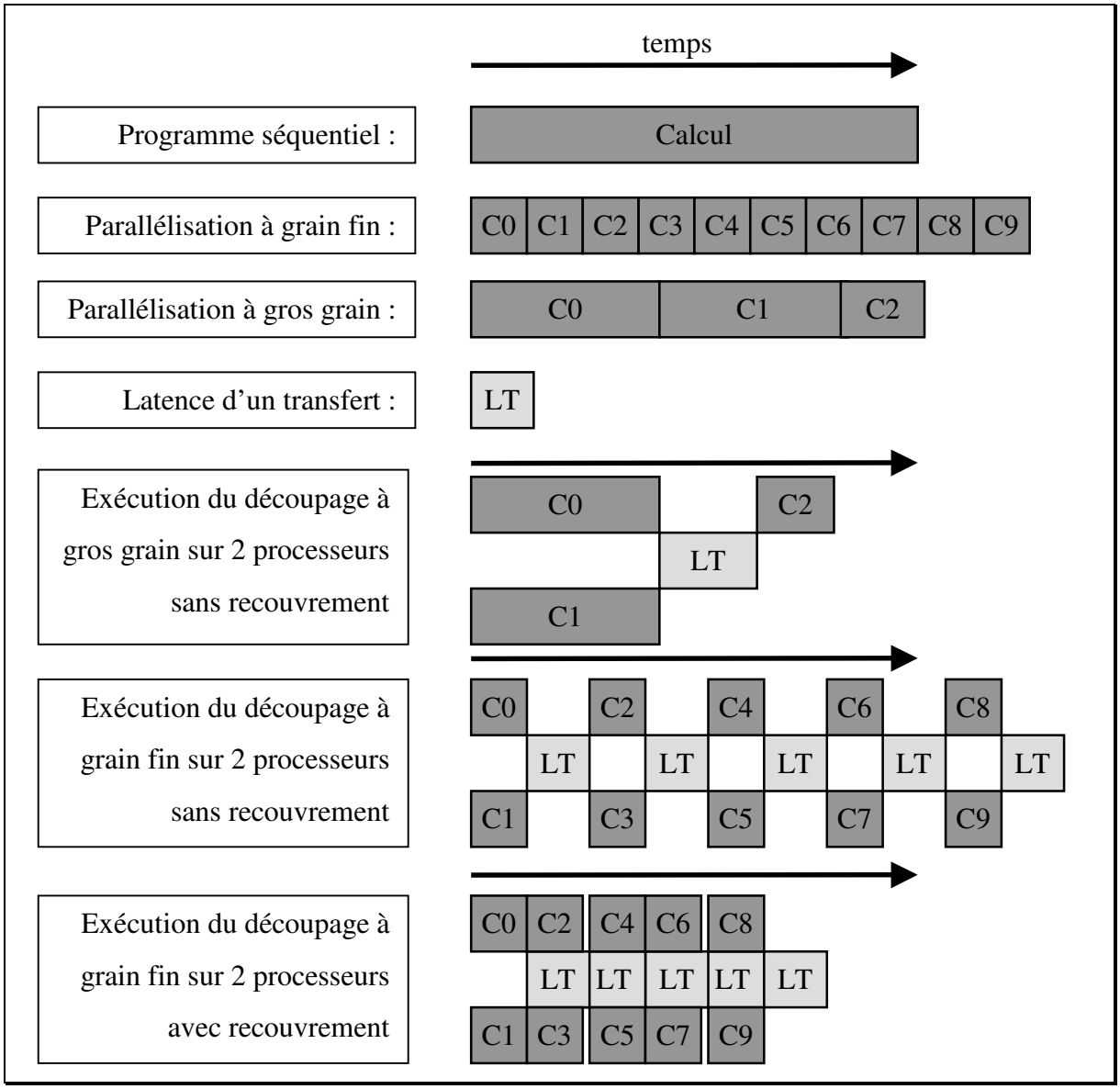


Figure 2.4 Découpage en tâches parallèles.

Les deux premiers exemples du schéma précédent montrent bien que plus le grain de l'application parallèle est important, moins fréquentes sont les communications et donc moindre est leur incidence sur les performances de l'application. Ces applications sont donc moins exigeantes en termes de latence de communication.

Le troisième exemple montre que pour une application à grain fin, les communications sont plus fréquentes et donc l'incidence de la latence est plus importante.

Le dernier exemple montre qu'il est possible de réduire l'incidence de la latence en permettant le recouvrement temporel des calculs et des transferts de données.

Ce recouvrement constitue une contrainte pour le programmeur ou éventuellement le compilateur et, suivant les dépendances contenues dans l'application, n'est pas toujours possible.

Il convient donc d'une façon générale de minimiser la latence des communications.

2.6.3. Le surcoût pour le nœud de calculs

Le troisième exemple montre qu'il est nécessaire de différencier deux sources de latence :

- la latence due au contrôleur réseau, donc l'incidence sur les performances de la machine, peut être limitée par le recouvrement temporel des calculs et des communications ;
- la latence due aux traitements effectués par le processeur du nœud de calculs et pour lequel par définition, il ne peut y avoir recouvrement.

Pour obtenir de bonnes performances, il est indispensable de minimiser les traitements liés aux communications effectués par le processeur du nœud de calculs.

2.6.4. La fiabilité

Si le réseau n'est pas fiable, un émetteur devra toujours s'assurer que les données ont été bien reçues avant de s'autoriser à les modifier. Pour éviter cela, il sera souvent nécessaire d'effectuer une copie des données avant de les transmettre, d'où une augmentation de la latence et des surcoûts pour le nœud de calculs.

2.6.5. L'extensibilité

Une machine parallèle doit pouvoir comporter un nombre important de nœuds de calculs. Le réseau doit pouvoir supporter plusieurs centaines de nœuds sans que ses performances en soient affectées.

2.6.6. La programmabilité

Comme nous voulons utiliser notre machine parallèle pour effectuer des expérimentations portant sur les protocoles de communication, celui mis en œuvre par le contrôleur réseau devra pouvoir être modifié par un utilisateur averti.

L'existence d'un environnement de développement et de mise au point constituera un gain appréciable.

CHAPITRE 3

Etat de l'art

Dans ce chapitre, nous allons présenter trois exemples de machines parallèles du type « grappe de PC ». Le premier est la machine « The Hive » [HIV97] de la NASA. Cette machine est exclusivement construite en utilisant des composants standard tant matériels que logiciels. La seconde machine présentée est celle du LHPC de Lyon [Herb98]. Elle utilise un réseau Myrinet [Myr] et un protocole de communication BIP [Prylli] développé spécifiquement pour les applications parallèles. Le troisième exemple MPC0 [Grei98] est un « cluster de PC » basé sur un réseau spécifique HSL et un protocole de communication spécifique DDSLR [Potter].

3.1. THE HIVE

The HIVE est une machine parallèle de type « grappe de PC » développée par la NASA. Cette machine utilise exclusivement des composants standards, tant matériels que logiciels. L'objectif de cette approche fut d'obtenir une machine en un minimum de temps. En quelques semaines, la machine a été opérationnelle et la principale difficulté rencontrée fut l'approvisionnement en armoire spécifique, permettant l'alimentation et le refroidissement des nœuds de calculs.

Il a existé deux versions de cette machine, la première réalisée en 1997 et, la seconde, consistant en un enrichissement de la précédente, est en cours de réalisation.

3.1.1.1. Architecture de la machine

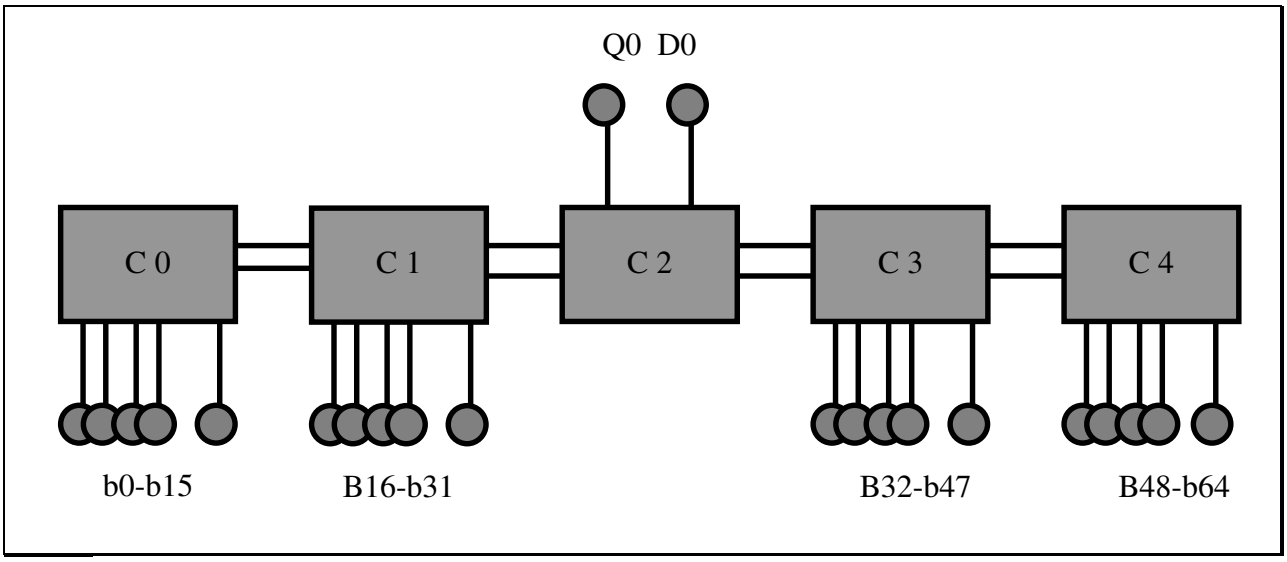


Figure 3.1 The HIVE 97

Les nœuds Q0 (The Queen) et D0 (The Drone) supervisent le fonctionnement de la machine et servent de console.

L'ensemble de la machine est refroidie par 262 ventilateurs dont le bruit est à l'origine du nom de la machine « the HIVE » (la ruche).

3.1.1.2. Les nœuds de calculs (b0 à b63)

The Hive est constituée de 64 nœuds de calculs, chaque nœud est constitué de :

- 2 processeurs pentium Pro (200Mhz, 256K de cache) ;
- 64 Moctets de RAM ;
- 2.5 Goctets d'espace disque.

Il s'agit de cartes mères standards de PC placées dans des boîtiers conçus pour être insérés dans des « racks ».

3.1.1.3. Le réseau de communication

Chaque nœud de calculs dispose d'une carte fast Ethernet standard 100 Mbits/s. Les nœuds de calculs sont reliés par 5 commutateurs Fast ethernet Full-duplex (C0-C4).

La bande passante disponible entre deux commutateurs est de 400 Mbits/s.

3.1.1.4. Le système d'exploitation

La distribution REDHAT de LINUX est utilisée sur tous les nœuds de calculs ainsi que sur les consoles. Aucune adaptation spécifique n'a été réalisée. Le contrôle des périphériques réseau est assuré par les pilotes standards.

3.1.1.5. Les protocoles de communication

THE HIVE offre aux utilisateurs trois bibliothèques de communication par passage de message : PVM, MPI, BSP. Ces bibliothèques utilisent les couches de communication TCP et UDP offertes en standard par les systèmes d'exploitation UNIX.

3.1.2. Les performances

3.1.2.1. Les débits

Les débits sont mesurés en considérant *nproc* processus communicants, *nproc*/2 processus réalisant des émissions, et *nproc*/2 processus réalisant des réceptions. Les mesures sont effectuées pour des tailles de paquets variant entre 2 octets et 4 Moctets.

Les débits sont mesurés pour les deux types de communications suivants :

- 16 processus placés sur 16 nœuds connectés au même commutateur ;

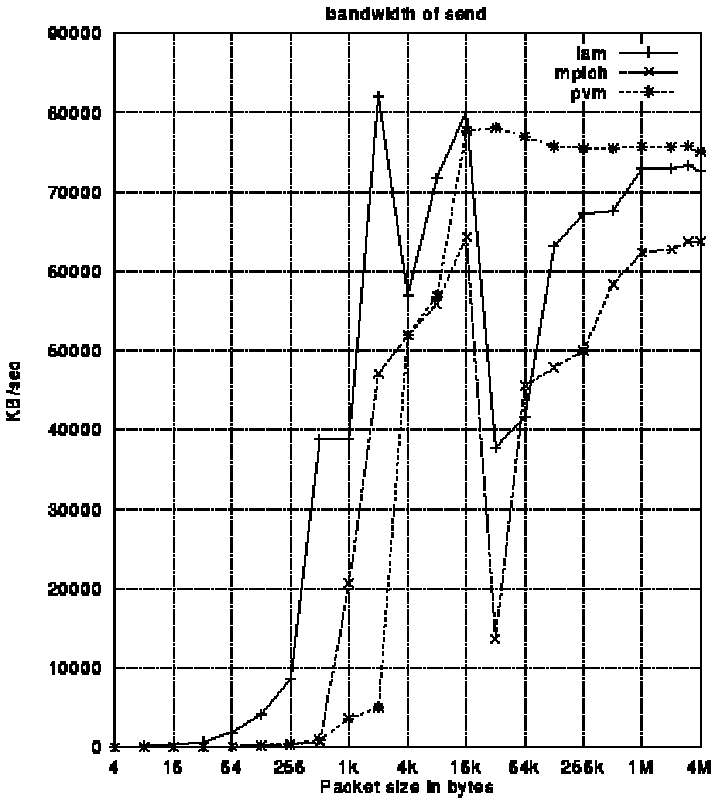


Figure 3.2 Débit switch

Les débits semblent atteindre leur asymptote pour des paquets de 4 Moctets. La valeur mesurée est alors comprise entre 70 et 80 Moctets/s, soit un débit de 70/8=8.75 Moctets/s pour chaque nœud émetteur.

Nous pouvons également observer que le demi-débit est atteint pour des paquets de taille comprise entre 1.5 et 4 koctets.

- Débits pour 64 processus s'exécutant sur 64 nœuds de calculs différents utilisant donc des commutateurs différents.

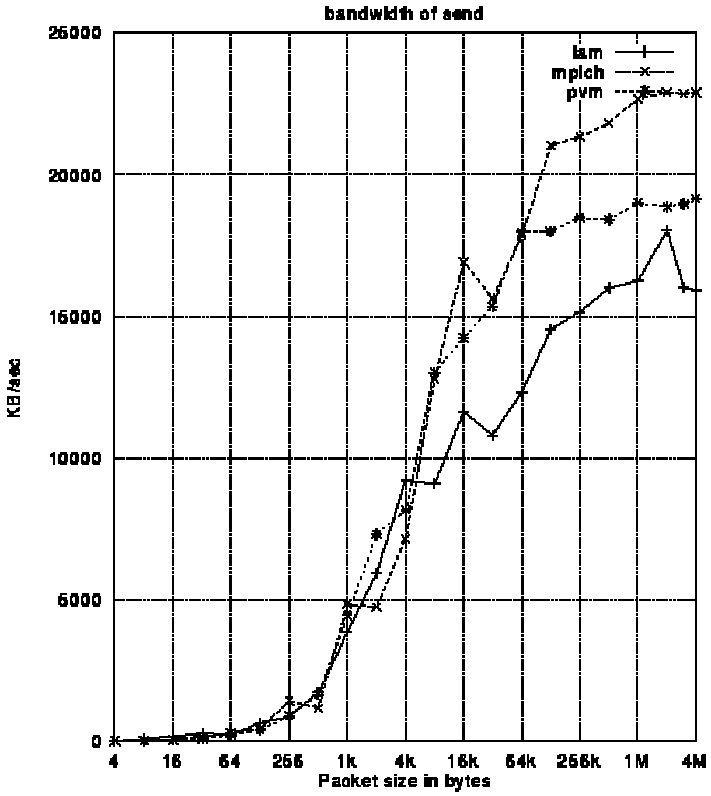


Figure 3.3 Débit inter-switch.

Dans le cas communications inter-commutateurs, les débits maximaux sont plus faibles, de l'ordre de 20 Moctets/s pour 64 nœuds, ce qui correspond à un débit de 600 Koctets/s et par nœud.

Le demi-débit est dans ce cas atteind pour des paquets dont la taille est comprise entre 4 et 16 koctets.

3.1.2.2. La latence

La latence est mesurée en répétant n fois un aller et retour, comme le montre le pseudo code suivant :

Nœud maître :

```
Répéter pour tous les messages de taille B variant de 2 octets à 4 Moctets
        Démarrer chrono
        Répéter N fois
                Envoyer (B octets)
                Recevoir (B octets)
        Arrêter chrono.
```

Nœud esclave :

```
Répéter pour tous les messages de taille B variant de 2 octets à 4 Moctets
        Démarrer chrono
        Répéter N fois
                Recevoir (B octets)
                Envoyer (B octets)
Arrêter chrono.
```

Les latences sont mesurées pour trois types de communications :

- 2 nœuds connectés au même commutateur :

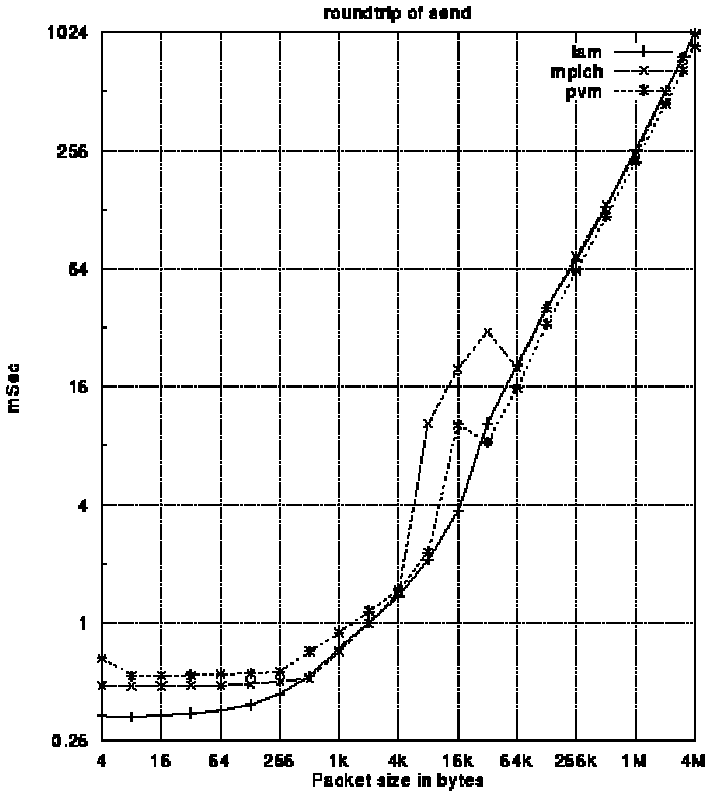


Figure 3.4 Latence switch.

- 2 nœuds connectés à des commutateurs différents :

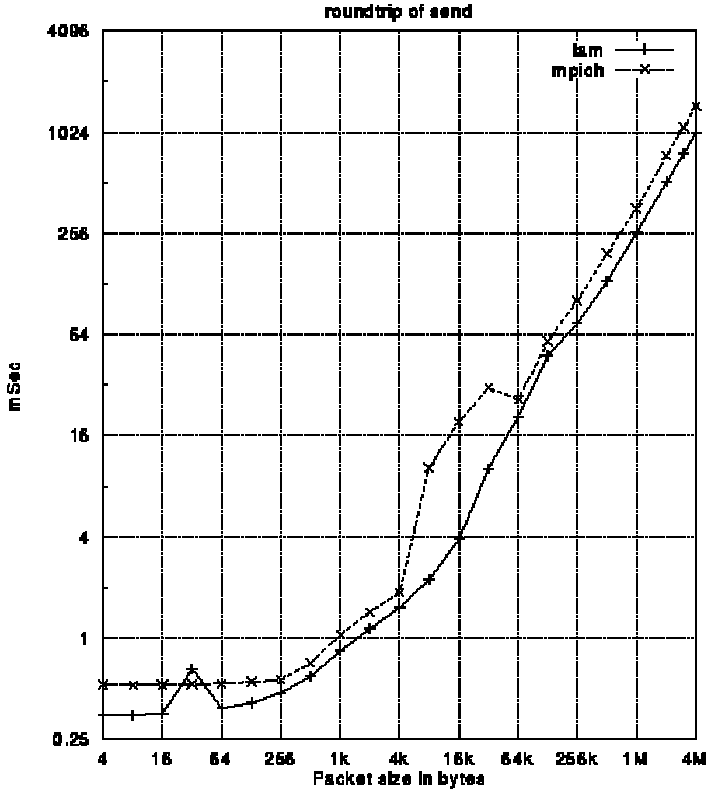


Figure 3.5 Légende inter-switch.

- Communication sur un unique nœud :

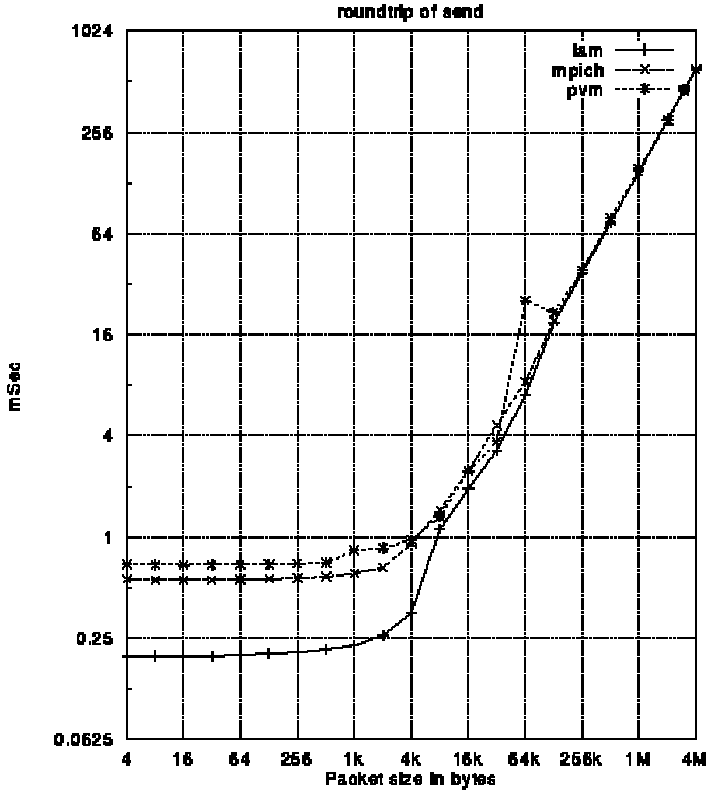


Figure 3.6 Latence nœud.

Si nous considérons les résultats pour PVM ou MPI, dans tous les cas et pour des paquets de taille minimale, un temps pour un aller et un retour légèrement supérieur à 500 µs a été mesuré. Ce qui correspond à une latence de 250 µs.

3.1.3. Conclusion

Les débits sont faibles. Pour remédier à ce défaut, the HIVE est en cours d'évolution : la communication entre les commutateurs est maintenant assurée par des liaisons ethernet à 1 Gbit/s. Les 64 nœuds d'origine sont maintenant connectés au même commutateur 72 ports. Deux commutateurs supportant respectivement 64 bi-pentium Pro et 16 quadri pentium III plus 10 bi-pentium III ont été ajoutés.

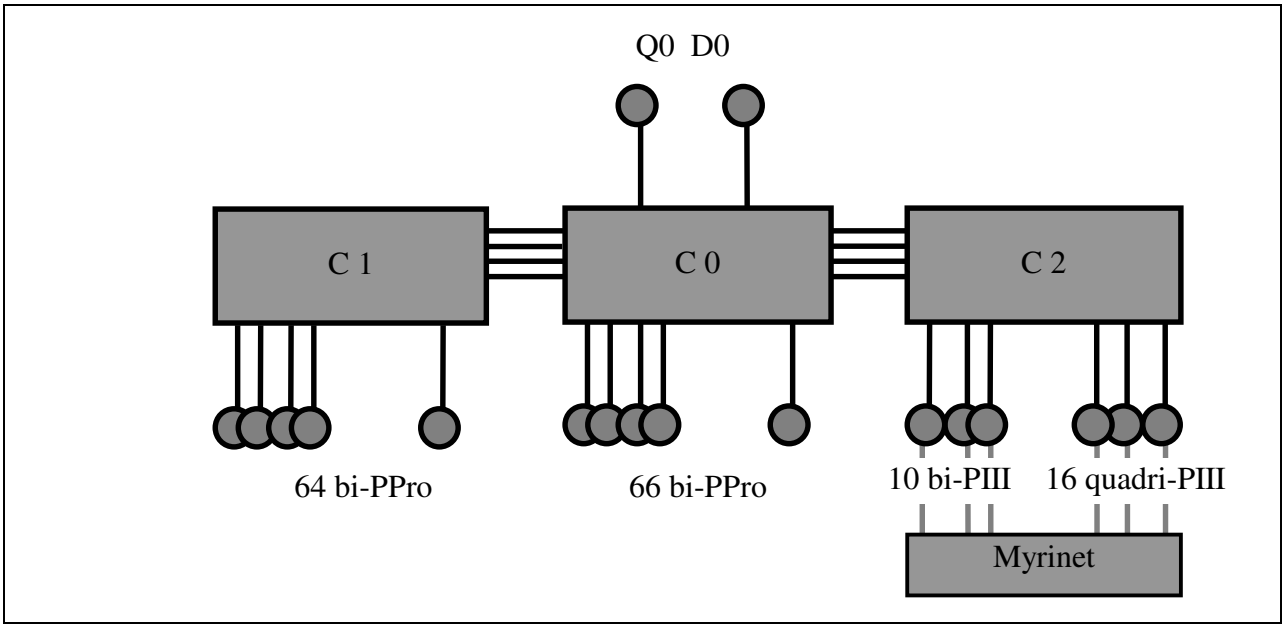


Figure 3.7 The HIVE 99

The HIVE est maintenant composée de 344 processeurs.

La latence égale à 250 µs ne permet pas d'exécuter des applications parallèles fortement couplées.

Les courbes montrant la latence pour des communications intra-nœud sont peu différentes des courbes mesurant la latence inter-noeuds, ce qui semble indiquer que la majeure partie de la latence mesurée est imputable aux traitements effectués par les processeurs de calculs. Cette latence est doublement coûteuse, car elle ne permet pas le recouvrement du temps de communication par du calcul.

La fiabilité est assurée par les couches de communication standards ce qui explique la latence élevée.

L'extensibilité est un des points faibles de cette architecture, les communications inter commutateurs faisant chuter dramatiquement la bande passante.

La programmabilité est restreinte aux couches de communication s'exécutant sur le processeur de calculs, car les cartes d'interface ethernet ont une architecture « fermée » (l'utilisateur n'a pas de possibilité de modifier leur comportement).

3.2. La machine BIP/Myrinet du LHPC

Le LHPC est un laboratoire de recherche sur les ordinateurs massivement parallèles et le calcul haute performance.

3.2.1. Architecture de la machine

La machine parallèle du LHPC est du type « grappe de PC », c'est une plate-forme d'expérimentation basée sur des stations de travail de type PC et DEC Alpha reliées par un réseau spécifique Myrinet.

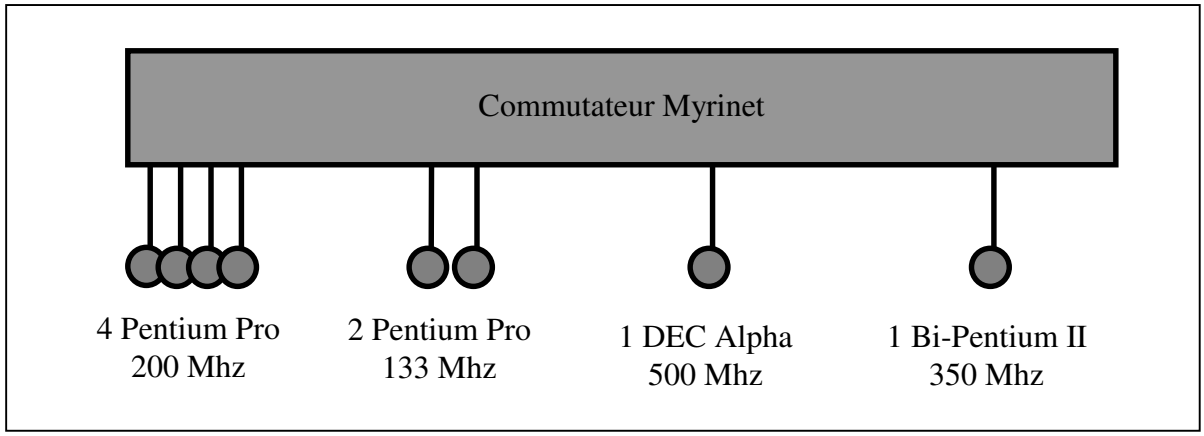


Figure 3.8 Architecture de la machine du LHPC

3.2.2. Le réseau de communication

Myrinet est un réseau spécialement conçu pour les applications du type machines parallèles. C'est le fruit de deux projets de recherche : Caltech Mosaic C Multicomputer [Seitz93] et USC/SI ATOMIC LAN [Feld94]. La technologie Myrinet est commercialisée par la société Myricom.

Les réseaux Myrinet sont composés de trois principaux éléments :

- un lien de communication de type parallèle 9 bits bidirectionnel et disposant d'un mécanisme de contrôle de flux ;
- des routeurs de paquets permettant de connecter de 4 à 16 liens ;
- un contrôleur réseau spécifique qui permet de relier un lien Myrinet à une station de travail par l'intermédiaire de son bus d'entrée/sortie PCI.

3.2.3. Le lien Myrinet

Le débit du lien Myrinet est de 1.28 Gbits/s dans chaque direction. Il s'agit d'une liaison parallèle 9 bits, (8 bits de données et un bit permettant de distinguer les caractères de contrôle).

Le contrôle de flux est assuré par des caractères spécifiques qui sont insérés dans le flux inverse. L'existence de ce contrôle de flux permet d'assurer qu'aucun paquet ne sera perdu. Il existe deux types de câbles :

Figure 3.9 Câble LAN

Figure 3.10 Câble SAN

Ces câbles existent pour des longueurs variant de 30cm à 3m pour les versions SAN, et de 1.5m à 10m pour les versions LAN. Ces longueurs sont un atout en termes d'extensibilité.

3.2.4. Le Routeur

Comme pour Ethernet la fonction de routages des réseaux Myrinet est centralisée et située dans des boîtiers externes. Le routage est explicite, les paquets contiennent la liste des ports de sortie qu'ils devront emprunter. Il existe des routeurs permettant d'interconnecter 4 à 16 nœuds.

Figure 3.11 Le routeur Myrinet

La machine du LHPC est équipée d'un routeur 8 ports.

3.2.5. Le contrôleur réseau

La carte Myrinet est bâtie autour de deux ASIC's et contient de 2 à 4 Moctets de mémoire.

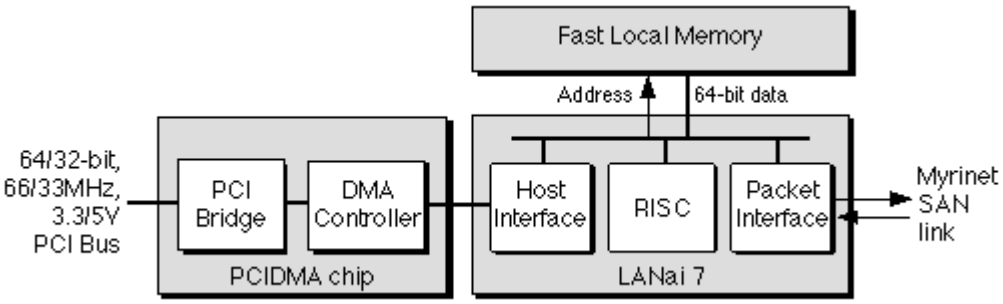


Figure 3.12 Le contôleur réseau Myrinet.

Cette architecture peut être décomposée en trois éléments majeurs : le contrôleur de DMA, l'interface Paquet et le processeur RISC [Lanai].

3.2.5.1. Le contrôleur de DMA

Les transferts de données correspondent à des accès en mode maître, sur le bus PCI pour le contrôleur de DMA. En émission, il lit directement les données dans la mémoire du nœud de calculs et dépose ces données dans la mémoire de la carte Myrinet. En réception, il recopie directement les données de la mémoire de la carte réseau, vers la mémoire de la station de travail. Dans ces deux cas, aucune intervention du processeur de la station de travail n'est nécessaire.

Ce contrôleur de DMA est compatible PCI 33/66 Mhz, 32/64 bits permettant des débits théoriques de 528 Moctets/s. Ces débits semblent suffisants pour supporter celui du lien Myrinet qui est de l'ordre de 2*100 Moctets/s.

Une autre caractéristique de ce contrôleur de DMA est de supporter toutes les combinaisons de longueurs et d'alignements des données à transférer.

La carte d'interface réseau utilisée par la machine du LHPC est une ancienne version de celle présentée ici, et elle supporte uniquement le bus PCI 32 bits fonctionnant à 33 Mhz. Ce qui limite le débit à 132 Moctets/s.

3.2.5.2. Le gestionnaire de paquet

Le gestionnaire de paquets a pour fonction de mettre en forme les paquets à émettre et de recevoir les paquets venant du réseau Myrinet. Il est vu par le processeur embarqué comme un ensemble de registres de commandes appartenant à son espace mémoire.

Du point de vue des émissions, le gestionnaire de paquets offre un certain nombre de registres 32 bits permettant sa commande :

- **SB** (Send Byte) permet l'envoi d'un octet ;
- **SH** (Send Half-Word) permet l'envoi d'un demi mot (16 bits) ;
- **SW** (Send Word) permet l'envoi d'un mot de 32 bits ;
- **ST** (Send Tail) marque la fin d'un paquet et envoi un caractère de CRC[9] ;
- **SML** (Send Memory Limit) initie un transfert entre la mémoire locale et le gestionnaire de paquets. Les données contenues dans la mémoire jusqu'à l'adresse indiquée comme limite seront directement envoyées sur le réseau Myrinet ;
- **SMP** (Send Memory Pointer) indique l'adresse dans la mémoire locale de la première donnée à transférer ;
- **SMLT** (Send Memory Limit + Tail) est identique à SML mais avec l'ajout d'un marqueur de fin de paquet et d'un caractère de CRC.

Avec cette méthode, l'en-tête du paquet est directement construit par les écritures venant du processeur embarqué. Le gestionnaire de paquets va se charger d'assurer le contrôle du DMA, avec la mémoire locale pour constituer le cœur du paquet. Une fois l'ordre donné, l'intervention du processeur embarqué ne sera plus sollicitée dans la construction du paquet.

[9] Cyclic-Redundancy-Check

La réception d'un paquet est symétrique à son émission. Elle se fait par une série de lectures et d'écritures dans des registres de commande du gestionnaire de paquets. Voici la liste des registres :

- **RB** (Receive Byte) la lecture de ce registre permet d'extraire un octet du réseau ;
- **RH** (Receive Half Word) la lecture de ce registre permet d'extraire un demi mot du réseau ;
- **RW** (Receive Word) la lecture de ce registre permet d'extraire un mot du réseau ;
- **RMP** et **RML (**Receive Memory Pointer) et (Receive Memory Limit), l'écriture dans ces registres permet de déclencher un DMA vers la mémoire locale.

Lors de la réception d'un paquet, la réaction du processeur embarqué doit être la plus rapide possible pour éviter d'engorger le réseau.

Ce contrôleur réseau a un comportement de type « store and forward » : lors d'une réception, les données ne transitent pas directement du réseau vers le bus PCI de la station de travail et, inversement, lors d'une transmission. Dans les deux cas, les données sont temporairement copiées dans la mémoire de la carte.

3.2.5.3. Le processeur RISC

Le processeur contenu à l'intérieur de l'interface LANAI est de type RISC. Le mot d'instruction est de 32 bits, et ce processeur possède un pipeline à quatre étages qui lui permet d'exécuter une instruction par cycle d'horloge. Il fonctionne à une fréquence d'horloge de 66 Mhz.

Une particularité intéressante de ce processeur est de supporter en matériel deux contextes d'exécution. Ainsi, tous les principaux registres sont dupliqués, pour permettre une commutation de contexte quasi immédiate, sans attendre que le pipeline soit vide.

Les deux contextes d'exécution sont différenciés : il y a un contexte dit « utilisateur » qui autorise les interruptions et un contexte dit « système » qui ne les autorise pas.

Le programme qui s'exécute sur ce processeur s'appelle le MCP[10]. Il est chargé à partir de la station de travail, au démarrage des pilotes Myrinet. Myrinet fournit les sources en langage d'assemblage de son MCP, ainsi qu'un compilateur croisé C/C++ adapté au jeu d'instruction du processeur LANAI. Un utilisateur averti a donc la possibilité d'écrire son propre programme de contrôle.

[10] Myrinet Control Program

Peu d'informations sont fournies concernant la structure du MCP et son utilisation des deux contextes d'exécution. Néanmoins, nous pouvons considérer que le contexte dit « système » est dédié au traitement des interruptions, et que le contexte dit « utilisateur » est attribué aux autres traitements.

Le fait de disposer de deux contextes d'exécution distincts permet au processeur du LANAI de gérer deux activés simultanées : par exemple, l'émission d'un paquet s'exécutant dans le contexte dit « utilisateur » et la réception d'un paquet s'effectuant dans le contexte dit « système ». La structure même du gestionnaire de paquets rend indispensable l'utilisation des interruptions. Le processeur embarqué doit au plus vite extraire et analyser l'en-tête d'un paquet, pour limiter la contention sur le réseau Myrinet.

Le LBUS[11] (qui relie le LANAI à la mémoire embarquée sur la carte) fonctionne à une fréquence double de celle du processeur, permettant ainsi simultanément le transfert des données entre la mémoire locale et le gestionnaire de paquets et les accès initiés par le processeur embarqué.

3.2.6. Le protocole réseau

Le protocole réseau est mis en œuvre par deux intervenants : le logiciel s'exécutant sur processeur de la station de travail, et le MCP s'exécutant sur le processeur embarqué dans le LANAI.

Myrinet fournit avec ses cartes un ensemble de composants logiciels MCP et Myrinet API qui offrent des primitives d'émission et de réception de messages. Ces primitives sont basées sur l'utilisation de canaux aussi bien en émission qu'en réception. A chaque canal sont associées trois files :

- une file d'émission qui contient les descripteurs des messages à transmettre ;
- une file de réception symétrique de celle d'émission qui indique à quel emplacement mémoire doivent être déposées les données reçues sur ce canal ;
- une file d'acquittement contenant la liste des messages reçus pour un canal.

Le canal de numéro 0 est réservé à la mise en œuvre du protocole TCP/IP, les autres sont mis à la disposition des applications. L'utilisation des notifications par interruptions est réservé au canal 0. Les applications utilisant l'API Myrinet doivent nécessairement utiliser des mécanismes de scrutation.

[11] Local Bus

Dans la file d'émission, chaque descripteur de message contient le numéro du nœud destinataire, ainsi qu'un pointer sur une liste de descripteurs des différentes zones mémoires constituant le message à transmettre. La description des buffers de réception est analogue à celle des buffers d'émission.

Ce protocole est complexe à mettre en œuvre, car il nécessite l'utilisation d'appels systèmes pour le partage de l'accès au réseau, ainsi que pour la déréférentiation des adresses virtuelles manipulées par les processus, vers des adresses physiques manipulables par le MCP. La complexité des descripteurs oblige le MCP à effectuer de nombreuses lectures dans la mémoire de la station de travail.

L'ensemble de ces opérations est coûteux en terme de performance, c'est pourquoi le LHPC a été amené à définir un protocole de communication simplifié BIP[12], pour être en mesure d'exploiter le performance du réseau.

3.2.6.1. Le Protocole BIP

L'objectif du protocole BIP [BIP] est de supprimer les appels systèmes qui sont très coûteux (quelques µs sur un processeur de type Pentium) et de limiter le nombre de recopies mémoire (avec l'apparition de réseaux performants, leur coût ne peut plus être négligé), et enfin de limiter au maximum les lectures sur le bus d'entrée/sortie de la station de travail.

Le protocole BIP est mono-utilisateur et ces primitives sont directement accessibles en mode utilisateur et ne nécessitent pas l'utilisation d'appels systèmes.

La déréférentiation des adresses virtuelles vers les adresses physiques est effectuée une fois pour toutes, et l'application devra réutiliser les tampons déjà utilisés pour de précédentes communications [Pryl97].

Le protocole BIP est basé sur l'utilisation d'une file d'émission et de plusieurs files de réceptions. Pour déclencher une émission, le processus écrit directement dans la mémoire de la carte Myrinet le descripteur du message à transmettre. Ce descripteur contient :

- le numéro du nœud destinataire ;
- un « tag » correspondant au numéro de la file de réception ;
- l'adresse où se trouvent les données dans la mémoire de l'émetteur (la zone contenant les données doit être contiguë en mémoire physique) ;
- la longueur des données à transmettre.

[12] Basic Interface for Parallelism.

Pour la mise en œuvre du protocole BIP, le LHPC a été amené à développer une version spécifique du MCP. Quelle que soit la longueur du message, la transmission prendra la forme d'un unique paquet sur le réseau. Afin de permettre un recouvrement entre les transferts, entre la mémoire de la station de travail et la mémoire de la carte Myrinet, ainsi que ceux intervenant entre la mémoire de la carte et le gestionnaire de paquet, la lecture des données dans la mémoire de la station de travail est réalisé par un ensemble de commandes de DMA distinctes.

Lors de la réception d'un message, le MCP lit, dans la file correspondant au « tag » transmis, l'adresse du buffer de réception correspondant, et effectue les commandes de DMA nécessaires à l'écriture des données en mémoire.

En réception, le MCP de BIP a un comportement différent suivant la taille des messages :

- pour les petits messages (de taille inférieure à environ 4 ko), les données sont écrites dans un unique buffer puis recopiées lors du « receive ». Cette stratégie évite la contention du réseau dans le cas ou le « receive » serait effectué après la réception des données ;
- pour les messages dits normaux, le paquet est bloqué dans le réseau, tant que le « receive » n'a pas été effectué.

Le protocole BIP peut directement être utilisé par les applications, ou il peut servir de base à la mise œuvre de protocoles plus complexes tel que PVM ou MPI [PTW99].

3.2.7. Les performances

Nous allons présenter les performances du réseau Myrinet, lors de l'utilisation directe par les applications du protocole BIP, et lors de l'utilisation des bibliothèques de passage de messages PVM et PMI.

3.2.7.1. Les débits

Utilisation directe de BIP :

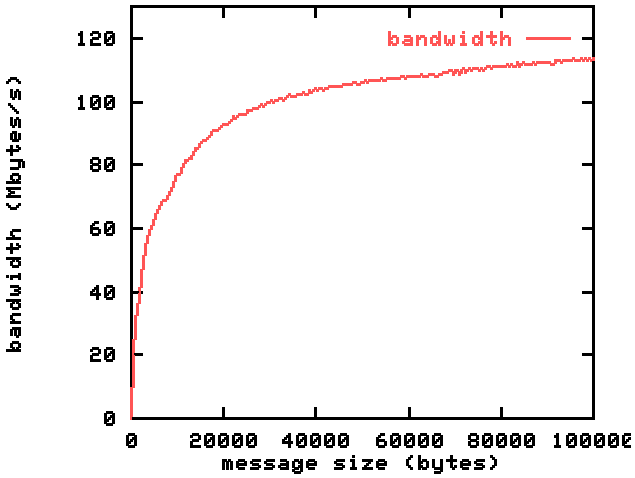


Figure 3.13 débit.

Nous observons que le débit est limité à 125 Mo/s ce qui est un très bon résultat. Cette limite vient du débit maximum théorique du bus PCI, qui est 132 Mo/s. Le demi-débit est obtenu pour une taille de message égale à 4Ko.

Le Débit, en utilisant une implémentation de MPI basé sur BIP, atteint 100 Mo/s pour des messages de 150 ko.

3.2.7.2. La latence

En utilisant BIP :

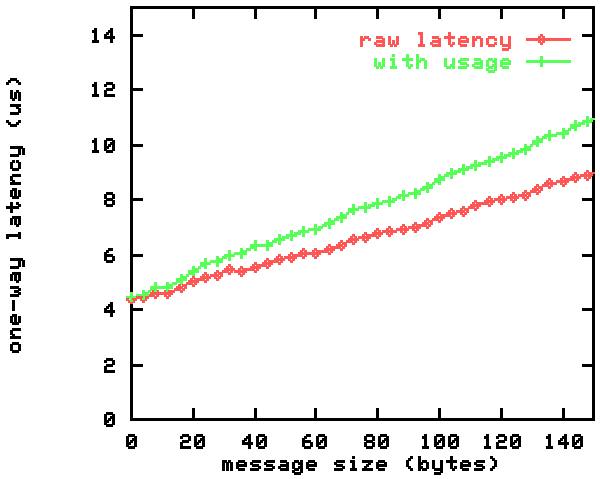


Figure 3.14 Latence sur BIP.

La « raw latency » correspond à la latence du « matériel » depuis l'envoi par le logiciel de l'ordre d'émission, jusqu'à l'écriture des données dans la mémoire du nœud destinataire. La seconde courbe prend en compte l'écriture des données à émettre dans un tampon d'émission

puis la lecture et la vérification des données reçues. Dans les deux cas, la latence est mesurée pour un « aller simple ».

Grâce à une implémentation sur BIP la bibliothèque MPI offre une très faible latence (9 µs [Prilly]).

3.2.8. Analyse

Les résultats sont dans l'ensemble excellents, mais BIP souffre dans son implémentation actuelle de quelques limitations :

- un unique processus utilisateur de BIP à la fois ;
- des tailles paquets trop importantes qui peuvent générer de la contention sur le réseau ;
- la complexité du MCP est peu propice à l'expérimentation de protocoles.

3.3. La machine MPC0

La machine MPC0 est également un machine parallèle du type « grappe de PC » conçue par le laboratoire LIP6. Il existe aujourd'hui cinq machines de ce type, dans différentes universités et écoles françaises.

3.3.1. Architecture de la machine

Cette machine est bâtie autour de deux composants spécifiques : un contrôleur d'interface réseau et un routeur à 8 ports.

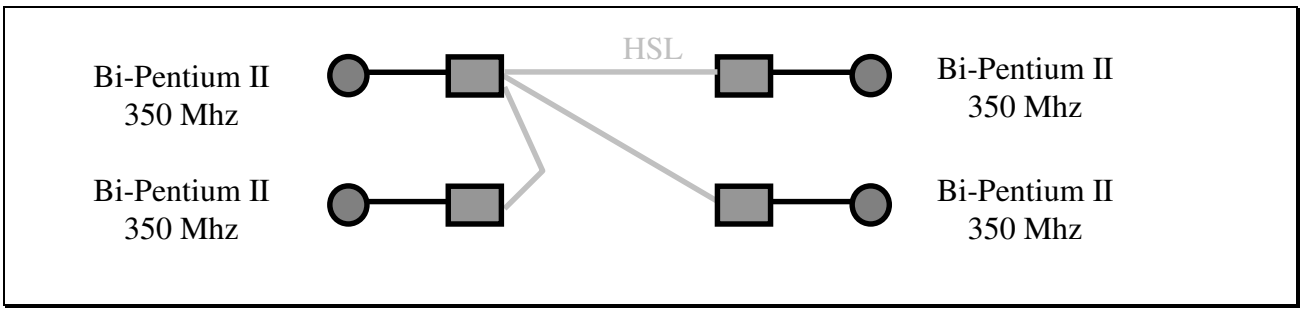


Figure 3.15 Architecture de la machine MPC0 du LIP6

Les nœuds de calculs de cette machine sont des cartes mères de stations de travail de type « PC », équipées chacune de 64 Mo de mémoire 2 Go de capacité disque.

Dans cette machine, la fonction de routage est décentralisée, et à chaque nœud de calculs est connectée une carte d'interface PCI nommée Fast HSL (contenant un contrôleur réseau PCIDDC et un routeur RCUBE [Reibaldi]).

3.3.2. Le réseau de communication

3.3.2.1. Le liens HSL

Le lien de communication HSL est de type série point-à-point à 1Gbit/seconde. Cette technologie est normalisée IEEE sous la référence IEEE 1355. Elle est le fruit d'une collaboration entre le laboratoire LIP6 et la société Bull.

Chaque lien est bidirectionnel et offre un débit de 1Gbit/s dans chaque direction. Les caractères transmis sur le lien sont codés sur 12 bits.

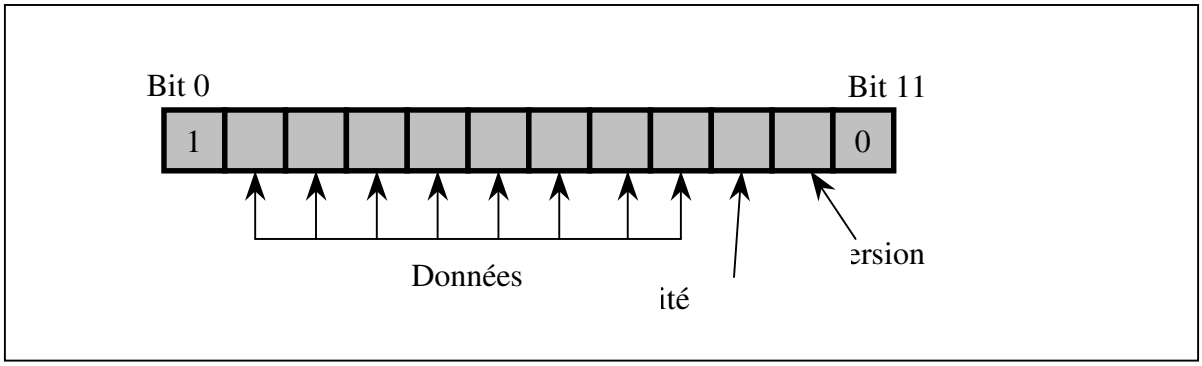


Figure 3.16 Codage 12 bit du caractère HSL

La séquence 0-1 séparant chaque caractère transmis permet la récupération de l'horloge, les caractères utiles transmis sont codés sur 9 bits : 8bits de donnée et le $9^{ième}$ bit servant à distinguer les caractères de contrôle.

Dans l'objectif de transmettre sur le lien autant de 0 que de 1, il est possible d'inverser tous les bits du caractère transmis, et cela en fonction des caractères précédemment transmis. Les caractères de contrôle on tous le même nombre de 0 que de 1. Le caractère 12 bits transmis sur le réseau ne contient pas de bit indiquant si il s'agit d'un caractère de donnée ou de contrôle, c'est pourquoi le bit d'inversion est utilisé pour transmettre cette information. Un caractère de donnée, lorsqu'il possède autant de 1 que de 0, est donc systématiquement inversé, alors qu'un caractère de contrôle ne l'est jamais.

Les câbles HSL peuvent atteindre 3m de long.

3.3.2.2. Le routeur RCUBE

Le laboratoire LIP6 a développé un routeur RCUBE adapté aux liens HSL. Ce routeur possède 8 entrées et 8 sorties totalement interconnectées. RCUBE permet de faire transiter un paquet

de n'importe quel lien d'entrée, vers n'importe quel lien de sortie. Les paquets ont la forme suivante :

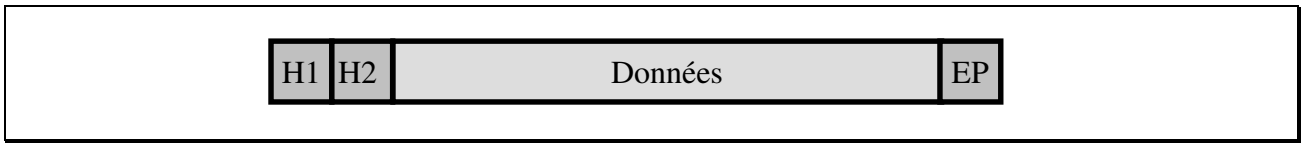


Figure 3.17 Format de paquet RCUBE.

Les paquets sont de taille quelconque. Il possèdent une en-tête H1 et H2 correspondant au numéro de leur nœud de destination, et leur fin est marqué par un caractère de contrôle spécifique EP[13].

Le routeur RCUBE est de type « Wormhole » [Zerr95] : les paquets ne sont pas stockés dans le routeur avant d'être réémis. La latence de ce routeur est très faible (150ns).

Pour éviter tout risque de perte de paquet, le réseau dispose d'un mécanisme de contrôle de flux fonctionnant sur la base d'un « flit » 32 caractères. Lorsqu'un récepteur est prêt à recevoir 32 caractères, il insère dans son flux sortant un caractère FCC[14] indiquant à l'émetteur qu'il peut lui envoyer un « flit ».

Pour chaque lien d'entrée dans RCUBE, l'espace d'adressage est découpé en 8 intervalles. A chaque intervalle correspond un ou plusieurs ports de sortie. Le lien de sortie qui sera effectivement emprunté par le paquet, sera sélectionné, suivant l'encombrement des liens et un mécanisme de priorité.

Ce réseau supporte le routage adaptatif. Il est donc possible de le configurer, de manière à ce que deux paquets originaires d'un même nœud puissent emprunter des chemins différents pour relier un même nœud destination.

3.3.3. Le contrôleur réseau

Le circuit PCIDDC[15] a été développé au LIP6. Il réalise l'interface entre le bus PCI et un port parallèle du routeur RCUBE.

[13] End of Packet.

[14] Flow Control Caracter.

[15] PCI Direct Deposit Component.

La primitive offerte par le contrôleur réseau PCIDDC est le « Remote Write »[16], consistant à recopier une certaine zone de la mémoire du nœud émetteur dans celle du nœud destinataire.

Le composant PCIDDC reçoit ses ordres directement du processeur de la station de travail, par des écritures dans des registres spécifiques, par l'intermédiaire d'accès en configuration sur le bus PCI.

Il réalise les accès en DMA, vers la mémoire de la station de travail pour lire les données. Il construit les paquets et les envoie sur le réseau. Quand l'émission est terminée, il peut assurer la notification au processeur de la station de travail, en lui adressant une interruption.

Le contrôleur d'interface PCIDDC assure lui-même le découpage des messages en paquets, suivant trois critères :

- une longueur maximale définie par l'utilisateur ;
- l'engorgement du réseau ;
- la disponibilité du bus PCI.

Les paquets transmis par PCIDDC sont conformes au format supporté par le routeur RCUBE ; leur en-tête est enrichi de quelques informations supplémentaires :

- un identificateur de message MI[17] permettant d'identifier les différents paquets d'un message ;
- l'adresse où les données doivent être déposées dans la mémoire du nœud destinataire ;
- un ensemble d'indicateurs permettant la notification.

PCIDDC renforce la sécurité des transmissions sur HSL (déjà couvertes par une parité caractère par l'ajout de caractères de CRC).

Lorsque PCIDDC reçoit un paquet du réseau, il en analyse l'en-tête, en extrait l'adresse de destination des données dans la mémoire du récepteur, et effectue les DMA nécessaires à l'écriture de celles-ci. Suivant les indicateurs qui lui ont été transmis, il peut assurer la notification au processeur destinataire par une interruption, pour lui indiquer que le message a été reçu.

L'architecture interne du composant PCIDDC correspond à un découpage des chemins de données d'émission et de réception :

[16] Ecriture Distante.

[17] Message Identifier

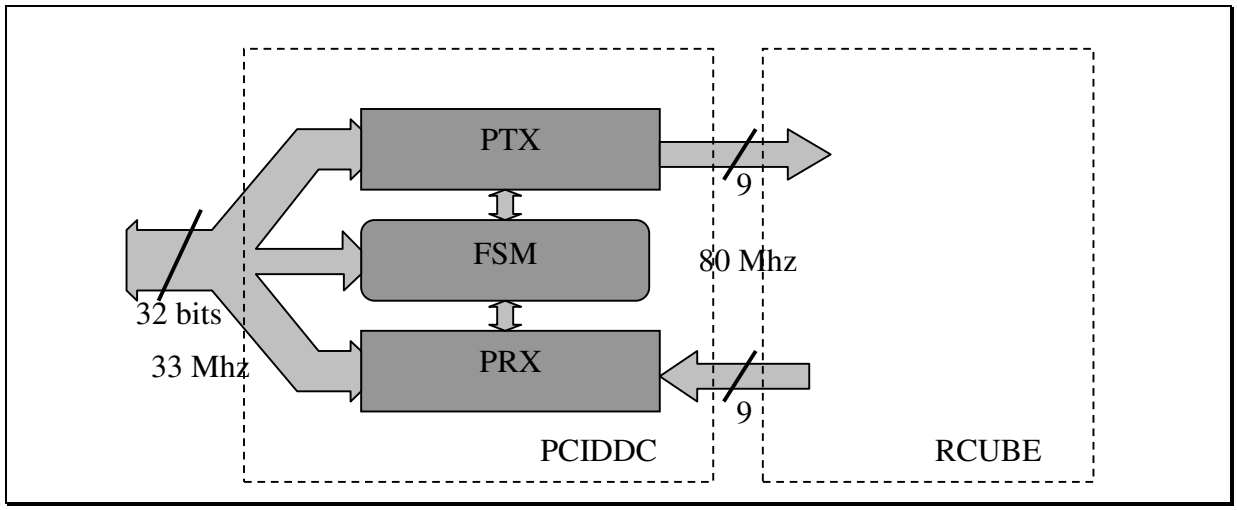


Figure 3.18Architecture du contôleur réseau PCIDDC

L'intégralité des traitements est supervisée par un unique automate FSM[18]. Cet automate participe à la mise en œuvre du protocole réseau DDSLR[19] qui est cablé dans le composant VLSI.

3.3.4. Le protocole réseau

Le protocole mis en œuvre par le composant PCIDDC est du type écriture distante : il est basé sur le principe que l'émetteur connaît l'adresse où doivent être écrites les données dans la mémoire du nœud destinataire, et c'est lui qui déclenche le transfert effectif. Un transfert peut être décomposé en 5 étapes :

[18] Finit State Machine

[19] Direct Deposit State-Less Receiver

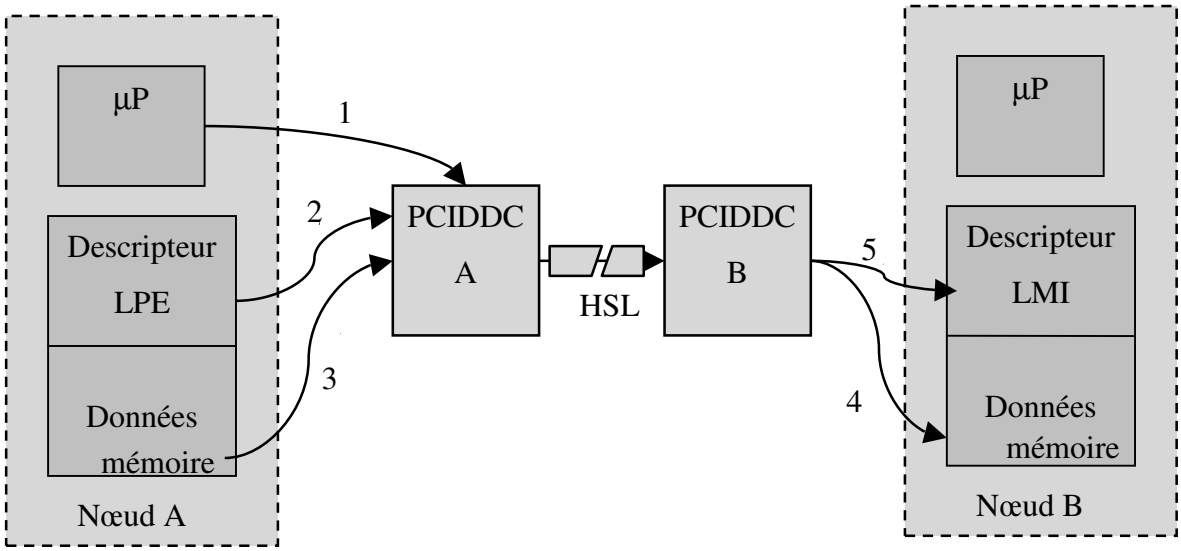


Figure 3.19 Synoptique d'une transmission DDSLR

1. écriture par le processeur de la station de travail dans le registre LPE_NEW de PCIDDC pour lui indiquer la présence d'un nouveau descripteur dans la LPE[20] ;
2. lecture du descripteur de message par PCIDDC ;
3. lecture des données par PCIDDC et envoi du message ;
4. écriture par le PCIDDC récepteur des données dans la mémoire du nœud destinataire ;
5. écriture dans la LMI[21] du descripteur correspondant au message reçu.

Ce protocole est très simple mais nécessite un rendez-vous préalable entre l'émetteur et le récepteur, pour indiquer à l'émetteur l'adresse de destination des données dans la mémoire du récepteur.

Il existe un seconde type de message dit « message court », pour lequel les données sont contenues dans les descripteurs, ce qui limite le nombre des transactions sur le bus PCI, et n'oblige pas l'émetteur à connaître l'adresse de destination des données dans la mémoire du destinataire.

3.3.5. Le logiciel de bas niveau

Deux interfaces logicielles permettent l'accès au composant PCIDDC :

[20] List of Pages to Emet.

[21] List of Message Identifier.

- la couche de communication PUT [Fenyö] qui est intégrée au noyau et qui nécessite l'utilisation d'appels système. Cette interface développée par le LIP6 fut la première disponible sert de base à l'implémentation de la bibliothèque de communication « à passage de message » MPI. Au dessus de cette couche PUT ont également été développées les couches SLRP et SLRV spécifiques qui permettent le passage de messages et utilisent la notion de canaux virtuels. Une version de la bibliothèque PVM a été développée sur la couche SLRV ;
- la couche de communication PAPI [Renault] développée au sein du laboratoire PRISM vise des objectifs différents. Cette interface de communication prend la forme d'une bibliothèque de fonctions directement accessibles en mode utilisateur. Papi est utilisé pour évaluer le « coût de la sécurité » dans les communications réseaux.

3.3.6. Les performances

Nous allons considérer les performances à trois niveaux : PUT, SLR et PVM [MPC].

3.3.6.1. Le débit

Au niveau PUT :

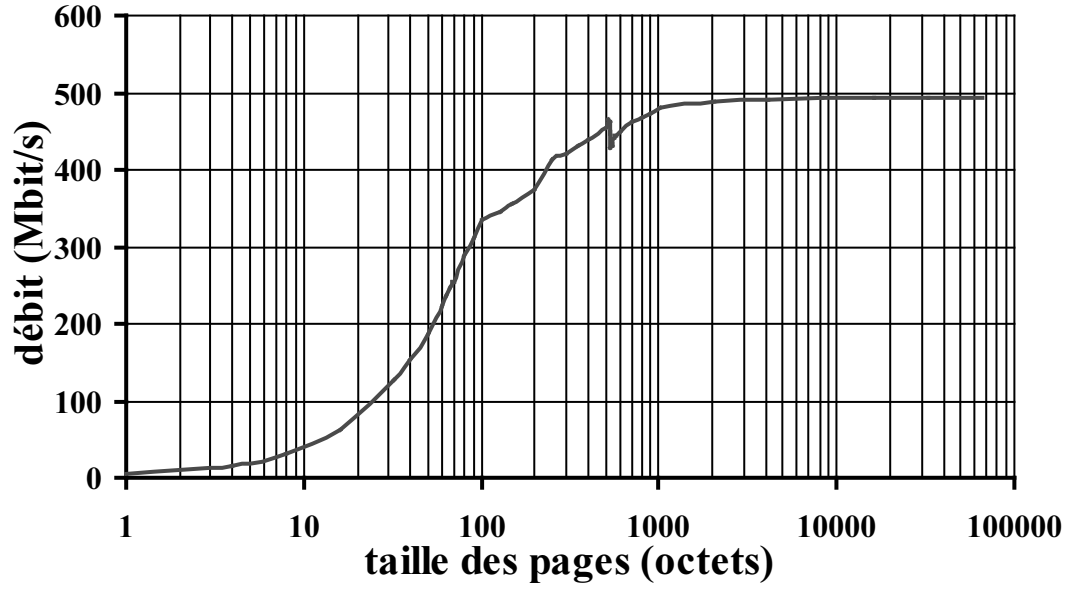


Figure 3.20 Débit MPC0 / PUT

Sur la carte servant aux mesures de performances, le réseau HSL fonctionne à 800 Mbits/s, ce qui correspond à un débit maximum théorique de 66 Mo/s. Le débit effectivement mesuré est

de 500 Mbits/s (62.5 Mo/s), soit 95 % de débit maximum théorique (c'est un bon résultat). Le demi-débit est obtenu pour des paquets de 60 octets, ce qui est un très bon résultat.

Aux niveaux SLR et PVM :

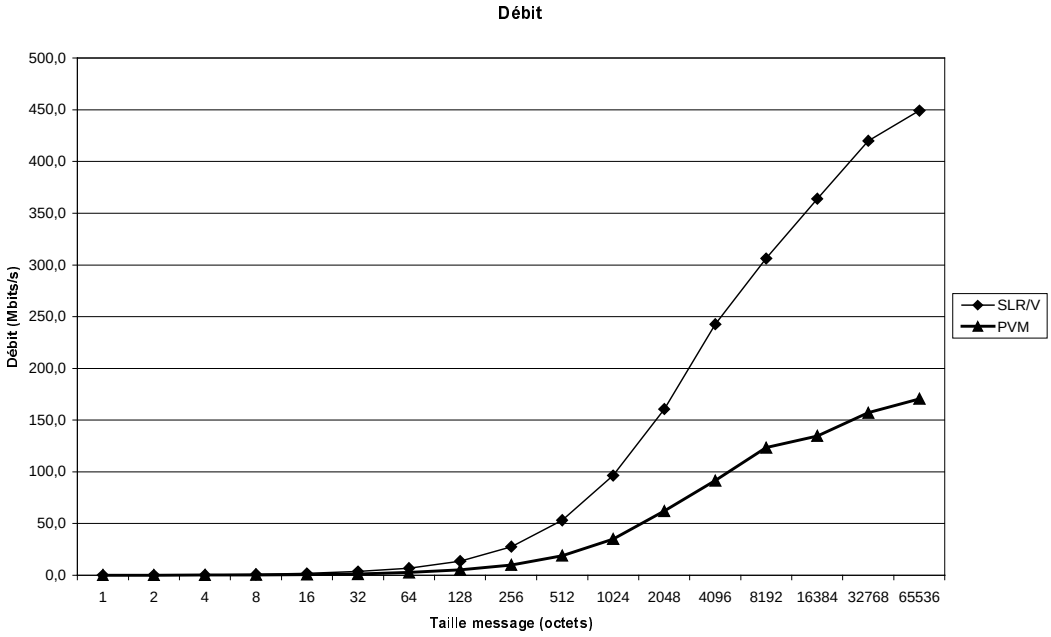


Figure 3.21 Débit MPC0 / PVM

Pour les communications au niveau PVM, le débit maximum atteint est de 170 Mbits/s, soit 30% du débit du réseau. Ce mauvais résultat est explicable par les trois recopies qui sont nécessaires au transfert d'un message. Le protocole SLR évite ces recopies, ce qui permet d'exploiter le plein débit du réseau.

3.3.6.2. La latence

La latence mesurée au niveau PUT est égale à 4µs, soit : 2µs pour les traitements matériels + 2µs pour les traitements logiciels sur le processeur de la station de travail, ce temps n'intègre pas les appels systèmes. Les résultats obtenus par l'interface PAPI sont très proches de ceux de PUT. Ils peuvent être considérés comme meilleurs par le fait que l'accès à PAPI ne nécessite pas l'utilisation d'appels systèmes.

Aux niveaux PVM et SLR :

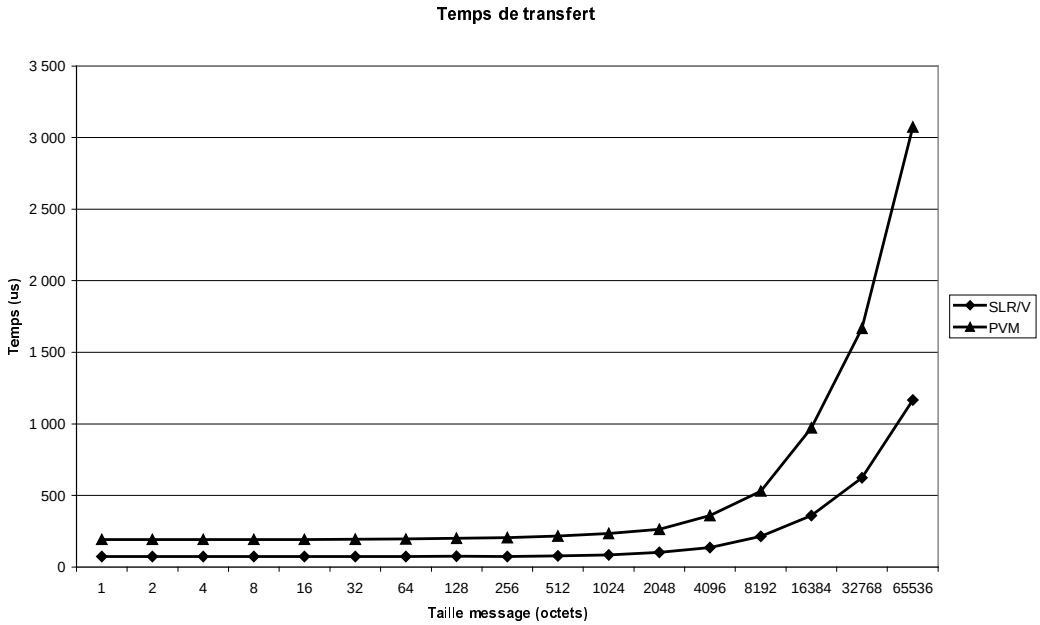


Figure 3.22 Latence MPC0 / PVM

Avec l'utilisation de PVM, la latence est très élevée (supérieure à 200 µs). Cette latence est intégralement due à des traitements logiciels effectués par le processeur de la station de travail.

3.3.7. Analyse

D'une façon générale les débits sont bons. Il convient donc de considérer plus précisément les résultats concernant la latence.

Les performances au niveau PUT sont très satisfaisantes. La latence de 4µs est à mettre en regard avec la latence du matériel qui est égale à 1.7µs, soit une latence de 2.3µs due à la mise en œuvre du protocole par le processeur de la station de travail.

Les latences aux niveaux SLR et PVM sont respectivement de 50µs et 200µs. Ces mauvais résultats peuvent être attribués aux nombreux appels systèmes (sur un PII 350 Mhz un appel système vide coûte 7µs). L'utilisation des interruptions est également très coûteux : il faut donc minimiser leur nombre.

L'extensibilité est excellente. La latence ajoutée par un niveau de routage supplémentaire est négligeable.

La carte réseau n'est pas programmable, aucune modification de son protocole n'est possible.

Sa commande, utilisant les accès en configuration sur le bus PCI, est également un défaut important, car ce type d'accès est de plus en plus mal supporté par les nouvelles générations de bridge PCI.

3.4. Synthèse

L'analyse des différentes architectures présentées permet de préciser les objectifs de notre travail. En reprenant un par un l'ensemble des critères que nous avions considérés comme pertinents vis-à-vis de nos objectifs, nous allons pouvoir préciser les points sur lesquels devront porter nos efforts.

3.4.1. Le débit

Il apparaît en premier lieu que les protocoles standards des LAN, tels que ceux mis en œuvre dans la machine HIVE, ne permettent pas d'atteindre des débits suffisants.

Lors de l'utilisation de protocoles spécifiques, le débit utilisable est déterminé par les caractéristiques du bus d'entrée/sortie et du réseau : le débit du bus PCI supérieur à 100 Mo/s est suffisant, le débit du réseau doit être du même ordre.

Les « grappes de PC » utilisant les réseaux Myrinet ou HSL permettent d'atteindre ces débits.

3.4.2. La latence

La latence est certainement le point le plus critique pour permettre l'utilisation des « grappes de PC », pour des applications parallèles. Les protocoles de bas niveaux, tels que BIP ou PUT, permettent d'obtenir des latences égales à 4µs. Ces latences sont tout à fait satisfaisantes pour les applications parallèles à gros grains adaptées aux « grappes de PC ».

Ces protocoles de bas niveau souffrent généralement de défauts qui les rendent difficilement directement utilisables par les applications :

- ils nécessitent l'utilisation d'adresses physiques ;
- ils ne supportent qu'un seul utilisateur à la fois ;
- ils sont fortement liés à l'utilisation d'un matériel spécifique.

L'ensemble de ces défauts a pour conséquence que les applications sont généralement écrites en utilisant des bibliothèques de communication standards, telles que PVM ou MPI. Or, comme nous le voyons sur le graphique 3.23, les latences correspondant à ces bibliothèques de communications sont d'un tout autre ordre de grandeur.

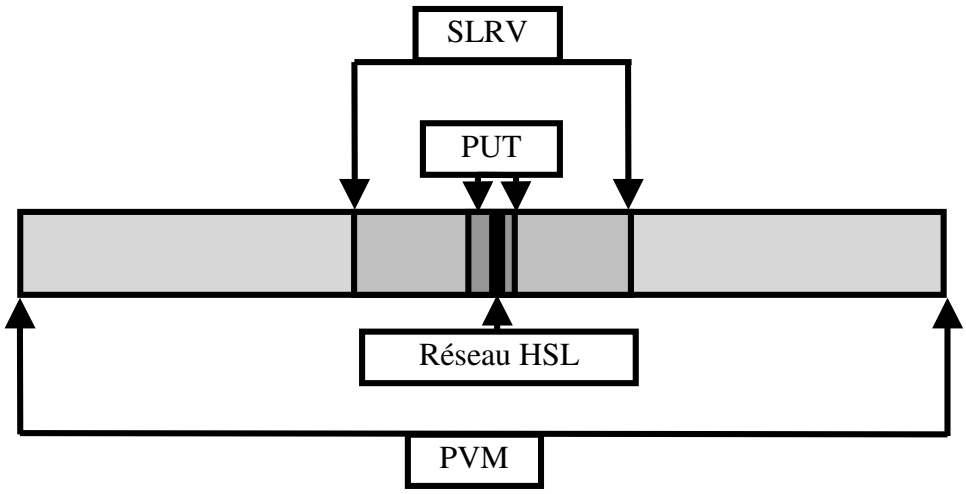


Figure 3.23 Comparaison des latences pour MPC0.

En observant ce graphique, on constate qu'une latence réseau égale à 0 ne réduirait la latence visible par une application que de moins de 1%.

Pour réduire cette latence, le principal effort doit donc porter sur les protocoles de communication et leur mise en œuvre.

3.4.3. Le surcoût pour le nœud de calculs

Afin de permettre le recouvrement du temps de communication par du temps de calcul, la part réalisée par la station de travail dans un transfert doit être la plus minime possible. Dans l'état actuel, 99% de la mise en œuvre des bibliothèques de communication, telle que PVM, est effectuée par le processeur de la station de travail, ce qui empêche tout recouvrement des communications par le calcul.

3.4.4. La fiabilité

La plupart des réseaux étudiés permettent la détection des erreurs. Seul le protocole sécurisé de la machine MPC0 permet une reprise sur erreur [FENYO].

3.4.5. L'extensibilité

Aussi bien pour Myrinet que pour HSL, il existe des routeurs permettant de construire un réseau possédant un nombre important de nœuds. L'implémentation actuelle du protocole BIP utilise des messages de taille très importante, ce qui constitue une source de contention lors de l'utilisation de plusieurs niveaux de routeurs.

3.4.6. La programmabilité

Seule la carte Myrinet offre la possibilité de modifier le protocole de bas niveau, mais la programmation du processeur embarqué dans le LANAI est complexe. De plus le fonctionnement du gestionnaire de paquet du LANAI (bloquage du réseau tant que le processeur n'a pas lu l'en-tête) limite très fortement les possibilités de programmation.

3.4.7. Récapitulatif

L'ensemble des caractéristiques des trois machines présentées peuvent être récapitulées dans un tableau :

	Débit	Latence	Surcoût Processeur	Extensibilité	Fiabilité	Programmabilité
The HIVE	Faible	Elevée	Elevé	Faible	Corrigée	Nulle
BIP/Myrinet	Elevé	Faible	Moyen	Moyenne	Bonne	Moyenne
MPC0	Elevé	Faible	Moyen	Excellente	Corrigée	Faible

3.5. Conclusion

Les enseignements tirés de l'analyse des « grappes de PC » présentées nous amènent à définir quelles devraient être les caractéristiques d'un contrôleur de réseau adapté à ce type de machines parallèles :

- il doit pouvoir être connecté au bus d'entrée/sortie le plus performant du moment ;
- il doit permettre de communiquer avec un réseau à haut débit, faible latence et grande extensibilité ;
- il doit être directement accessible depuis les processus applicatifs, sans nécessiter d'appels au système d'exploitation ;
- il doit pouvoir être commandé simultanément par plusieurs processus ou thread ;
- il doit supporter un mode de fonctionnement de type « wormhole », pour éviter les recopies ;
- il doit être facilement programmable, pour permettre les adaptions et les expérimentations de protocoles.

La suite de ce document sera consacré à la présentation d'un contrôleur de réseau, satisfaisant à ces contraintes.

CHAPITRE 4

Le protocole RWU

Comme nous l'avons vu dans le chapitre précédent, pour exploiter les performances d'un réseau à haut débit et faible latence, il est nécessaire d'utiliser un protocole spécifique.

Avant de définir l'architecture d'un nouveau contrôleur réseau, nous avons donc souhaité redéfinir un protocole qui devra pouvoir être mis en œuvre de façon efficace par cette architecture.

Ce chapitre va être consacré à la présentation du protocole RWU [RWU], dont l'objectif est de remédier aux limitations des protocoles existants. Ce protocole est le fruit de notre collaboration avec Pierre David, dans le cadre d'une coopération entre le LIP6 et le laboratoire PRiSM[22].

4.1. Les objectifs

Le protocole RWU est fortement inspiré de celui mis en œuvre par le composant d'interface PCIDDC de la machine MPC0. Les objectifs de ce protocole sont :

- exploiter la performance du matériel : le réseau offre un haut débit et une très faible latence, l'objectif de ce protocole est qu'une application puisse directement bénéficier des performances du réseau ;
- limiter l'intervention du processeur de la station de travail lors de l'envoi et de la réception d'un message ;

[22] Parallélisme Réseaux Systèmes Modélisation.

- fournir une interface de programmation élémentaire : cette interface doit offrir le minimum de primitives nécessaires aux communications et doit pouvoir servir de base à la mise en œuvre de bibliothèques standards de passages de messages ;
- afin de limiter la latence des communications, l'utilisation des appels au système d'exploitation doit être réservé aux procédures d'initialisation de la communication. Ils ne doivent pas être utilisés lors de communications proprement dites [Mark97] ;
- les processus manipulent des adresses virtuelles. Les primitives de communications ne doivent pas manipuler des adresses physiques. La déréférenciation virtuelle vers physique étant coûteuse en terme de latence et de surcoût pour le processeur de la station de travail, le contrôleur d'interface réseau doit être capable de manipuler des adresses virtuelles ;
- pour que ce protocole puisse être utilisé dans un contexte général muti-utilisateurs et faciliter la mise au point des applications, ce protocole doit offrir un minimum de protection [Lee98]. Les autorisations d'accès au différentes zones mémoires doivent être vérifiées par le contrôleur réseau ;
- ce protocole doit pouvoir fonctionner avec des nœuds de calculs multiprocesseurs, ainsi que dans des environnements à files d'exécution multiples.

4.2. Les principes de fonctionnement

La primitive élémentaire qui est offerte par le protocole RWU est le « Remote Write ». Cette primitive est comparable à celle offerte par le composant PCIDDC qui ne manipule que des adresses physiques et ne possède qu'une unique interface de commande. Pour permettre son utilisation par plusieurs processus, le composant PCI-DDC doit être contrôlé par le système d'exploitation. Ce passage obligé par le système d'exploitation lors des communications pénalise très fortement les performances.

4.2.1. Composant virtuel et composant réel

Le but est de rendre accessible la fonctionnalité de PCIDDC en mode utilisateur, en offrant à chaque processus l'illusion qu'il possède son propre contrôleur réseau.

Chaque nœud, dans la réalité, possède un unique contrôleur réseau, ce qui nous amène à définir deux composants distincts :

- le composant réel : c'est la carte d'interface qui est connectée au bus PCI. Elle est configurée par le système d'exploitation et elle interprète les ordres venant des processus ;

- le composant virtuel : c'est l'interface offerte par le composant réel aux processus. Chaque processus dispose d'un ou de plusieurs composants virtuels (plusieurs composants virtuels peuvent être utiles dans un environnement « multi-thread »).

Le composant PCIDDC offre un unique espace de commande constitué d'un ensemble de registres permettant de lui donner des ordres. Du point de vue de sa commande, un composant virtuel est comparable à un composant PCIDDC. Chaque composant virtuel offre un espace de commande spécifique, uniquement accessible par le processus associé à ce composant virtuel. Un processus contrôle son composant virtuel par des écritures et des lectures mémoire. Ces accès sont dirigés vers le composant réel qui les interprète.

4.2.2. Zone de contrôle du composant virtuel

A chaque composant virtuel est associé un numéro. L'ensemble des zones de contrôle des composants virtuels (VICCA[23]) sont regroupées dans une unique table contiguë en mémoire physique.

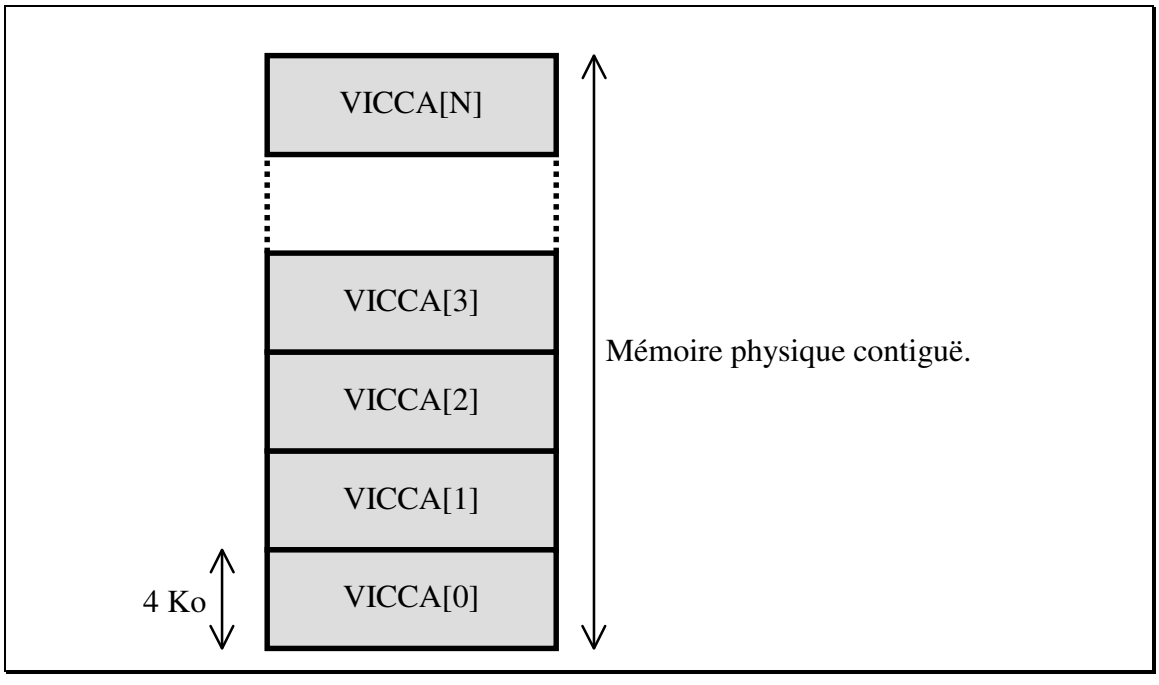


Figure 4.1 Zone des VICCA

Lorsqu'un processus veut commander une émission, il écrit dans sa VICCA. Cette écriture correspond à une adresse de la mémoire physiquement localisée sur la carte d'interface réseau.

[23] Virtual Interface Component Control Area.

En fonction de l'adresse physique transmise au composant réel, celui-ci peut en déduire le numéro du composant virtuel correspondant. Ce mécanisme permet au composant réel d'identifier le processus qui demande la communication et ainsi de vérifier les droits d'accès aux zones mémoires concernées par la communication.

L'abstraction apportée par le composant virtuel aux processus est du même type que celles apportées par le système d'exploitation UNIX. Chaque processus a l'illusion de posséder son propre contrôleur réseau comme il a l'illusion de posséder son propre processeur grâce au mécanisme de temps partagé, et il a l'illusion de posséder sa propre mémoire grâce au mécanisme de MMU[24].

4.2.3. Interface avec le système d'exploitation

Comme nous l'avons vu, l'utilisation des appels système est coûteuse en terme de latence. Pour rendre minimale cette latence mais aussi pour limiter le surcoût pour le processeur de calculs, les appels système ne doivent pas être utilisés lors des communications. Néanmoins, certains traitements nécessitant l'attribution de ressources tels que l'allocation des zones de communication ou l'établissement des canaux nécessitent l'utilisation d'appels système. Le coût de ces appels peut être négligé car il n'intervient pas lors des communications. L'émission d'un message ne nécessite qu'une écriture mémoire de la part du processus émetteur, et l'intégralité des traitements est assurée par le contrôleur d'interface.

4.2.4. Les Canaux

Un canal est un lien entre deux processus désirant communiquer par le biais de leurs composants virtuels respectifs. A chaque canal correspond un numéro. Les canaux sont alloués par le système d'exploitation.

Lors d'une émission, il suffit au processus d'indiquer le numéro du canal qu'il veut utiliser pour désigner le processus, ainsi que le nœud destinataire du message. Les mécanismes de diffusion ne sont pas supporté par RWU : à un canal correspond un unique destinataire.

Un composant virtuel peut gérer plusieurs canaux pour permettre à un processus de pouvoir communiquer avec plusieurs autres processus.

Un canal entre deux nœuds A et B est identifié par un numéro sur chacun des nœuds A et B. Ce numéro peut être différent sur les deux nœuds.

[24] Memory Management Unit.

4.2.5. Zone mémoire de communication

Les zones de mémoires (CMA[25]) contenant les données à transférer doivent préalablement avoir été allouées par le système d'exploitation. Ces zones doivent être contiguës en mémoire physique. Dans l'objectif de limiter les traitements effectués par le composant réel, à chaque canal correspond une unique zone mémoire de communication. Cette contrainte est raisonnable pour des applications parallèles. Dans tous les cas, ces zones doivent être verrouillées en mémoire pour éviter qu'elles soient « swappées » sur le disque [Wels98].

4.2.6. Les files d'émission et de réception

A chaque composant virtuel est associé une file d'émission contenant les descripteurs de messages à émettre (TxQ[26]) et une file contenant les descripteurs des messages reçus (RxQ[27]).

4.2.7. Gestion des adresses

Lors de l'allocation des canaux par le système d'exploitation, les adresses physiques et virtuelles ainsi que le numéro du composant virtuel correspondant à la CMA sont transmis au contrôleur réseau.

Quand un processus demande l'envoi d'un message, le composant réel est en mesure de vérifier que la zone de mémoire où se trouvent les données lui appartient. C'est pour que ce traitement soit le plus simple possible, et donc la latence minimale, que chaque composant virtuel ne dispose que d'une unique CMA par canal (ce qui correspond à une unique zone mémoire physiquement contiguë).

4.2.8. Les Tables

La gestion du protocole RWU utilise principalement deux tables :

- la table des composants virtuels (VICT[28]) : pour chaque composant virtuel, cette table contient les adresses des files TxQ et RxQ ainsi qu'un ensemble d'indicateurs propres à ce composant virtuel.
- La table des canaux (CT[29]) : chaque entrée dans cette table contient les adresses physiques et virtuelles de la CMA, la longueur de la CMA, le numéro du composant virtuel associé,

[25] Communication Memory Area

[26] Transmit Queue.

[27] Receive Queue.

[28] Vitual Interface Component Table.

le numéro du nœud distant ainsi que le numéro du canal correspondant sur le nœud distant.

4.2.9. Emission d'un message

Lors de l'émission d'un message, le processus en mode utilisateur :

1. écrit le message dans la zone mémoire de communication, si ce n'est déjà fait;
2. écrit un descripteur d'émission dans la file d'émission;
3. écrit le nombre de descripteurs ajoutés dans la zone de contrôle du composant virtuel.

Ces étapes sont résumées en partie gauche de la figure ci-dessous :

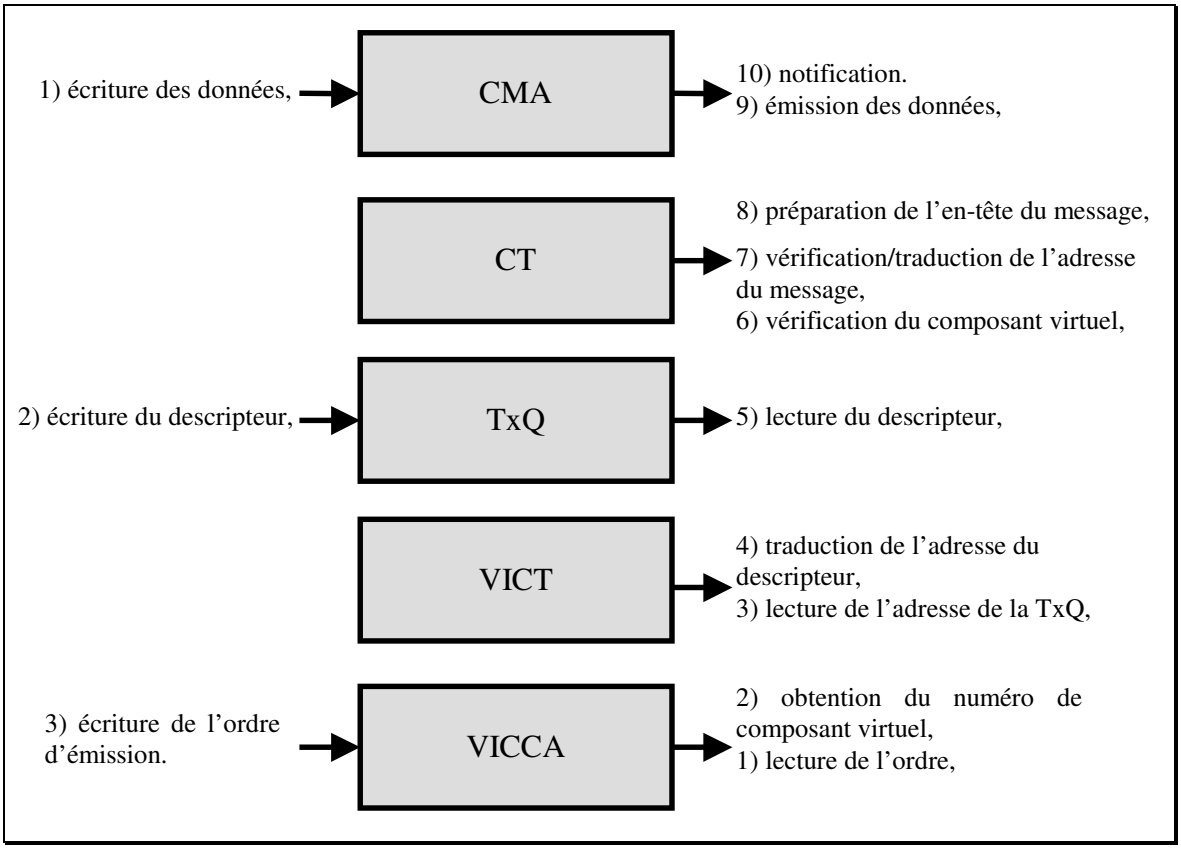


Figure 4.2 synoptique d'une émission

Les actions réalisées par le composant sont résumées sur la partie droite :

[29] Chanel Table.

1. le composant réel récupère l'ordre donné au composant virtuel, à travers la zone de contrôle ;
2. à partir de l'adresse physique émise par la MMU sur le bus PCI, il en déduit le numéro du composant virtuel ;
3. à partir de la table des composants virtuels, il lit l'adresse de la zone des descripteurs d'émission ;
4. à partir de l'adresse physique du début de la TxQ, il obtient l'adresse physique du descripteur écrit par le processus ;
5. le composant peut ensuite lire le descripteur ;
6. le descripteur contenant le numéro de canal, le composant accède à la table des canaux et vérifie que le canal est bien ouvert par le processus ;
7. la table des canaux contient également les paramètres de la zone mémoire de communication : le composant peut alors vérifier la validité de l'adresse et la longueur du message, et traduire l'adresse de début du message ;
8. tout est réuni, le composant peut maintenant envoyer l'en-tête du message ;
9. puis, il peut réaliser l'émission des données ;
10. lorsque toutes les données ont été émises, selon la configuration du composant virtuel, le composant réel peut éventuellement interrompre le processeur.

4.2.10. Réception d'un message

La réception est symétrique:

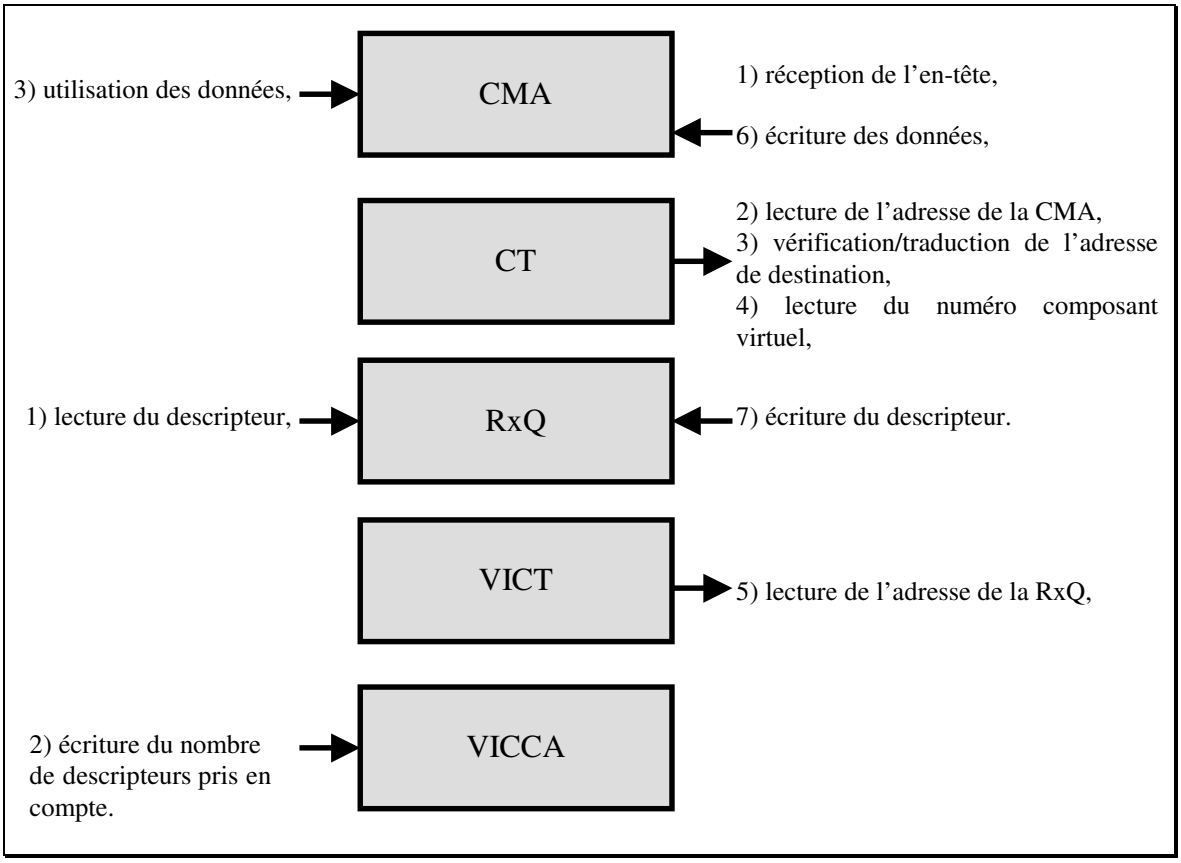


Figure 4.3 synoptique d'une réception

Les actions réalisées par le composant récepteur sont résumées sur la partie droite :

1. il reçoit l'en-tête du message, c'est-à-dire le numéro de canal vu du côté récepteur, et l'adresse virtuelle où il doit placer les données reçues ;
2. puis, à partir du descripteur de canal lu dans la table des canaux, il obtient les caractéristiques de la zone mémoire de communication ;
3. il peut donc vérifier et traduire l'adresse virtuelle qui lui a été transmise par le processus émetteur ;
4. toujours dans cette table, il lit le numéro de composant virtuel associé au canal ;
5. ayant le numéro de composant virtuel, le composant réel peut donc lire la table des composants virtuels et en déduire l'adresse de la RxQ ;
6. les données sont ensuite reçues du réseau et placées en mémoire ;
7. un nouveau descripteur est écrit, pour signaler l'arrivée d'un message ;
8. et une interruption est éventuellement générée, pour signaler l'arrivée d'un nouveau message.

Le processus récepteur peut alors accomplir les actions suivantes, résumées sur la partie gauche du synoptique :

1. le processus consulte la file des descripteurs de messages reçus (RxQ) ;
2. il indique, par une écriture dans la VICCA, qu'il a pris en compte le descripteur ;
3. puis, il peut utiliser les données.

4.3. La sécurité

Les mécanismes du protocole RWU permettent de garantir la sécurité à différents niveaux :

- chaque processus ne peut accéder qu'à sa propre file d'émission (protection mémoire UNIX) ;
- lors de l'envoi d'un ordre au contrôleur d'interface réseau, l'adresse émise sur le bus PCI permet d'identifier le processus émetteur (mécanisme garanti par la MMU) ;
- le contrôleur d'interface réseau connaissant le numéro du processus émetteur, il peut vérifier grâce aux tables qui lui sont fournies par le système, si les adresses contenues dans le descripteur placé dans la TxQ correspondent à la zone de communication allouée à ce processus ;
- lors de la réception d'un message, le contrôleur d'interface réseau interprète le numéro de canal ainsi que l'adresse virtuelle qui lui ont été transmis, pour calculer à l'aide des tables fournies par le système d'exploitation l'adresse physique de destination des données. Ce mécanisme garantit que les données ne peuvent être déposée ailleurs que dans une zone qui a précédemment été allouée à cet effet.

L'ensemble de ces mécanismes permet de garantir l'intégrité du système d'exploitation et la non corruption des processus entre eux.

4.4. VIA versus RWU

Parmi les protocoles bas niveaux existants nous pouvons citer le protocole VIA[30] qui est proposé par un consortium regroupant Intel, Compaq et Microsoft. Ce protocole est basé sur l'utilisation de files de descripteurs d'émissions et de réceptions. Comme dans le cas de Myrinet, les descripteurs de messages peuvent contenir une liste de zones mémoires constituant un message. La particularité du protocole VIA est d'introduire le concept d'interface virtuelle : chaque processus dispose de ses propres files d'émission et de réception.

[30] Virtal Interface Architecture for San's [Dunn98] [Eick98].

Bien que n'étant pas présenté dans ce document, nous pouvons rappeler rapidement les différents points communs et principales différences entre le protocole RWU et le protocole VIA :

- **notion de composant virtuel** : VIA propose une notion équivalente qui est l'interface virtuelle (VI). A chaque VI est associé une interface de commande spécifique accessible par un unique processus ;
- **canal** : VIA n'utilise pas cette notion. A chaque VI est associé un unique VI distant, un processus dispose d'autant de VI que d'interlocuteurs ;
- **adresses virtuelles** : de même que RWU, VIA interprète les adresses virtuelles grâce à une table de correspondance gérée par le système d'exploitation ;
- **primitives** : l'interface de VIA est plus riche que celle de RWU, car elle offre trois types de primitives : écriture distante, lecture distante, message de type « send receive » ;
- **zones de communications** : contrairement à RWU, VIA supporte les zones discontinues en mémoire physique et permet, suivant certaines limites, l'agrégation et la fragmentation de zones discontinues en mémoire virtuelle ;
- **protection mémoire** : le mécanisme utilisé par VIA est différent de celui de RWU. Pour VIA, à chaque zone mémoire est associée une clé, qui est utilisée par le contrôleur d'interface pour vérifier si un processus est autorisé à accéder à cette zone mémoire.

D'une façon générale le protocole RWU est plus « minimal » que le protocole VIA. Son objectif est de fournir les services indispensables aux communications avec les meilleures performances. Il apparaît, d'ailleurs, que la complexité de VIA nuit à ces performances : la latence obtenue en utilisant un réseau Myrinet est de 50µs qui sont à comparer avec les 4µs obtenus par le protocole BIP pour le même support matériel. Nous ne disposons pas encore d'évaluation de performance pour le protocole RWU.

4.5. Conclusion

Comme nous avons pu le constater, seules quelques écritures dans l'espace mémoire du contrôleur réseau sont nécessaires pour déclencher une émission. Du fait de l'utilisation directe des adresses virtuelles, aucun appel système n'est nécessaire. La quasi-totalité de la latence est due aux traitements réalisés par le contrôleur d'interface, ce qui permet de bénéficier du recouvrement du temps de communication par le temps de calcul.

Ce protocole est très bien adapté aux architectures modernes de cartes mères, car aucun traitement spécifique n'est nécessaire pour supporter deux ordres d'émissions simultanées provenant de plusieurs processus ou thread s'exécutant sur des processeurs différents.
Ce protocole permet enfin d'assurer un bon niveau de sécurité entre les processus ce qui facilite la mise au point des programmes et permet l'exécution simultanée de plusieurs applications sur la machine. Cette sécurité permet également d'éviter que le système soit corrompu par une application défectueuse, la stabilité de la machine parallèle est donc considérablement accrue.

CHAPITRE 5

Le contrôleur de réseau NOÉ

Le chapitre précédent nous a permis de présenter le protocole RWU permettant aux applications de tirer parti de la performance du réseau. La définition de ce protocole nous a amené à préciser un ensemble de contraintes concernant le contrôleur réseau. Dans ce chapitre, nous allons présenter NOÉ qui est un contrôleur réseau satisfaisant aux caractéristiques idéales présentées dans l'état de l'art et qui permet de supporter efficacement la mise en œuvre du protocole RWU.

5.1. Les objectifs

L'objectif du contrôleur d'interface réseau est de satisfaire aux critères généraux définis dans l'état de l'art et plus précisément il doit permettre la mise en œuvre efficace d'un protocole tel que RWU :

- il doit répondre aux écritures et lectures mémoires venant directement des processus ;
- il doit répondre à une large plage d'adresses mémoire pour permettre son contrôle simultané par plusieurs processus ;
- il doit être capable de réaliser des traductions d'adresses virtuelles en adresses physiques en utilisant des tables qui lui ont été fournies par le système d'exploitation ;
- lors d'un accès mémoire dont il est cible, il doit pouvoir analyser l'adresse de manière à être capable d'identifier le processus qui est à l'origine de cet accès ;
- il doit être capable de déclencher des interruptions à destination du processeur de la station de travail.

L'ensemble de ces contraintes nous a amené à définir l'architecture d'une carte d'interface NOE-HSL.

5.2. Architecture générale

L'architecture de la carte NOE-HSL s'articule autour d'un composant spécifique NOÉ, d'un microcontrôleur et de la mémoire RAM et ROM.

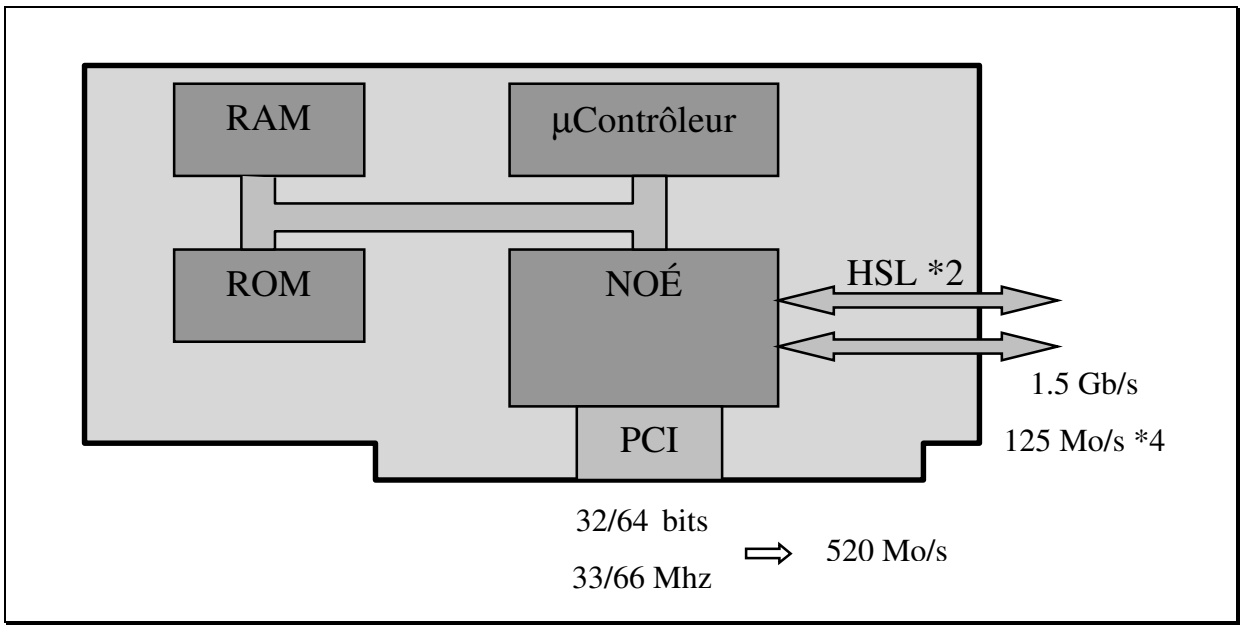


Figure 5.1 La carte NOE-HSL

5.2.1. Le bus d'entrée/sortie

Le bus PCI constitue le standard actuel sur les cartes mères de station de travail de type PC. Il permet à une carte d'extension d'accéder directement à la mémoire située sur la carte mère. Les registres situés à l'intérieur d'un périphérique peuvent être rendu visibles dans l'espace mémoire du processeur de la station de travail, les processus peuvent donc directement accéder aux registres de commande des périphériques. Dans sa version 64 bits / 66 Mhz il offre des performances très élevées (jusqu'à 520 Mo/s). Dans sa version 32 bits / 33 Mhz, il offre déjà une bande passante supérieure à 100 Mo/s.

NOÉ se comporte comme un initiateur ou comme une cible sur le bus PCI ; il est compatible 32/64 bits et supporte les fréquences comprises entre 33 et 66 Mhz.

5.2.2. Le réseau HSL

Cette étude a été réalisée dans le contexte de l'équipe ASIM du laboratoire LIP6 qui est le berceau du réseau HSL. Comme nous l'avons vu dans l'état de l'art, ce réseau possède un bon

débit, une latence très faible et, grâce au routeur RCUBE, il dispose d'une très bonne extensibilité.

Depuis la première génération de la machine MPC, le réseau HSL a évolué et a profité de la progression des technologies CMOS (le débit peut atteindre aujourd'hui 1.5 Gbit/s, soit 125 Mo/s utiles dans chaque direction). Il existe aujourd'hui une version industrielle du routeur RCUBE, utilisé dans la fabrication des commutateurs gigabit Ethernet qui permet de profiter du nouveau débit HSL et qui permet d'en réduire les coûts.

Une version 16*16 de ce routeur est en cours de développement par la société Tachys Technologies.

Les caractéristiques intrinsèques du réseau HSL et la pérennité qui est assurée par son utilisation dans les commutateurs ATM ou Gigabit Ethernet en font un très bon support pour bâtir une machine parallèle du type « grappe de PC ».

De manière à équilibrer les débits et dans la perspective de supporter un sous-réseau HSL spécifique aux transferts des messages de contrôle, la carte NOE-HSL dispose de 2 liens HSL « full duplex ».

5.2.3. Le composant NOÉ

Le composant NOÉ réalise l'interface entre le bus PCI et le réseau HSL. Ce composant est dédié aux transferts de données, à la gestion des DMA vers la mémoire et à la gestion des paquets émis vers le réseau HSL et reçus du réseau HSL.

5.2.4. Le microcontrôleur

Le protocole de communication est réalisé par un microcontrôleur situé sur la carte NOÉ-HSL. Il s'agit d'un microcontrôleur RISC 32 bits du type Motorola PowerPC.

L'utilisation d'un microcontrôleur permet de rendre programmable le protocole. L'exemple du réseau Myrinet montre les intérêts de cette approche. Elle permet d'adapter les protocoles pour un type d'application spécifique.

Le choix d'un microcontrôleur externe au composant d'interface réseau permet de limiter la complexité de celui-ci. Choisir d'utiliser un microcontrôleur externe rend les accès au composant NOÉ plus coûteux, mais le circuit NOÉ est déjà un composant complexe et, au moment du développement nous ne disposions pas d'un cœur de RISC directement utilisable.

Le microcontrôleur supervise le fonctionnement du composant NOÉ. Les différents registres de commandes et de status de NOÉ sont visibles dans l'espace mémoire du microcontrôleur.

C'est à travers des écritures et des lectures dans les registres de NOÉ, que le microcontrôleur commande les opérateurs de NOÉ.

5.3. Le découpage en opérateurs

Le composant d'interface NOÉ peut être vu comme un ensemble d'opérateurs plus ou moins indépendants les uns des autres. Chacun de ces opérateurs offre une interface de programmation accessible par le microcontrôleur.

L'écriture par le microcontrôleur dans un registre appartenant à un opérateur de NOÉ déclenche l'action de cet opérateur. Les ordres ainsi transmis sont d'assez haut niveau sémantique, l'exécution par l'opérateur de l'opération commandée dure généralement plusieurs dizaines de cycles.

Lorsqu'un opérateur a terminé l'opération qui lui avait été demandée, il peut notifier la complétude de l'opération en écrivant un 1 dans le bit correspondant du status général de NOÉ.

La présence d'un 1, dans ce registre, peut éventuellement déclencher une interruption dirigée vers le microcontrôleur.

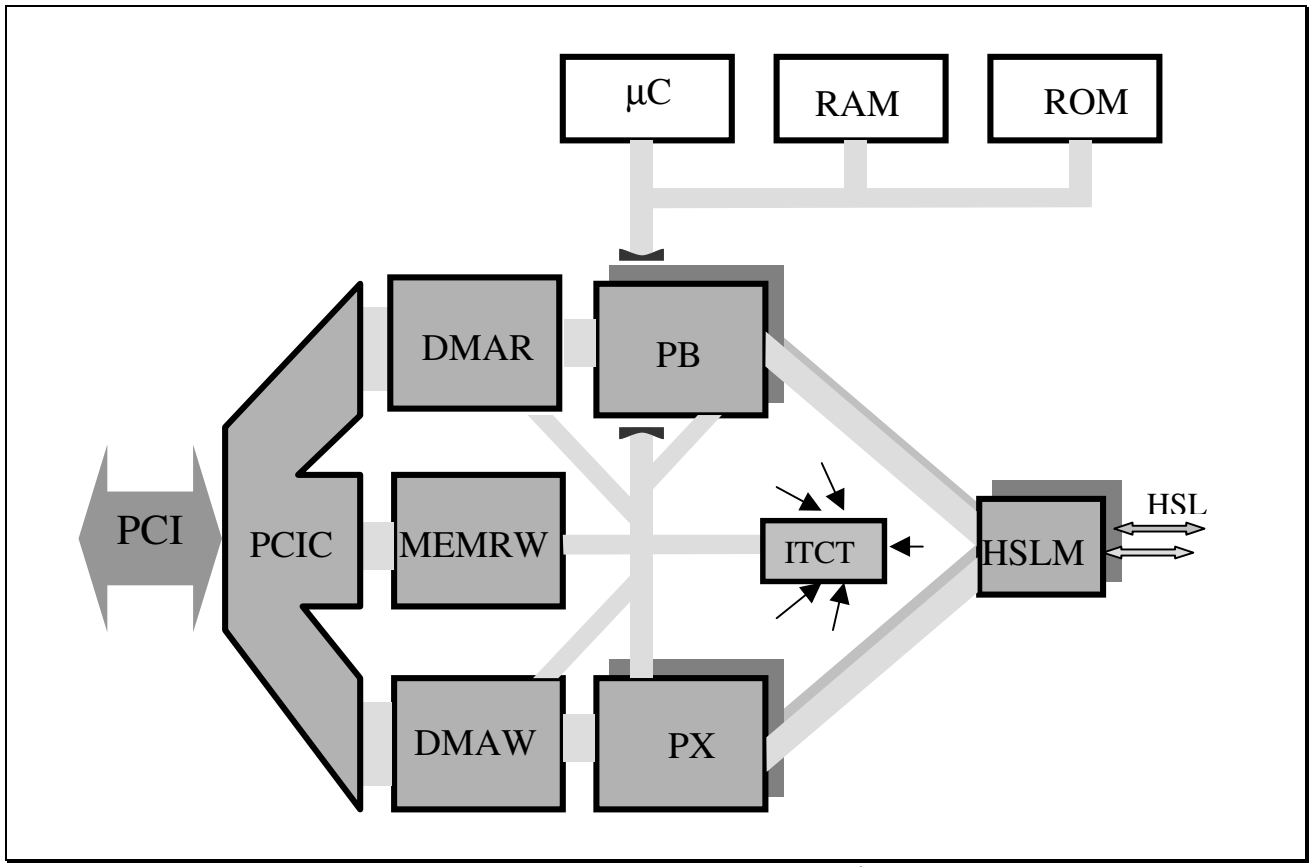


Figure 5.2 Les opérateurs de NOÉ.

L'ensemble des opérateurs est relié au microcontrôleur par l'intermédiaire d'un bus de contrôle de type PI-Bus[31]. Le bloc nommé PCIC[32] n'est pas un opérateur, au sens qu'il n'est pas directement programmable par le microcontrôleur : il est chargé de la mise en œuvre du protocole PCI, aussi bien en mode initiateur qu'en mode cible.

NOE disposant de deux accès au réseau HSL, les opérateur PB et PX (réalisant la mise en forme des paquets émis sur le réseau et l'extraction des paquets provenant du réseau) ont été dupliqués. Cette caractéristique fait qu'en pratique NOE dispose de 2 chemins de données A et B correspondant chacun à un lien HSL. Le bus PCI étant unique, les opérateurs qui lui sont associés n'ont pas été dupliqués. NOE contient donc 2 PB[33], 2 PX[34] et 2 HSLM[35] mais une unique instance des opérateurs liés à la gestion du bus PCI.

Nous allons maintenant détailler un par un les 7 types d'opérateurs présents dans NOÉ :

5.3.1. MEMRW

L'opérateur MEMRW[36] est destiné à traiter les requêtes PCI dont le composant NOÉ est cible. NOÉ rend accessible depuis le bus PCI deux espaces mémoire distincts :

- le premier d'une dimension de 4 Ko contient un ensemble de registres de contrôle et de status du composant NOÉ. Il contient :
    - un registre d'interruption 32 bits à destination du microcontrôleur (ce registre est uniquement accessible en écriture par le processeur de la station de travail) ;
    - un registre d'interruption 32 bits à destination du processeur de la station de travail (ce registre est uniquement accessible en écriture par le microcontrôleur) ;
    - un banc de 32 registres d'usage général (accessible uniquement en écriture par le processeur de la station de travail) ;
    - un banc de 32 registres d'usage général (accessible uniquement en écriture par le microcontrôleur) ;
    - un ensemble registres permettant la configuration du composant NOÉ.

[31] Peripheral Interconnect.

[32] PCI Bus Controleur.

[33] Packet Builder.

[34] Packet Extractor.

[35] HSL Module.

[36] MEMory Read Write.

Tous ces registres sont accessibles en lecture aussi bien par le processeur de la station de travail que par le microcontrôleur ;

- le second dont la dimension peut être configurée par le microcontrôleur entre 4Ko et 4Go sert à recevoir les écritures et les lectures dont le microcontrôleur est la destination.

Cet opérateur dispose d'une pile de type « fifo », qui lui permet de poster jusqu'à 16 requêtes à destination du microcontrôleur. Il contient également un registre destiné à répondre aux lectures dont le microcontrôleur est la destination.

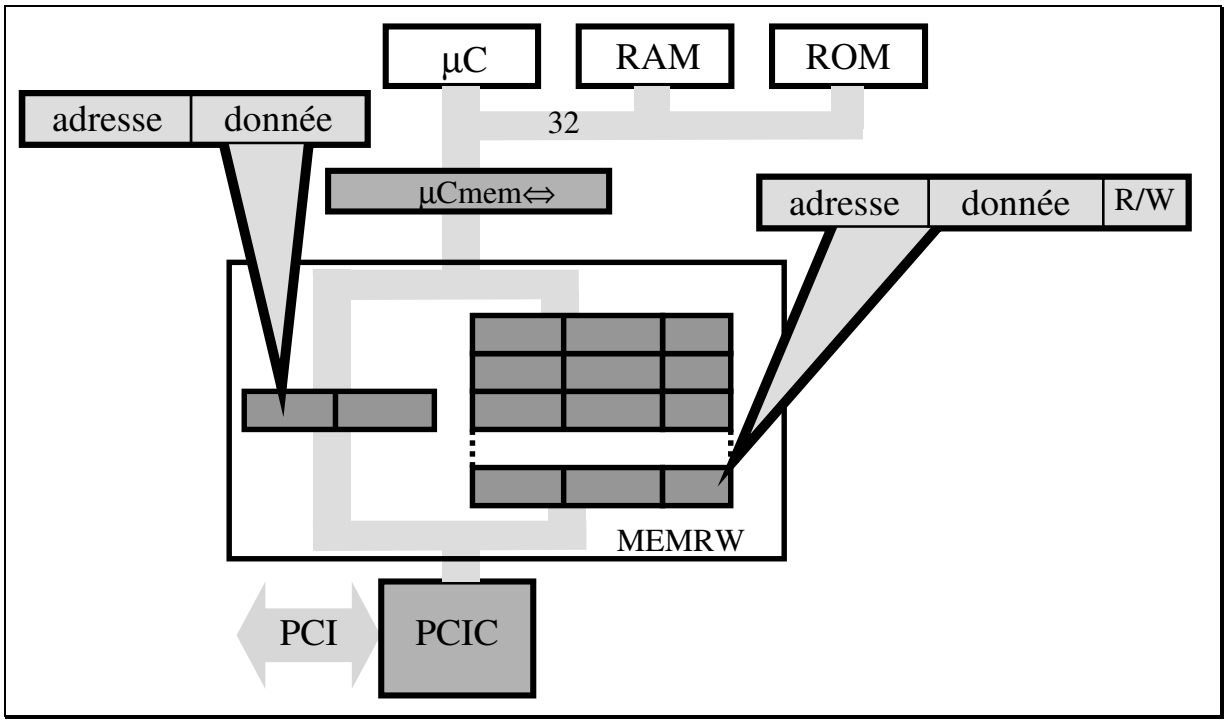


Figure 5.3 Détail de MEMRW

La fifo permet de poster aussi bien des requêtes d'écriture que des requêtes de lecture à destination du microcontrôleur. Nous appelons « Read Buffer » le dispositif destiné à répondre aux lectures à destination du microcontrôleur, il peut être modélisé comme un automate à 3 états :

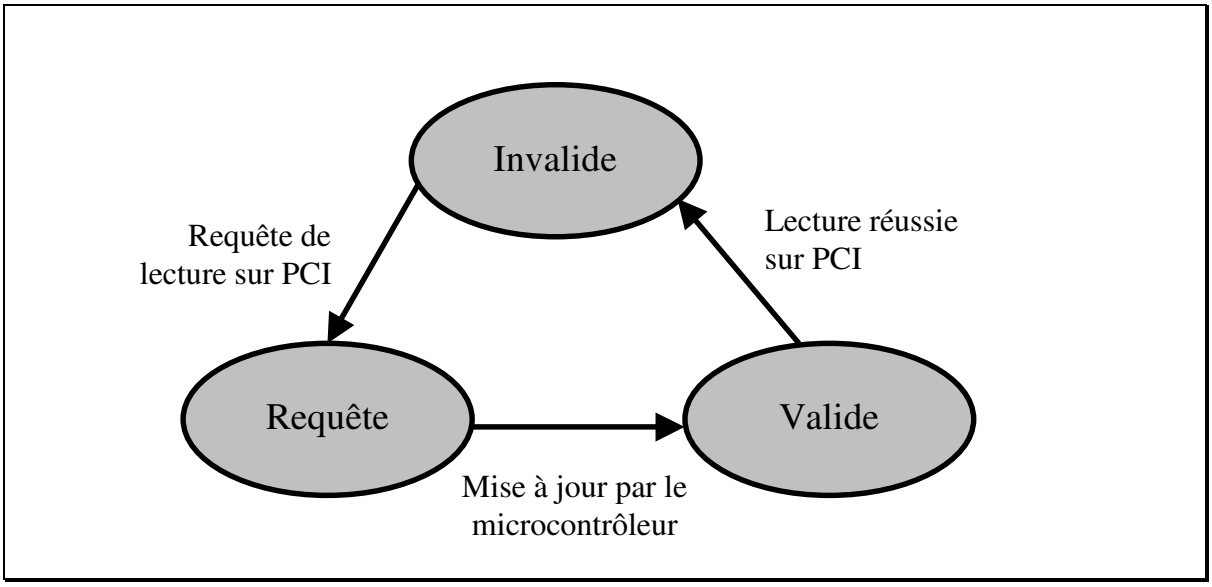


Figure 5.4 Fonctionnement du "Read Buffer"

- **Invalide** : le « Read Buffer » ne contient aucune donnée pertinente.
- **Requête** : une requête de lecture a été reçue sur le bus PCI, NOÉ a demandé à l'initiateur de la requête de la réitérer plus tard. La requête est transmise au microcontrôleur et l'adresse est écrite dans le « Read Buffer », afin d'éviter de poster une nouvelle requête au microcontrôleur concernant la même adresse.
- **Valide** : le microcontrôleur a mis à jour la donnée contenue dans le « Read Buffer », elle restera valide tant que la lecture n'aura pas été répétée et satisfaite, ou que le microcontrôleur ne l'aura pas invalidée.

Contrairement aux écritures qui sont postées dans la fifo de requête, les lectures peuvent être très coûteuses pour leur initiateur. Pour donner des ordres au contrôleur réseau mieux vaut donc utiliser des écritures.

Deux bits du registre général de status de NOÉ indiquent la présence d'une requête de lecture ou d'une requête d'écriture pendante : « Posted Read » et « Posted Write ». Le microcontrôleur peut à ce moment aller lire l'adresse et la donnée dans le cas d'une écriture dans deux registres correspondant à la sortie de la fifo de requêtes. C'est la lecture de l'adresse qui déclenche le « pop » dans la fifo.

5.3.2. DMAR

L'opérateur DMAR correspond au fonctionnement de NOÉ en tant qu'initiateur sur le bus PCI. Il permet d'effectuer des lectures en mode DMA, dans la mémoire de la station de travail.

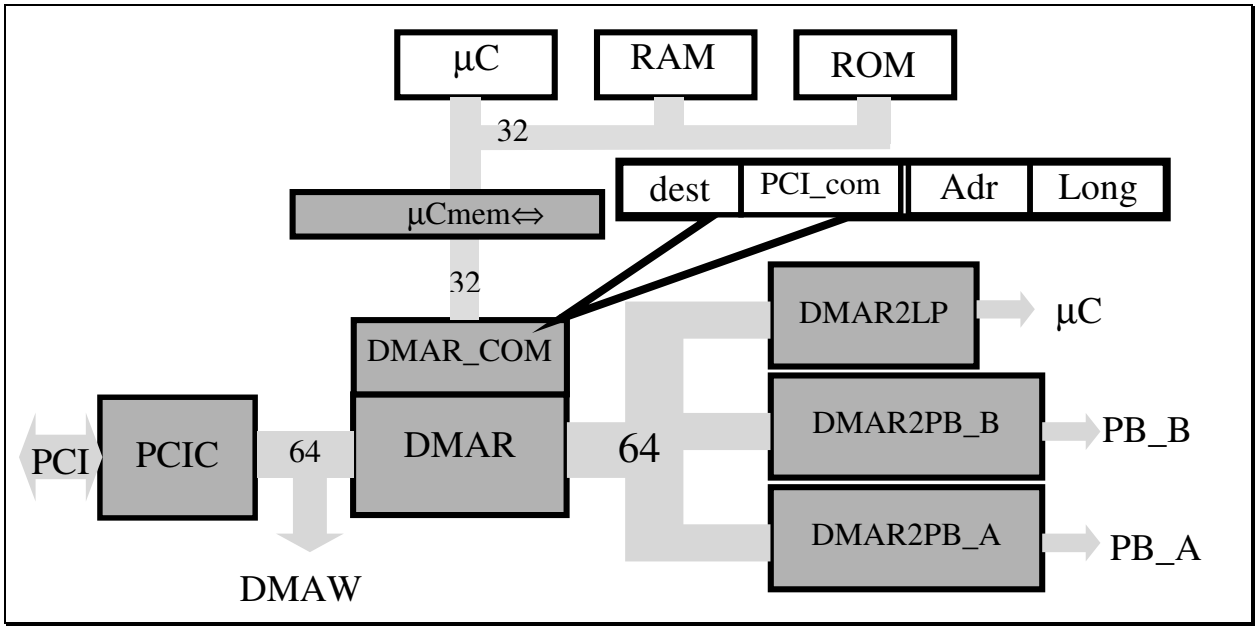


Figure 5.5 Détail de l'opérateur DMAR

Du point de vue des données, cet opérateur réalise l'interface entre le bus PCI et 3 fifos correspondant à 3 destinations spécifiques pour les données :

- **DMAR2LP** : les données qui sont placées dans cette fifo sont à destination du microcontrôleur. Dans le cas d'une émission, il peut s'agir du descripteur correspondant au message à émettre.
- **DMAR2PB_B** : les données qui transitent dans cette fifo sont destinée à être consommées par le « Packet Builder B ». Ces données peuvent être utilisées pour constituer le corps du message.
- **DMAR2PB_A** : la fonction de cette fifo est identique à DMAR2PB_B mais concerne le chemin de donnée A.

Cet opérateur réalise les opérations qui correspondent aux commandes qui lui sont transmises par le microcontrôleur. Il dispose d'une fifo permettant de contenir 2 commandes comportant les champs suivants :

- la destination des données indique dans quelle fifo devront être écrites les données lues dans la mémoire de la station de travail (DMAR2LP, DMAR2PB_A ou DMAR2PB_B) ;
- l'adresse correspondant à la première donnée à lire dans la mémoire de la station de travail. Les zones mémoires à lire doivent être contiguës en mémoire physique. L'adresse fournie à DMAR est une adresse octet, aucune contrainte d'alignement n'est exigée par cet opérateur. L'adresse fournie à DMAR est une adresse 64 bits ce qui lui permet d'être compatible avec les nouvelles architectures de processeurs ;
- la longueur indique à l'opérateur le nombre d'octets à lire. La longueur peut être quelconque comprise entre 1 octet et inférieure à 256 Koctets ;
- le champ PCI_Com permet au microcontôleur de préciser le type de commande PCI à utiliser : en effet, sur le bus PCI, une lecture mémoire peut correspondre à différentes commandes suivant la longueur des données et la stratégie adoptée vis-à-vis des caches du ou des processeurs de la station de travail.

La commande de cet opérateur nécessitant l'écriture de plusieurs mots de 32 bits, il a été décidé que ce serait l'écriture de la longueur qui déclencherait effectivement le démarrage de l'opérateur. Si les champs indiquant l'adresse n'ont pas été écrits, l'adresse correspondant à la commande précédente sera utilisée. Cette caractéristique permet de ne pas se soucier des poids forts de l'adresse (bits 32 à 63) dans le cas d'une machine ne supportant que 32 bits d'adresse.
Une fois la commande transmise par le microcontrôleur à l'opérateur, DMAR se charge de sa réalisation de façon autonome (aucune intervention du microcontrôleur n'est nécessaire). Si les fifos de destination deviennent pleines, il interrompt la transaction et ne reprend que lorsqu'il dispose d'une place suffisante pour effectuer une transaction PCI de longueur suffisante.
L'opérateur se charge de maintenir à jour l'adresse courante de la transaction et le nombre d'octets qu'il reste à lire.
Lorsque l'opérateur a terminé l'opération de DMA et qu'il a écrit toutes les données dans la fifo correspondant à la destination des données, il en informe le microcontrôleur en positionnant à 1 le bit « Dmar_Done » du registre général de status de NOE. Cette écriture peut éventuellement déclencher un interruption destinée au microcontrôleur.
Si le microcontrôleur poste dans la fifo de commande de DMAR, deux ordres successifs ayant la même destination, il devra s'assurer que la même séquence devra être respectée pour lire les données dans la fifo destination.

5.3.3. DMAW

L'opérateur DMAW est le pendant de l'opérateur DMAR mais concerne les écriture en mode DMA, dans la mémoire de la station de travail. NOE se comporte alors comme un initiateur sur le bus PCI.

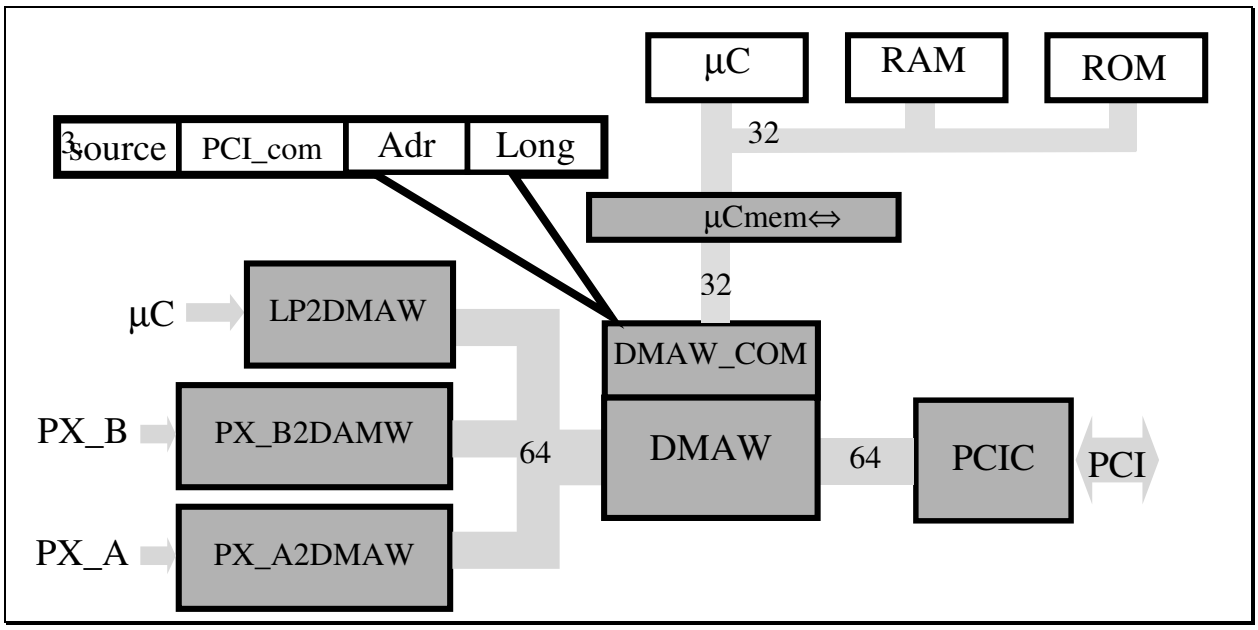


Figure 5.6 Détail de l'opérateur DMAW

Du point de vue des données, cet opérateur réalise l'interface entre le bus PCI et 3 fifos correspondant à 3 sources possibles pour les données :

- **LP2DMAW** : le microcontroleur écrit des données dans cette fifo, ces données sont recopiées par DMAW dans la mémoire de la station de travail. Lorsqu'une réception est terminée le microcontrôleur peut utiliser cette fifo pour écrire un descripteur correspondant au message reçu ;
- **PX_B2DMAW** : les données qui transitent dans cette fifo proviennent du « Packet Extractor » B. Le « Packet Extractor » B a extrait ces paquets qu'il a reçu du réseau ;
- **PX_A2DMAW** : le fonction de cette fifo est identique à PX_B2DMAW mais concerne le chemin de donnée A.

Cet opérateur dispose d'une fifo de commande à 2 cases, lui permettant de mémoriser deux ordres venant du microcontrôleur. Les ordres sont traités de façon séquentielle par l'opérateur. La commande de l'opérateur DMAW est sensiblement la même que celle de DMAR :

- la source des données indique dans quelle fifo devront être lues les données qui seront écrites dans la mémoire de la station de travail (LP2DMAW, PB_A2DMAW ou PB_B2DMAW).
- l'adresse correspond à la première donnée à écrire dans la mémoire de la station de travail. Les zones mémoires à écrire doivent être contiguës en mémoire physique. L'adresse fournie à DMAW est une adresse octet, aucune contrainte d'alignement n'est exigée par cet opérateur. De même que pour DMAR, l'adresse fournie à DMAW est codée sur 64 bits ;
- la longueur indique à l'opérateur le nombre d'octets à écrire, elle peut être quelconque comprise entre 1 octet et inférieure à 256 Koctets ;
- le champ PCI_Com permet au microcontôleur de préciser le type de commande PCI à utiliser.

L'opérateur DMAW prend en compte les indications de remplissage des fifos qui lui servent de source. Si la fifo source du DMA en écriture est vide, il interrompt la transaction PCI et reprend celle-ci plus tard lorsque la fifo est suffisamment remplie. De même que pour DMAR, DMAW maintient constamment à jour l'adresse où il doit écrire, et le nombre d'octets qui lui reste à écrire.

Si le champ indiquant le nombre d'octets à écrire est égal à 0 alors, l'opérateur écrira les données contenues dans la fifo source jusqu'à ce qu'il rencontre un marqueur de fin. Cette caractéristique permet la réception de paquets dont la taille n'est pas inscrite dans l'en-tête.

Lorsque DMAW a fini d'écrire les données, il prévient le microcontrôleur en mettant à 1 le bit « Dmaw Done » du registre général de status de NOE.

De même que pour DMAR, le microcontrôleur devra garantir la même séquence pour les écritures dans les fifos sources et commandes transmises à DMAW.

5.3.4. PB

Le « Paquet Builder » est l'opérateur qui se charge de la mise en forme des paquets et de l'envoi des paquets sur le réseau. D'une façon générale, il assemble les paquets à partir d'un en-tête qui lui a été fourni par le microcontrôleur et de données provenant de l'opérateur DMAR. Il existe deux instances de cet opérateur permettant d'envoyer des paquets sur les deux liens HSL.

L'objectif qui a guidé la définition de l'architecture de cet opérateur est de permettre la plus grande souplesse possible dans le format des paquets. Un utilisateur averti de ce contrôleur de

réseau peut adapter le programme réalisant la mise en œuvre du protocole et s'exécutant sur le microcontrôleur. Il peut également utiliser le format de paquet qui convient le mieux à son application.

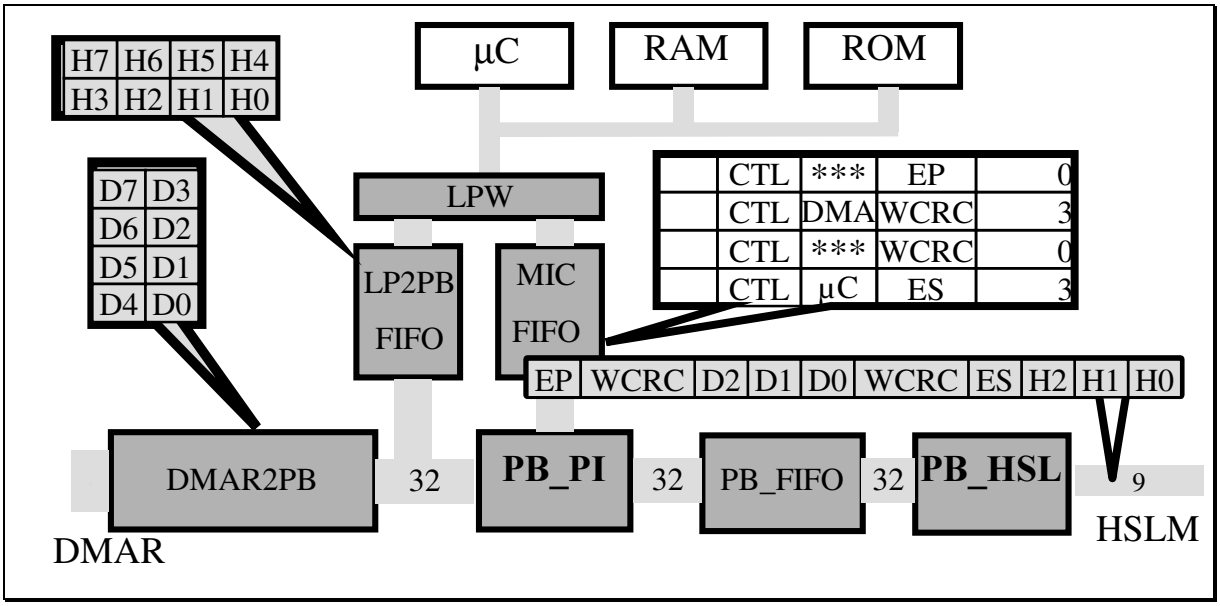


Figure 5.7 Détail du "Packet Builder"

Pour assembler les paquets qui vont être transmis sur le réseau, le « Packet Builder » utilise deux sources de données :

- la fifo DMAR2PB qui contient les données que l'opérateur DMAR a lu dans la mémoire de la station de travail. Ces données constituent généralement le corps de données du paquet ;
- la fifo LP2PB qui contient les données produite par le microcontrôleur. Ces données sont généralement utilisées pour fabriquer les en-têtes ou les fins de paquets.

La figure ci-dessus présente le format des paquets transmis sur le réseau. Un paquet généré par le composant NOÉ se compose d'une ou plusieurs sections. Chaque section est séparé de la suivante par un caractère ES[37]. Le découpage des paquets en sections conditionne la manière dont il va être interprété par l'opérateur de réception de paquet PX. La description de l'opérateur PX nous permettra d'apporter plus de précisions sur l'usage qui est fait de ces sections.

La construction et l'émission d'un paquet par PB sur le réseau HSL sont commandées par le microcontrôleur. La description du paquet à émettre prend la forme d'une séquence de

[37] End Of Section

microinstructions transmises par le microcontrôleur au « Paquet Builder ». Chaque microinstruction correspond à la description d'une fraction de section composée d'un certain nombre de données de même origine (LP2PB ou DMAR2PB), suivie ou non d'un caractère de contrôle. Chaque microinstruction correspond à un mot de 32 bits décomposé en un ensemble de champs :

- la source des données qui peut être soit la fifo DMAR2PB, soit la fifo LP2PB ;
- le nombre d'octets à lire dans la fifo source ;
- un indicateur permettant d'indiquer si les données doivent être suivies d'un caractère de contrôle ou non ;
- le caractère de contrôle qui suit éventuellement les données ;
- un indicateur permettant la notification au microcontrôleur de la fin d'exécution de la microinstruction.

La figure ci-dessous nous montre le résultat d'une séquence particulière de 4 microinstructions :

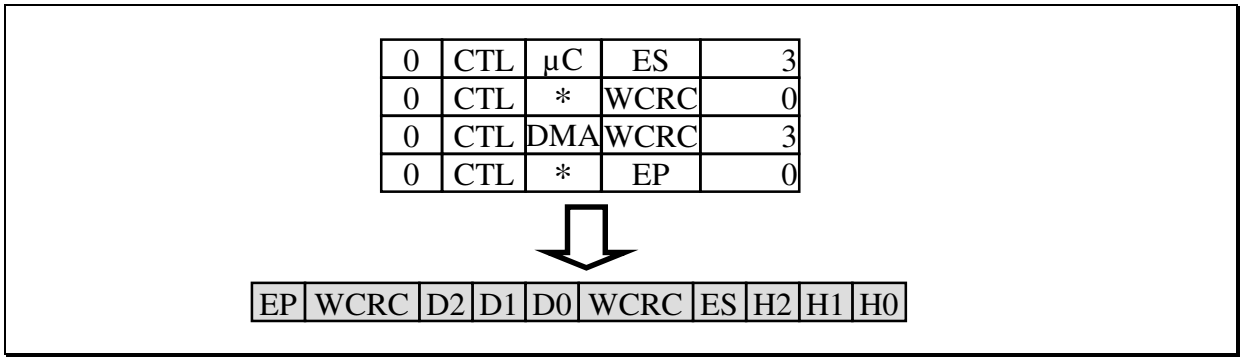


Figure 5.8 De la microinstruction au paquet.

Les microinstructions doivent être lues de bas en haut et le paquet doit être lu de la droite vers la gauche :

- la première microinstrunction indique au PB qu'il doit lire 3 octets dans la fifo contenant les données produites par le microcontrôleur, et y ajouter le caractère de contrôle ES ;
- la seconde microinstruction demande uniquement l'ajout d'un caractère de CRC (mécanisme détaillé plus tard) ;
- la troisième microinstruction correspond à l'ajout de 3 données provenant de l'opérateur DMAR suivi d'un caractère WCRC ;

- la dernière microinstruction sert uniquement à ajouter le caractère EP, pour indiquer la fin du paquet.

Nous pouvons noter que le microcontrôleur ne demande pas d'être prévenu, lorsque le paquet sera intégralement transmis sur le réseau. Si il le voulait, il devrait positionner l'indicateur correspondant à 1, pour la dernière microinstruction.

A chaque caractère transmis sur le réseau HSL est associé un bit de parité, qui permet de garantir l'intégrité des données transmises. Néanmoins, cette parité ne permet pas de détecter une double faute ou la perte d'un caractère complet. Pour remédier à ce problème, il est possible de demander à la macrocellule HSL de réaliser le calcul et l'insertion d'un caractère de CRC codé sur 32 bits. Trois caractères de contrôle spécifiques permettent la gestion de ce mécanisme de CRC :

- **WCRC**[38] : l'insertion de ce caractère indique au module HSL qu'il doit insérer, à la suite de ce caractère, un mot de CRC. Ce caractère provoque également la remise à zero du registre qui sert de base au calcul du CRC ;
- **RAZCRC**[39] : le module HSL calcule un CRC, à partir de tous les caractères transitant sur le réseau. La présence dans le flux de données de ce caractère provoque la remise à zero du registre servant au calcul du CRC. Ce caractère est nécessaire car il permet d'éviter qu'une fraction d'un paquet transmis soit pris en compte dans le calcul du CRC. Cette caractéristique est indispensable lorsque le routeur RCUBE est utilisé pour effectuer un routage explicite : dans ce mode de fonctionnement, chaque routeur utilise le premier caractère contenu dans l'en-tête pour déterminer quel port de sortie doit emprunter le paquet. Ce mécanisme permet d'imposer un cheminement spécifique aux paquets à l'intérieur d'un réseau de routeur RCUBE. La conséquence immédiate de ce mode de fonctionnement est qu'un en-tête ne contient pas le même nombre de caractères lors de son émission et lors de sa réception. Comme il est impossible calculer un CRC portant sur cette fraction de l'en-tête, les caractères servant au routage sont suivis par un caractère RAZCRC indiquant que le calcul du CRC doit redémarrer uniquement pour le reste de l'en-tête ;

[38] Write CRC.

[39] Remise A Zero du CRC.

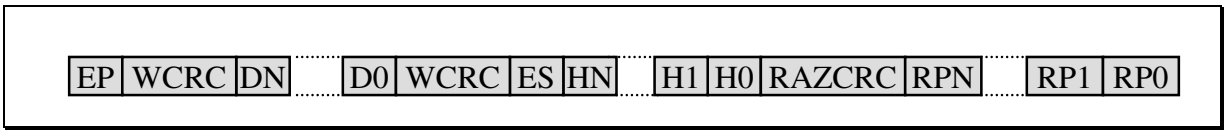


Figure 5.9 Utilisation du caractère RAZCRC

- **EP** : ce caractère marque la fin d'un paquet et a le même effet que le caractère RAZCRC sur le calcul du CRC. Il provoque la remise à zéro du registre utilisé pour le calcul du CRC.

La gestion des CRC est intégralement réalisée par le module HSL. En émission, il calcule et il insère les mots de CRC. En réception, il extrait des paquets et vérifie la validité des mots de CRC qui ont été transmis.

La capacité de la fifo contenant les microinstructions est de 16 cases. La capacité de la fifo destinée à recevoir les en-têtes des paquets est de 32 mots de 32 bits. Ces capacités permettent au microcontrôleur de commander plusieurs constructions de paquets simultanément. Le découpage en paquet est intégralement commandé par le microcontrôleur. Il est indépendant de la longueur des rafales en lecture sur le bus PCI, ainsi que de l'engorgement du réseau. Le microcontrôleur peut rassembler, dans un même paquet, les données provenant de plusieurs zones mémoires correspondant à plusieurs ordres donnés à l'opérateur DMAR. Cette caractéristique lui permet d'effectuer du « gathering ». Il peut également effectuer du « scattering », en décomposant les données contenues dans une zone mémoire correspondant à un ordre de DMA en plusieurs paquets transmis sur le réseau. Le découpage d'un message en plusieurs petits paquets permet de fluidifier le trafic sur le réseau [REIBALDI98].

5.3.5. PX

L'opérateur PX est chargé de traiter les paquets provenant du réseau. Il réalise l'interface entre le réseau, le microcontrôleur et l'opérateur DMAW. Deux exemplaires de cet opérateur sont associés aux deux chemins de données correspondant aux deux liaisons HSL.

Les objectifs qui ont guidé l'architecture de cette opérateur sont :

- la souplesse : cet opérateur doit être capable d'interpréter différents formats de paquets (ceux qui sont générés par l'opérateur PB et, dans une certaine mesure, d'autres formats de paquets et dans ce cas sans garantie de performance) ;
- l'autonomie : PX doit être capable d'extraire les données provenant du réseau sans nécessiter l'intervention immédiate du microcontrôleur.

Cet opérateur assure le découpage des paquets provenant du réseau :

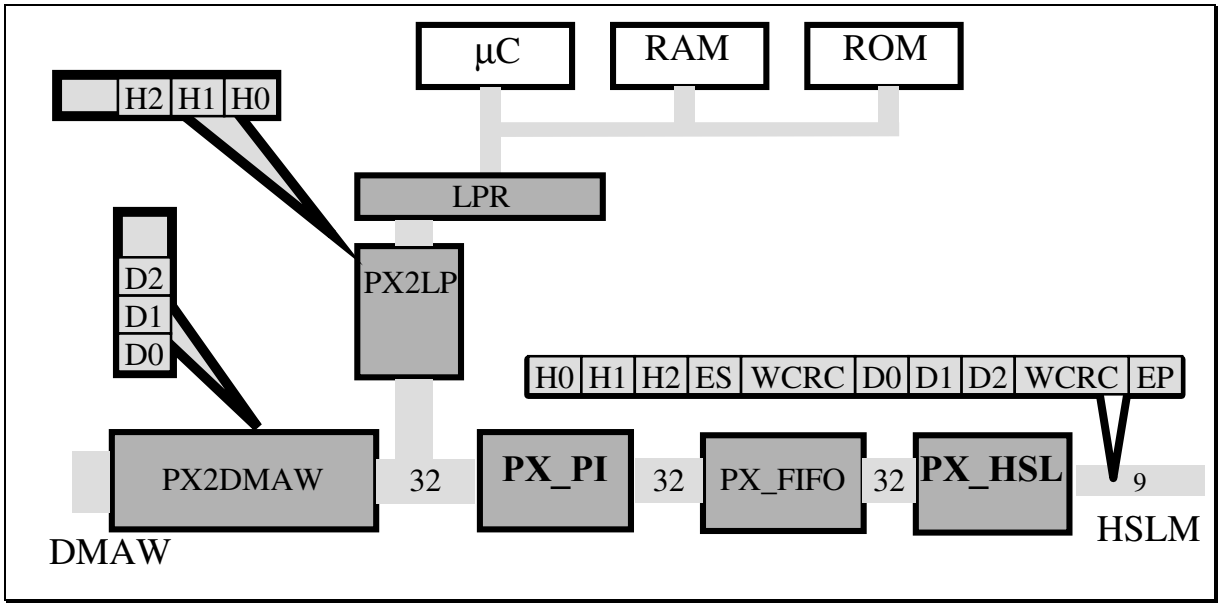


Figure 5.10 Détail du "Packet Extractor"

A chaque section correspond une destination spécifique pour les données contenues dans cette section. Par convention, la première section d'un paquet est à destination du microcontrôleur. Cette section correspond généralement à l'en-tête du paquet et il est naturel que cet en-tête soit interprété par le microcontrôleur.

La seconde section, celle qui suit le premier caractère ES, est à destination de l'opérateur DMAW. Il s'agit généralement des données contenues dans le paquet. A chaque nouveau caractère ES reçu correspond un changement de destination pour les données (alternativement le microcontrôleur et l'opérateur DMAW).

La réception d'un caractère EP indique la fin d'un paquet et donc les données qui suivent ce caractères correspondent nécessairement à l'en-tête d'un nouveau paquet, et elles sont destinées à être interprétées par le microcontrôleur.

Lorsque le « Packet Extractor » a reçu l'intégralité d'une section destinée au microcontrôleur, il l'en informe en positionnant à 1 le bit « Header Extracted » du registre général de status de NOÉ.

PX dispose d'une fifo d'une capacité de 32 mots de 32 bits pour stocker les sections des paquets destinées à être interprétées par le microcontrôleur. Suivant le format de paquet et plus particulièrement la taille des en-têtes, PX peut recevoir plusieurs paquets sans nécessiter l'intervention immédiate du microcontrôleur. Cette caractéristique dépend également de la

quantité de données contenues dans les paquets reçus, car la capacité des fifos réalisant l'interface entre PX et DMAW est limitée à 1.5 Ko.

Lorsque ces fifos sont saturées, l'opérateur PX ne peut plus extraire les données du réseau. Le mécanisme de contrôle de flux permet de rétropropager cette contention en évitant toute perte de donnée.

5.3.6. ITCTL

L'opérateur ITCTL regroupe tous les mécanismes permettant la notification au microcontrôleur, notamment le registre général de status :

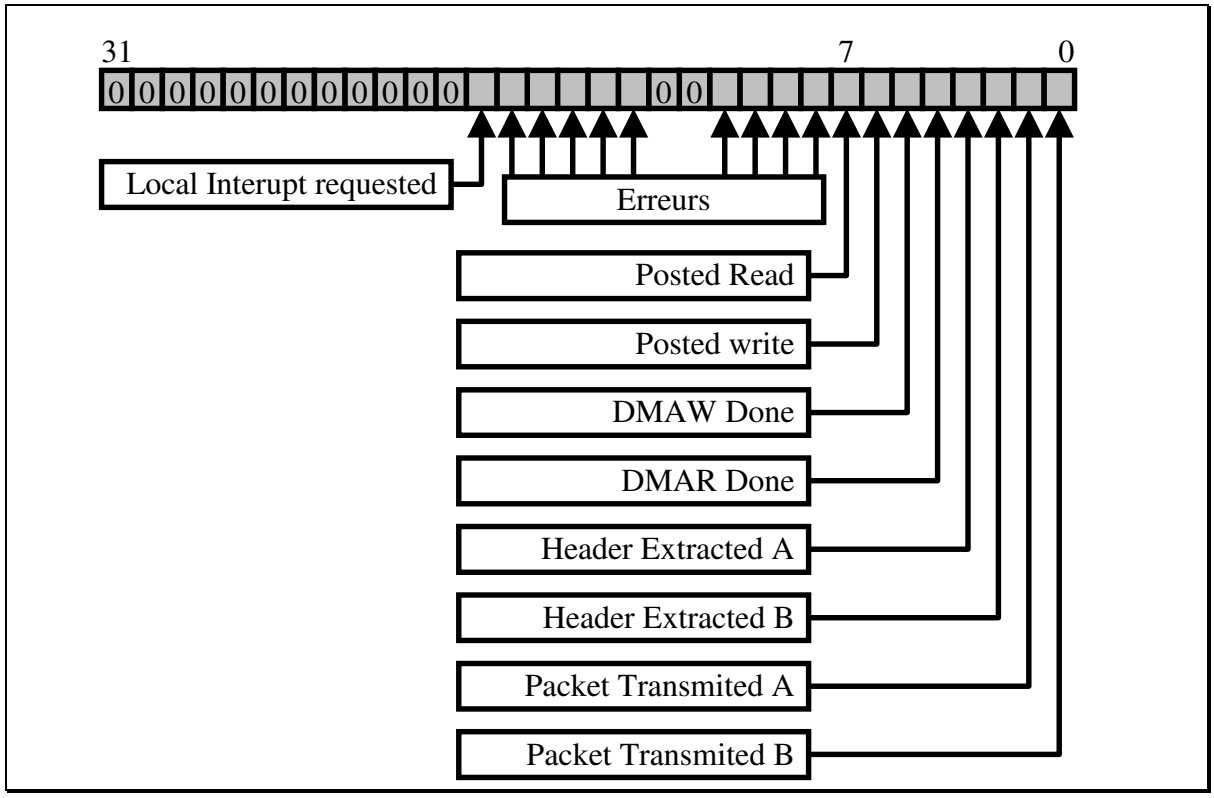


Figure 5.11 Le registre général de status.

Un registre servant au masquage des interruptions, identique à celui-ci permet de déclencher sélectivement un interruption destinée au microcontrôleur.

Dans l'objectif d'accélérer l'interprétation du registre de status par le microcontrôleur, il existe un registre de status codé qui utilise un arbre binaire de priorité pour n'indiquer que le numéro du bit de status actif le plus prioritaire :

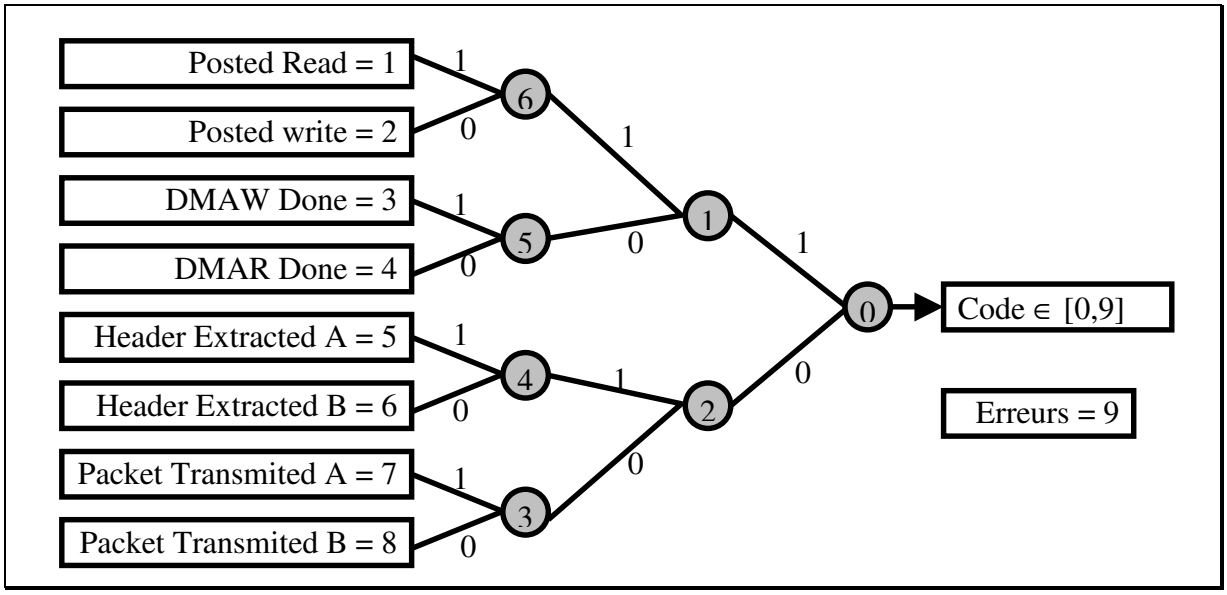


Figure 5.12 Codage du status.

Un registre de priorité contenu dans ITCTL permet au microcontrôleur de définir un ordre de priorité de prise en compte des différentes sources d'événements. A chaque nœud de cet arbre est associé un bit du registre de priorité : par exemple, si le bit numéro 3 est égal à 1 l'événement « Packet Transmited A » est prioritaire sur « Packet Transmited B ». Si les deux événements interviennent simultanément, c'est « Packet Transmited A » qui sera pris en compte en premier par le microcontrôleur.

Deux valeurs du code ont un sens particulier :

- **0** : aucune source d'événements n'est active (le registre général de status est égal à 0) ;
- **1** : une erreur a été signalé.

Un registre identique au registre de status permet de masquer une à une les sources d'événements pour le codage du status.

Certains opérateurs accumulent plusieurs notifications d'un même type : terminaison de plusieurs ordres, réception de plusieurs en-têtes de paquets. Ils utilisent un compteur pour mémoriser le nombre de notifications qu'ils ont à communiquer au microcontrôleur. Quand une notification doit intervenir, ils incrémentent le compteur correspondant.

Quand le microcontrôleur veut indiquer à un opérateur qu'il a pris en compte la notification qui lui était adressée, il écrit un 1 dans le bit correspondant du registre général de status. Cette écriture a pour effet de décrémenter le compteur associé.

Ce mécanisme permet de traiter une par une les notifications sans risque d'en oublier.

5.3.7. HSLM

C'est l'unique opérateur dont le fonctionnement est spécifique à l'utilisation du réseau HSL. Il assure les deux principales fonctions :

- la gestion de la sécurité des données transmises (cet opérateur assure le calcul, l'insertion et la vérification des caractères de CRC) ;
- la sérialisation et la désérialisation des données transmises sur le lien HSL ;
- la notification des erreurs.

Cet opérateur peut être configuré à l'aide d'un ensemble de registres qu'il rend visible pour le microcontrôleur.

5.4. Utilisation de NOÉ pour les transferts de paquets

Un ensemble d'actions est nécessaire pour permettre les transferts de paquets :

5.4.1. Emission

L'émission d'un paquet contenant des données provenant de la mémoire se décompose en trois actions distinctes pour le microcontrôleur :

- commander l'opérateur DMAR pour la zone de mémoire concernée en indiquant comme destination la fifo reliant l'opérateur DMAR à un des opérateurs PB ;
- écrire, dans la fifo destinée au PB concerné, l'en-tête du paquet ;
- écrire dans la fifo de commande du PB concerné, la séquence de microinstructions correspondant à la description du message à transmettre.

Il n'est pas nécessaire de respecter une séquence particulière pour l'enchaînement de ces trois actions. Les deux opérateurs se synchronisent en utilisant leurs fifos respectives. Si le microcontrôleur commande le PB en premier, il permettra au contrôleur de réseau récepteur d'anticiper le décodage de l'en-tête sur la réception des données et, donc, de diminuer la latence. Mais cependant, il risque d'accroître l'engorgement du réseau en « tenant le vers par la queue ». La meilleure stratégie pourra être déterminée par l'expérimentation des deux alternatives.

5.4.2. Réception

Deux opérations doivent être effectuées par le microcontrôleur, pour recevoir un paquet contenant des données et dont il ne connaît pas l'adresse de destination dans la mémoire de la station de travail :

- lire l'en-tête qui a été extraite par le « Packet Extractor » et l'interpréter ;
- commander l'opérateur DMAW en lui indiquant ou doivent être écrites les données en mémoire.

Comme pour l'émission, les opérateurs mis en jeu par une réception ont la possibilité de se synchroniser entre eux, en utilisant les fifos qui les relient. Il est possible d'envisager de commander l'opérateur de DMA en écriture avant d'avoir reçu l'en-tête du paquet si le microcontrôleur sait au préalable où il doit déposer les données dans la mémoire de la station de travail.

5.4.3. Conclusion

Comme le montre ces exemples de transferts de paquets, les actions que doit réaliser le microcontrôleur sont peu nombreuses et peu complexes. Cela lui permet de se consacrer à la mise en œuvre d'un protocole de plus haut niveau et donc de décharger le processeur de la station de travail.

5.5. Architecture interne

Du fait de ces trois interfaces, le composant NOÉ contient trois domaines d'horloges. Les frontières, correspondant à ces différents domaines d'horloge, nous ont été utiles pour découper le circuit en trois modules distincts permettant ainsi de paralléliser les développements :

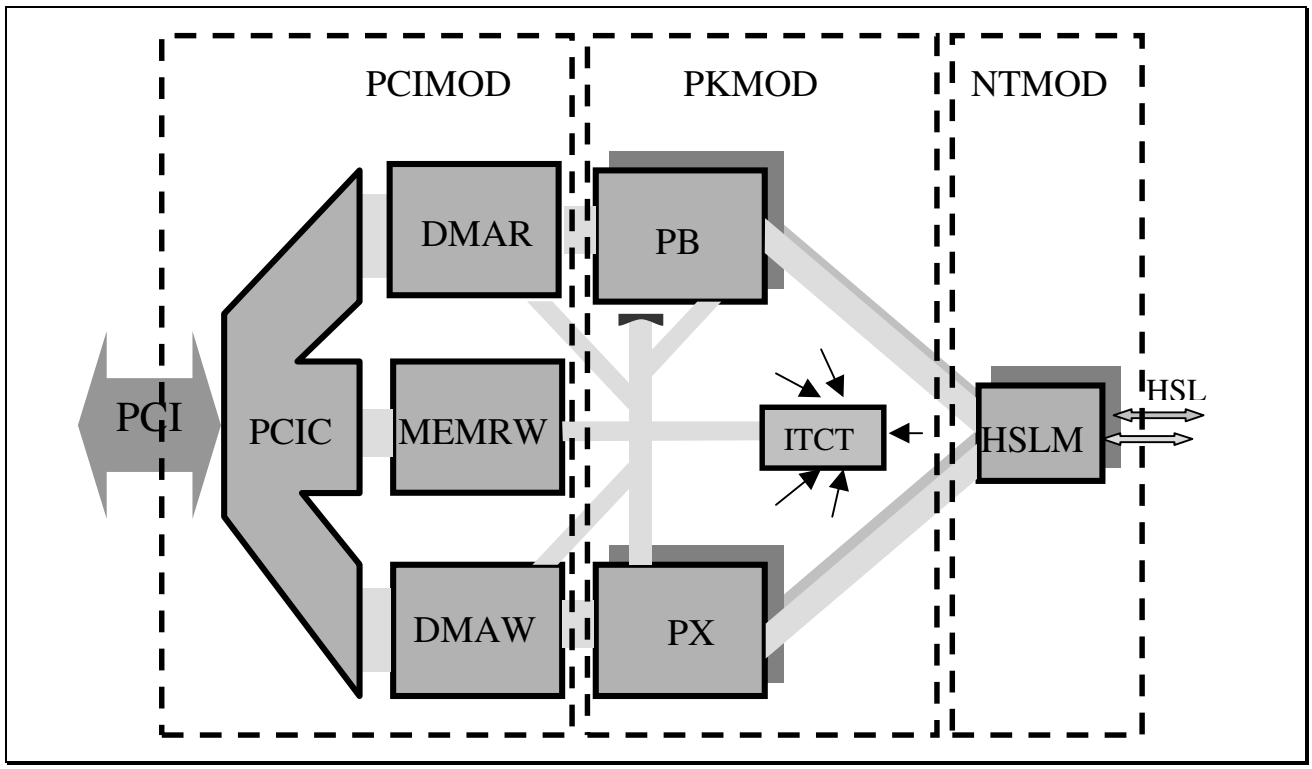


Figure 5.13 Le découpage de NOÉ en modules.

Chaque module correspond plus particulièrement à l'utilisation d'un signal d'horloge et réalise un certain type de fonction :

- **PCIMOD** : ce module regroupe les opérateurs qui possèdent une interface avec le bus PCI : DMAW, DMAR et MEMRW, l'horloge provenant du bus PCI est la plus répandue dans ce module ;
- **PKMOD** : ce module regroupe les opérateurs réalisant la gestion des paquets :PB et PX ainsi que ITCTL qui sert à la notification, l'horloge utilisée est celle générée par l'interface mémoire du microcontrôleur ;
- **NTMOD** : ce module contient les opérateurs propres au réseau HSL : HSLM, ils assurent la gestion des caractères de CRC et assurent la sérialisation et la désérialisation des données transmises sur le réseau. C'est le module le plus contraint en fréquence car il utilise l'horloge HSL qui correspond au débit mot (9 bits) du réseau HSL (débit bit HSL divisé par 12).

5.5.1. Résultats

Le développement du circuit NOÉ a nécessité 6 hommes*ans réparti sur 3 années, il a été envoyé en fabrication en juillet 1999 et nous avons reçu en novembre 1999 5 exemplaires mis en boîtier de ce circuit et 10 exemplaires nus.

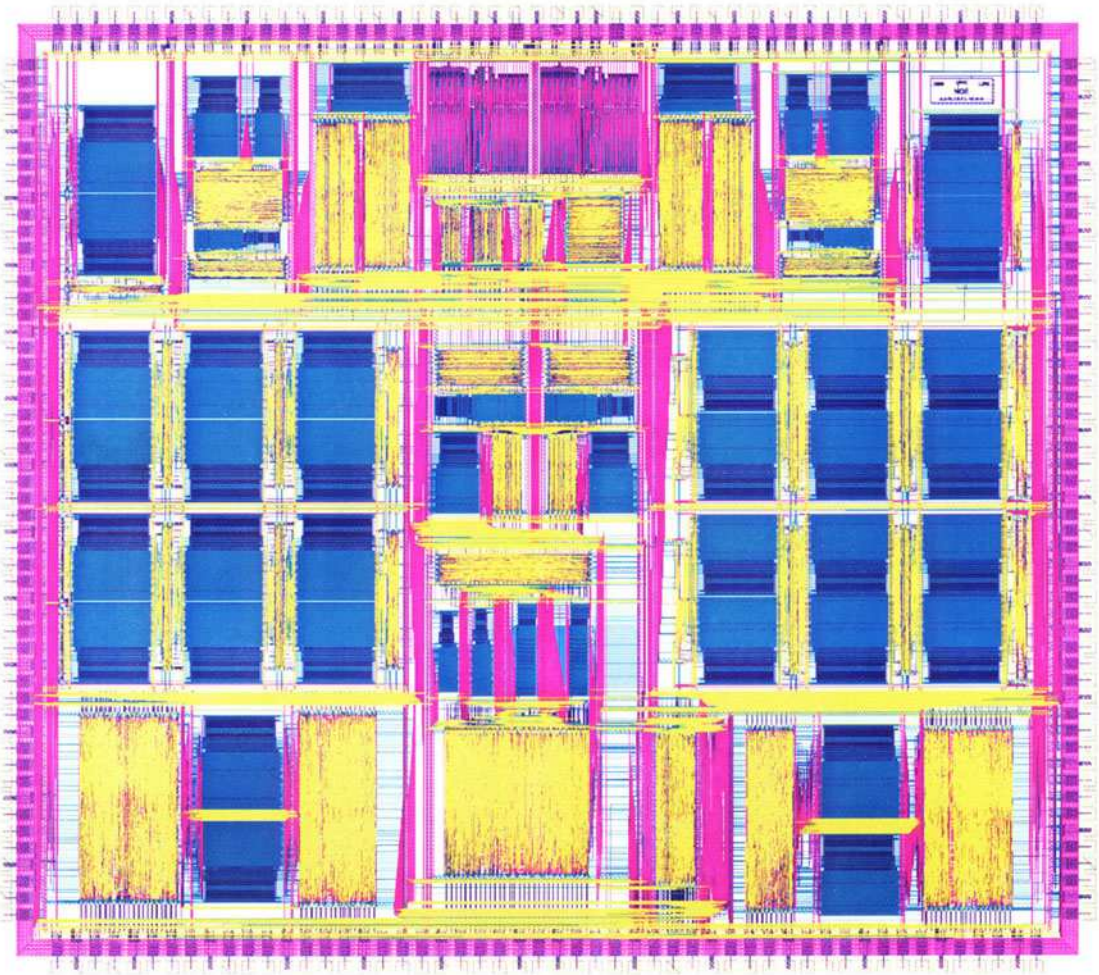


Figure 5.14 Vue physique du circuit NOÉ

Ce circuit contient 1.3 Mtransistors et sa surface est égale à 50mm². Il a été réalisé en technologie CMOS 0.25 µm par ST[40].

Le circuit NOÉ comporte 208 broches et il s'insère dans un boîtier de type QFP[41].

L'analyse temporelle de ce circuit nous fournit trois fréquences de fonctionnement pour les trois domaines d'horloges de ce circuit :

[40] SGS Thomson.

[41] Quad Flat Pad.

- PCIMOD : >66 Mhz ;
- PKMOD : > 120 Mhz ;
- NTMOD : > 150 Mhz.

Faute de moyens suffisants, le circuit n'a pas pu encore être testé, mais le fait qu'il ai été fabriqué démontre pleinement la faisabilité de cette architecture.

5.6. Conclusion

5.6.1. L'architecture

La conclusion de la problématique nous amenait à définir quelles devaient être les caractéristiques d'un contrôleur de réseau adapté à une utilisation de type « grappe de PC ». Nous allons passer en revu l'ensemble de ces caractéristiques, et les confronter à l'architecture du contrôleur de réseau NOÉ :

- il doit pouvoir être connecté au bus d'entré-sortie le plus performant du moment,
  - ⇒ NOÉ peut être connecté au bus PCI 32/64 bits, 33/66Mhz ;
- il doit permettre de communiquer avec un réseau à haut débit, faible latence et grande extensibilité,
  - ⇒ NOÉ dispose de 2 liens HSL, 4*125 Mo/s et 150 ns de latence par routeur ;
- il doit être directement accessible depuis les processus applicatifs, sans nécessiter d'appels au système d'exploitation,
  - ⇒ NOÉ est accessible par des écritures et des lectures mémoires ;
- il doit pouvoir être commandé simultanément par plusieurs processus ou thread,
  - ⇒ la particularité de l'architecture NOÉ est que les accès mémoire dont NOÉ est cible sont interprétés par le microcontrôleur. Cette caractéristique fait qu'il est possible pour le composant NOÉ de répondre à une très vaste plage d'adressage et ainsi offrir à chaque processus un espace mémoire qui lui est propre pour commander le contrôleur de réseau (se reporter au chapitre présentant RWU pour plus de détails) ;
- il doit supporter un mode de fonctionnement de type « wormhole », pour éviter les recopies,
  - ⇒ les données transitent par des fifo entre les opérateurs ce qui évite toute recopie inutile ;

- il doit être facilement programmable, pour permettre les adaptations et les expérimentations de protocoles,

  ⇒ comme nous l'avons vu, le protocole est mis en œuvre par un microcontrôleur. Le niveau de commande et l'autonomie des opérateurs permet de découpler le fonctionnement du microcontrôleur de celui du composant NOÉ. Ce découplage procure la souplesse nécessaire à la programmation de protocoles expérimentaux.

D'une façon générale, l'architecture du contrôleur de réseau NOÉ est bien adapté à une utilisation dans un contexte de machines parallèles du type « grappe de PC ».

5.6.2. Le composant NOÉ

Le développement d'une carte NOE-HSL, permettant de vérifier le bon fonctionnement du contrôleur NOÉ, n'a pu être réalisé dans le cadre de cette thèse.

De plus, l'expérience du contrôleur PCI-DDC montre que le test et la mise au point d'un circuit VLSI complexe demandent un effort considérable, hors de portée pour un laboratoire universitaire sans un fort soutien industriel.

Il s'ouvre néanmoins aujourd'hui une nouvelle perspective pour l'architecture du contrôleur d'interface NOÉ. Cette perspective est basée sur l'utilisation d'un composant à logique programmable. Ce qui était impossible, il y a deux ans, le devient aujourd'hui avec l'apparition de composants supportant l'équivalent de 400 000 portes logiques. Un projet a commencé, il vise l'intégration du composant NOÉ et de son microcontrôleur sur un unique composant programmable de type FPGA[42]. Cette approche devrait considérablement simplifier le développement ainsi que la mise au point.

[42] Field Programable Gate Area.

CHAPITRE 6

Le micronoyau BEE

Le chapitre précédent nous a permis de présenter l'architecture du contrôleur de réseau NOÉ. Un des principes de cette architecture est que la gestion du protocole est réalisée par un microcontrôleur programmable. Dans ce chapitre, nous présentons le programme qui s'exécute sur ce microcontrôleur. C'est une question qui ne doit en aucun cas être négligée, car les performances du réseau en dépendent directement. Après une courte présentation des solutions utilisées par les autres contrôleurs réseaux utilisant des microcontrôleurs, nous terminerons par la présentation du micronoyau BEE[43], lequel a été développé pour tirer parti de l'architecture NOÉ.

6.1. Le problème

Nous allons commencer par identifier quelles sont les sources d'événements produites par le composant NOÉ et qui nécessitent la réaction du microcontrôleur. Nous pouvons identifier trois sources :

- l'arrivée d'un en-tête de paquet en provenance du lien A ;
- l'arrivée d'un en-tête de paquet en provenance du lien B ;
- une écriture ou une lecture PCI dont NOÉ est la cible.

Ces trois événements étant dus à trois opérateurs matériels indépendants, ils peuvent survenir simultanément. Ces événements correspondent à des sources extérieures ; le microcontrôleur n'en est pas l'initiateur. Par ailleurs, NOÉ peut exécuter 4 activités parallèles :

- émission d'un paquet sur le chemin de donnée A ;
- émission d'un paquet sur le chemin de donnée B ;

[43] Best Effort Executive

- réception d'un paquet sur le chemin de donnée A ;
- réception d'un paquet sur le chemin de donnée B.

Ces activités partagent l'accès au bus PCI et utilisent des opérateurs spécifiques indépendants les uns des autres. Il est donc nécessaire que la structure du programme qui s'exécute sur le microcontrôleur tienne compte de ce parallélisme. Tous les contrôleurs de protocoles ne gèrent pas deux liaisons, mais ils gèrent tous une émission et une réception qui peuvent intervenir simultanément. Nous allons donc étudier les solutions qui sont proposées.

6.2. Les solutions existantes

Nous allons présenter, succinctement, les solutions utilisées par deux architectures réseaux, pour lesquelles, la mise en œuvre du protocole est assurée par un processeur.

6.2.1. Le contrôleur gigabit Ethernet Alteon

Ce contrôleur de réseau [Alt99] assure l'interface entre un bus PCI et une liaison Ethernet fonctionnant à 1 Gbit/s.

Ces objectifs sont proches des nôtres : décharger le processeur de la station de travail du traitement du protocole réseau. Le programme qui s'exécute sur la carte gère lui-même le protocole TCP/UDP.

Peu d'informations sont disponibles concernant son architecture interne. Il utilise deux mémoires embarquées distinctes sur la carte, pour stocker les données qui sont en transit entre les gestionnaires de paquets et les opérateurs de DMA PCI. Une mémoire est utilisée pour les réceptions, une pour les émissions. Concernant la gestion simultanée des activités parallèles d'émissions et de réceptions, le problème est résolu par l'utilisation de deux processeurs RISC embarqués. Un processeur gère les émissions et l'autre les réceptions.

Ce contrôleur réseau pourrait constituer une base intéressante pour bâtir une machine parallèle du type « grappe de PC », mais son architecture est « fermée » et optimisée pour les protocoles classiques de LAN. Or, nous avons vu qu'ils ne constituaient pas une bonne base pour le développement d'applications parallèles.

6.2.2. Le LANAI de la carte myrinet

Ce processeur a déjà été présenté dans l'état de l'art. Nous avons vu que le processeur embarqué intégré au LANAI disposait de deux contextes d'exécutions :

- un contexte dit « système » destiné au traitement des interruptions. Ce contexte est utile pour répondre rapidement aux événements matériels que constitue typiquement la réception d'un paquet ;
- un contexte dit « utilisateur » pour les autres traitements tels que l'émission d'un paquet.

Comme pour le contrôleur gigabit Ethernet, le contrôleur myrinet fournit un support matériel à la gestion des activités parallèles.

6.3. Présentation de BEE

Notre architecture fait appel à l'utilisation d'un unique processeur RISC standard. Nous allons donc devoir gérer explicitement le parallélisme des traitements. Nos réflexions nous ont amené à définir le micronoyau BEE. C'est un micronoyau au sens qu'il offre les quatre services minimaux [Tan] :

- un mécanisme de communication inter-tâches. Toutes les tâches possèdent le même espace d'adressage, elles peuvent donc communiquer par des mécanismes de mémoire partagée ;
- la gestion de la mémoire : chaque tâche dispose de sa propre pile ;
- l'ordonnancement des tâches : BEE permet d'initialiser des tâches et offre un mécanisme d'ordonnancement adapté aux contraintes matérielles ;
- des entrées/sorties bas niveaux, les tâches disposent de deux fonctions *noe_read()* et *noe_write()* qui permettent d'accéder au composant NOE.

BEE n'utilise qu'un unique mode d'exécution, il peut être défini par ses quatre principales caractéristiques :

- **Multitâche** : caractéristique nécessaire pour répondre au problème posé par la gestion des quatre activités simultanées du composant NOÉ ;
- **Minimal** : l'unique objectif du micronoyau est le contrôle efficace du composant NOÉ (la légèreté du noyau est synonyme de performance) ;
- **Non préemptif** : contrairement au gestionnaire de paquets Myrinet, l'opérateur de réception de paquets de NOÉ n'impose pas que le microcontrôleur réagisse instantanément. Les tâches qui s'exécutent sur le microcontrôleur participent à la réalisation d'une même application (elles sont supposées coopérer entre elles). Pour cet ensemble de raisons, nous avons choisi de ne pas utiliser les interruptions pour la commutation de contexte, ce qui réduit la complexité de notre micronoyau ;

- **Monolithique** : BEE ainsi que les tâches qui réalisent la mise en œuvre du protocole constituent un unique programme. Il n'y pas de création dynamique de tâche en cours d'exécution. Le micronoyau n'offre pas non plus de service d'allocation de mémoire dynamique.

BEE ne peut pas être qualifié de « temps réel », car il nous est impossible de définir une borne maximale absolue pour l'exécution de certaines commandes par les opérateurs de NOÉ. BEE est qualifiable de « best effort », ce qui veut dire qu'il cherche à faire de son mieux.

6.3.1. Les Tâches

Le nombre de tâches est défini à la compilation. Les tâches sont supposées être « bonnes citoyennes ». Elles décident d'elles-mêmes de rendre la main au système. Toutes les tâches partagent le même espace d'adressage et il peut exister plusieurs instances d'une même tâche (même code).

Chaque tâche est caractérisée par l'état dans lequel elle se trouve. Une tâche peut prendre trois états différents :

- **Running** : une seule tâche se trouve dans cet état à un instant donné (c'est la tâche qui est en cours d'exécution sur le microcontrôleur) ;
- **Ready** : cette tâche est prête à être exécutée sur le processeur ;
- **Waiting** : la tâche est en attente sur un sémaphore.

Chaque tâche dispose de sa propre pile. Si elles utilisent exclusivement des variables automatiques (locales), plusieurs tâches correspondant à un même code peuvent s'exécuter simultanément. Plusieurs tâches peuvent également partager des structures de données globales et dans ce cas, elles doivent utiliser des sémaphores avant d'y accéder.

6.3.2. Les sémaphores

Le micronoyau BEE offre un certain nombre de sémaphores qui permettent aux tâches de partager l'accès aux opérateurs matériels. Chaque sémaphore correspond à une ressource. Il est constitué de deux éléments :

- un compteur dont la valeur, si elle est positive ou nulle, indique le nombre de ressources disponibles. Si sa valeur est négative, elle indique le nombre de tâches qui sont en attente sur ce sémaphore ;
- une liste contenant les numéros des tâches qui sont en attente sur ce sémaphore.

Deux primitives sont offertes par le micronoyau BEE, permettant ainsi l'utilisation et la gestion de ces sémaphores :

- **P**[44] : cette primitive permet de faire l'acquisition de la ressource associée. Quand une tâche fait un P sur un sémaphore, le compteur associé au sémaphore est décrémenté. Si la nouvelle valeur du compteur est strictement négative, la tâche est mise en attente sur ce sémaphore (elle passe dans l'état « Waiting ») et la main est rendue au micronoyau. Si la nouvelle valeur du compteur est positive ou nulle, l'exécution de la tâche peut continuer et celle-ci peut utiliser la ressource correspondant au sémaphore ;
- **V**[45] : cette primitive sert à indiquer qu'une ressource est disponible. Quand un V est effectué pour un certain sémaphore, le compteur associé est incrémenté. Si la nouvelle valeur contenue dans le compteur est négative ou nulle, la tâche, qui se trouvait en tête de la liste des tâches en attente sur ce sémaphore, passe dans l'état « Ready » et peut être élue par l'ordonnanceur.

BEE n'étant pas préemptif, les tâches doivent rendre d'elles-mêmes la main. A chaque fois qu'elles veulent utiliser un opérateur, elles doivent faire un P sur le sémaphore correspondant. L'appel à cette primitive peut provoquer l'ordonnancement, si la ressource correspondante n'est pas disponible. Les différents appels à P constituent des points de synchronisation pour les tâches, comme le montre le schéma suivant.

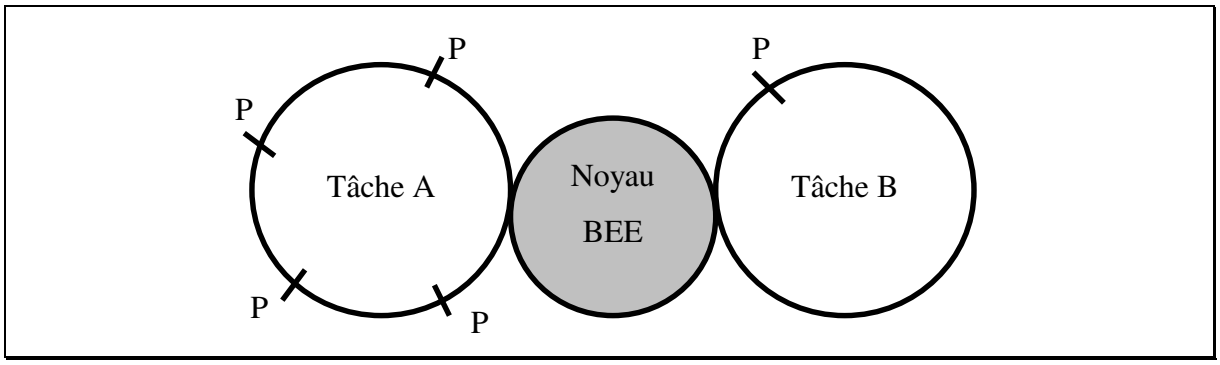


Figure 6.1 Synchronisation des tâches.

Comme nous le voyons sur ce schéma, la tâche B ne fait qu'une seule fois appel à P et monopolise donc longtemps le processeur, ce qui est néfaste pour le bon fonctionnement de l'intégralité du contrôleur réseau.

[44] Puis-je.

[45] Vas-y.

Pour remédier à ce problème, nous avons décidé d'associer un sémaphore au microcontrôleur (par convention, le numéro 0). De cette manière une tâche bonne citoyenne peut regarder si d'autres tâches sont en attente du processeur, en faisant un P sur le sémaphore 0.

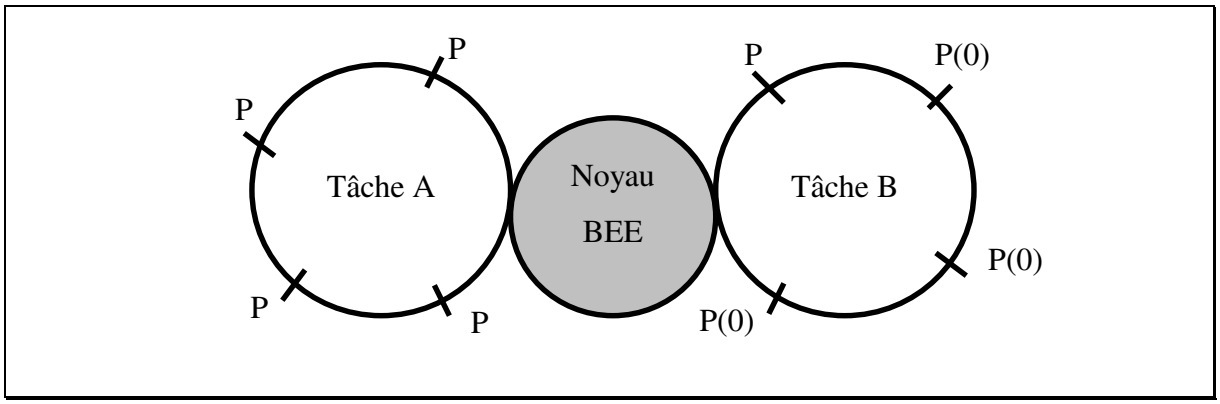


Figure 6.2 Synchronisation avec P(0).

Un P sur le sémaphore 0 place la tâche courante dans la file d'attente associée à ce sémaphore et provoque un réordonnancement. La file d'attente associée au sémaphore 0 contient les tâches prêtes à être exécutées, à chaque itération de l'ordonnanceur, le processeur est attribué à la tâche qui se trouve en-tête de la file. Chaque ordonnancement correspond implicitement à un V sur le sémaphore 0.

L'action de primitives P et V nous permet de définir le diagramme des états que peut prendre une tâche.

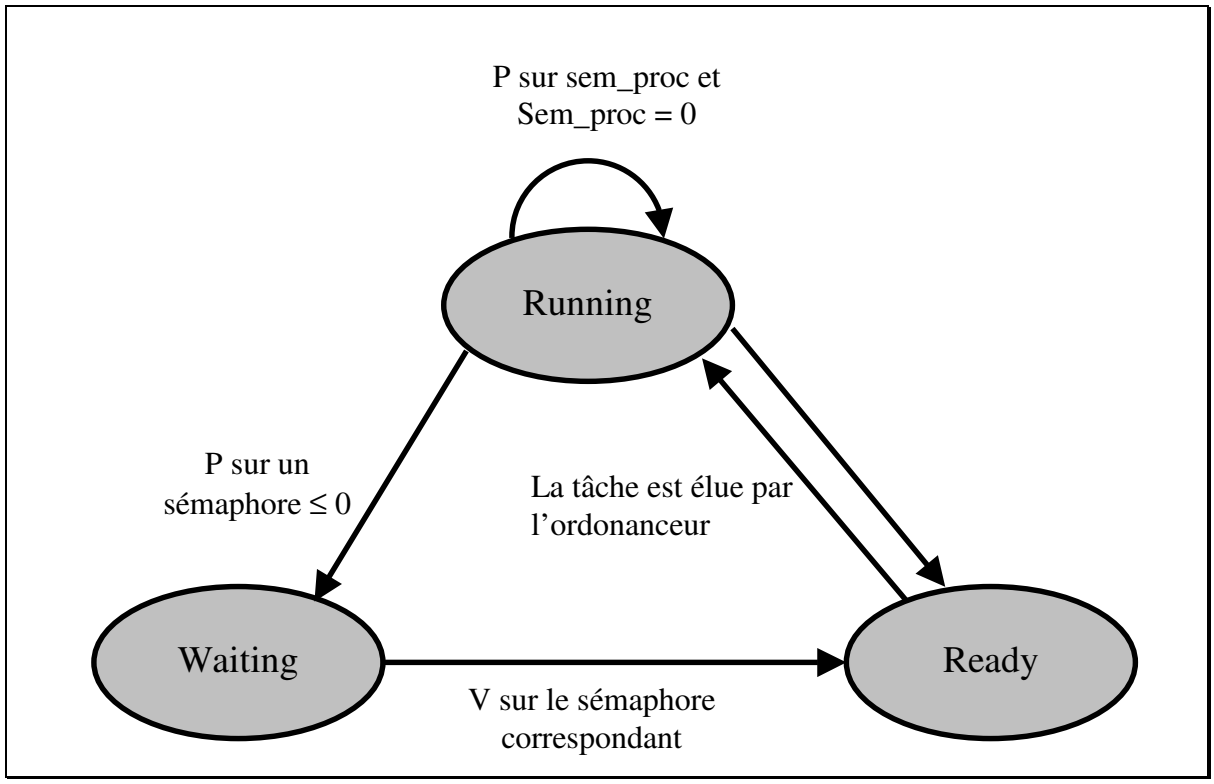


Figure 6.3 Fonctionnement du "Read Buffer"

L'ensemble des sémaphores permettant d'accéder aux opérateurs de NOÉ et au microcontrôleur sont dits « matériels ». Les tâches peuvent utiliser d'autres sémaphores dits « logiciels », pour se synchroniser entre elles.

6.3.3. La commutation de tâche

Par définition, la commutation de tâche consiste à suspendre l'exécution de la tâche en cours, sauvegarder son contexte, rétablir le contexte de la nouvelle tâche et déclencher son exécution. Le micronoyau BEE n'étant pas préemptif, la commutation de tâche s'effectue à l'initiative de la tâche en cours d'exécution. Toutes les tâches partageant le même espace d'adressage, et le système n'offrant que les services minimaux (pas de descripteurs de fichiers), le contexte d'une tâche se réduit à :

- un compteur ordinal ;
- des registres de travail ;
- une pile.

Dans l'objectif de simplifier la commutation de tâche, nous avons choisi de sauvegarder le compteur ordinal et les registres de travail sur la pile associée à la tâche. Par conséquent, du point de vue du noyau, le contexte d'une tâche se réduit à :

- une pile ;
- un pointeur de sommet de pile.

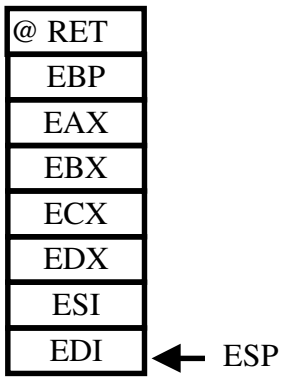
La commutation de tâche se fait par l'appel à une fonction particulière « switch task », dont nous fournissons une version pour le processeur i386 :

```
void switch_task(int next_task)
    {
```

Le prélude de cette fonction qui est généré par le compilateur C, sert à sauvegarder le compteur ordinal ainsi que le registre BP sur la pile.

```
asm ("pushl %eax");
asm ("pushl %ebx");
asm ("pushl %ecx");
asm ("pushl %edx");
asm ("pushl %esi");
asm ("pushl %edi");
```

Sauvegarde du contexte sur la pile




```
asm ("movl %esp,stack_pointer");
task[cur_task].sp = stack_pointer;

cur_task = next_task;

stack_pointer = task[cur_task].sp;
asm ("movl stack_pointer,%esp");
```

Permutation des pointeurs de pile :

- le pointeur de pile correspondant à cur_task est sauvegardé ;
- le pointeur de pile correspondant à next_task est restauré.

```
asm ("popl %edi");
asm ("popl %esi");
asm ("popl %edx");
asm ("popl %ecx");
asm ("popl %ebx");
asm ("popl %eax");
```

Restauration des registres de travail.

Le postlude de cette fonction permet de restaurer le registre EBP et de reprendre l'exécution de la nouvelle tâche où elle avait été suspendue.

```
}
```

Comme nous pouvons le constater, si cette fonction est appelée avec next_task égale à cur_task, son exécution n'aura aucun effet. Du fait de son utilisation dans un contexte particulier (mono-processeur sans interruption), cette fonction ne nécessite pas d'être exécutée en « section critique ». Dans l'éventualité de l'utilisation d'un microcontrôleur multiprocesseur, cette fonction devrait être modifiée, pour éviter l'utilisation de la variable globale *stack_pointer* qui n'est utilisée ici que pour simplifier l'interface entre le C et l'assembleur. Une fois compilée, cette fonction contient un total de 36 instructions assembleur. C'est la seule dont le code source contienne du langage assembleur. Ce micronoyau est donc aisément portable sur différentes architectures de microcontrôleurs (notamment les PowerPC).

6.3.4. L'ordonnanceur de tâche

Comme nous l'avons montré au cours de la présentation des opérateurs, il est nécessaire de garantir la séquentialité entre la commande des opérateurs et la consommation des données produites par cet opérateur. Deux tâches peuvent donner deux ordres consécutifs à l'opérateur DMAR, en indiquant la même fifo de destination. Si une tâche est la première à commander DMAR, elle doit être la première à lire les données dans la fifo destination (afin d'éviter qu'il y ait interversion entre les données destinées à la première et à la seconde tâche).

Pour respecter cette contrainte, l'ordonnanceur doit utiliser un mécanisme d'élection de type fifo. Le sémaphore 0 étant associé à la ressource microcontrôleur, l'ordonnanceur utilise la file associée au sémaphore 0 pour stocker les numéros des tâches prêtes à être exécutées par le microcontrôleur.

L'ordonnanceur prend la forme d'une boucle infinie. A chaque itération de cette boucle, il lit le registre général de status de NOÉ. En analysant les bits qui sont égaux à 1 de ce registre, il détecte quels opérateurs ont terminé les opérations qui leurs avaient été commandées. Muni de cette information, l'ordonnaneur fait appel à la primitive V pour mettre à jour les sémaphores associés à ces opérateurs. Lorsque tous les V ont été effectués, l'ordonnanceur attribue le processeur à la tâche qui se trouve en tête de la liste des tâches prêtes à être exécutées.

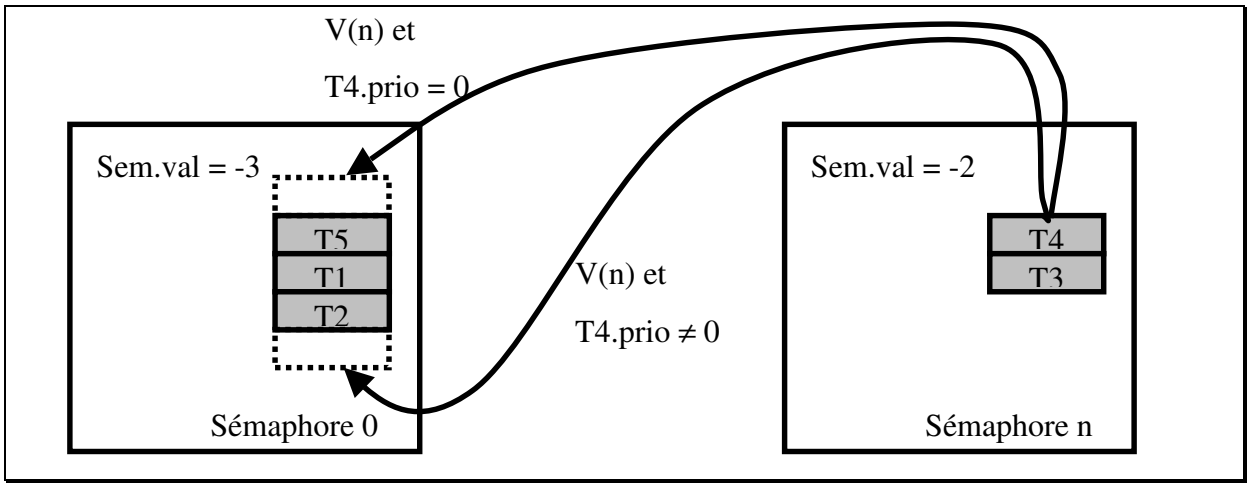


Figure 6.4 V effectués par l'ordonnanceur.

Une source d'événements doit être traitée en priorité sur les autres : « Posted Read » qui correspond à une lecture sur le bus PCI dont le « Read Buffer » de NOÉ est cible. Pour accélérer la réponse à cette requête, une tâche possédant une priorité supérieure aux autres tâches est dédiée à ce traitement. Lorsqu'un V est effectué sur le sémaphore « Posted Read », la tâche correspondante est placée en tête de la liste des tâches prêtes à être exécutées.

6.3.5. Une API

Pour rendre la tâche du programmeur chargé d'écrire le programme plus facile, et afin qu'il n'ait pas à se soucier du fonctionnement des opérateurs de NOE, nous avons développé quelques primitives.

Ces primitives se chargent d'effectuer des recopies entre la mémoire de la station de travail et celle associée au contrôleur réseau, ou facilitent l'envoi et la réception de paquets :

- *Host2local_memory* et *Local2host_memory* permettent les recopies entre la mémoire embarqué sur la carte d'interface et la mémoire hôte. Elles prennent comme paramètres les adresses dans la mémoire locale et la mémoire hôte, ainsi que la longueur des données à copier ;
- *Transmit_packet* déclenche l'accès en DMA pour lire les données et commande l'opérateur PB pour l'envoi du paquet. Cette fonction n'est pas spécifique à un protocole particulier, elle utilise juste un pointeur sur l'en-tête et la description des données à lire en mémoire hôte ;
- *Extract_header* permet de lire l'en-tête d'un paquet reçu par PX.

6.3.6. Un exemple de tâche

Nous allons présenter maintenant l'exemple d'une tâche de transmission pour un protocole élémentaire, comparable à celui mis en œuvre par le composant PCIDDC. Ce protocole est basé sur l'existence, dans la mémoire de la station de travail, d'un descripteur du message à transmettre. Ce descripteur contient l'adresse où se trouvent les données, la longueur des données ainsi que le numéro du nœud destinataire et l'adresse où doivent être écrites les données dans la mémoire du nœud destinataire.

```
#include "../Vnoe/noe_io.h"
#include "../Noe_api/noe_api.h"
#include "../Bee_kernel/bee_kernel.h"
#include "ddslr.h"


/*--------------------------------------------------------------------*/
/*--------------- Variables globales ---------------------------------*/
/*--------------------------------------------------------------------*/

unsigned long lpe_cur = LPE_BASE;
unsigned long lpe_new = LPE_BASE;

/*--------------------------------------------------------------------*/
/*--------------- TX_A -----------------------------------------------*/
/*--------------------------------------------------------------------*/

void tx_a()
    {

    lpe_entry_t lpe;
    header_t header;
    unsigned long i;
    unsigned long micro[4];

    micro[0] =   MIC_ES | ZERO | ZERO | CTL_CHAR |  HDL;
    micro[1] = MIC_WCRC | ZERO | ZERO | CTL_CHAR | ZERO;
    micro[3] =   MIC_EP |  NOS | ZERO | CTL_CHAR | ZERO;
/* les constantes ZERO ont pour unique objectif de faciliter la lecture */
/* des micro-instructions */
    while(1)
        {
        P(LPE_ENTRY);
        lpe_cur += 16;/* lpe_cur est paratagé par les 2 TX(A/B)*/
        host2local_memory( lpe_cur - 16, (unsigned long *) &lpe, 16);

        header.rphi = lpe.rphi;
        header.rplo = lpe.rplo;
        header.ml = lpe.pl;
        header.mi = lpe.mi;
        header.radr = lpe.radr;

        micro[2] = MIC_WCRC | ZERO | DMAR | CTL_CHAR | lpe.pl;

        transmit_packet((unsigned long *) &micro,
                        (unsigned long *) &header,
                        4, HDL, lpe.ladr, lpe.pl, 'a');
        }
    }
```

Comme le montre cet exemple, l'émission s'effectue en trois temps :

- lecture du descripteur dans la mémoire de la station de travail, en utilisant la fonction `host2local_memory` fournie par l'API ;
- construction de l'en-tête du paquet ;
- commande de l'émission du paquet en utilisant la fonction `transmit_packet` fournie par l'API.

6.4. Conclusion

Ce micronoyau permet de gérer efficacement le parallélisme contenu dans le composant NOÉ. Accompagné de son API, ce micronoyau rend abordable la programmation du microcontrôleur et satisfait donc pleinement les objectifs que nous nous étions fixés.

CHAPITRE 7

Les Performances

Les trois chapitres précédents nous ont permis de présenter un ensemble constitué du contrôleur de réseau NOÉ, d'un micronoyau BEE pour le microcontrôleur associé, et d'un protocole RWU sans appels systèmes. Pour évaluer les performances de cette architecture, nous avons développé la plate-forme de simulation VNOÉ que nous présentons dans ce chapitre.

7.1. Les objectifs

Trois principaux objectifs ont guidé la définition et la réalisation de cette plate-forme :

- permettre la mise au point du micronoyau BEE ;
- constituer une plate-forme pour la mise au point du micrologiciel dédié à la réalisation d'un protocole de communication particulier ;
- permettre l'évaluation des performances de l'architecture complète : contrôleur réseau, micronoyau et implémentation micrologicielle du protocole.

7.2. Architecture de la plate-forme VNOÉ

La plate-forme VNOÉ doit intégrer les trois composants de notre architecture :

- le modèle du composant NOÉ ;
- le micronoyau BEE ;
- Les tâches micro-logicielles réalisant le protocole de communication.

Nous avons choisi de rassembler ces trois éléments dans un unique programme constituant un unique processus UNIX. Pendant un temps, nous avons envisagé l'utilisation de threads pour modéliser le parallélisme de l'architecture. Cette direction a été rapidement abandonnée, car

elle ne nous permettait pas de mettre en œuvre facilement les mécanismes d'ordonnancement spécifiques du micronoyau BEE.

Nous avons choisi d'utiliser un PC sous LINUX comme environnement de développement. La mise au point de la plate-forme a été effectuée en trois temps :

- nous avons écrit la première version du micronoyau BEE, dans cet environnement ;
- nous avons écrit le modèle du composant NOÉ (ce modèle sera présenté par la suite) ;
- nous avons écrit les tâches nécessaires à la mise en œuvre du protocole DDSLR.

La mise au point du modèle du composant NOÉ s'est avérée plus complexe que nous ne l'avions envisagé initialement. Une difficulté centrale était la modélisation du temps : il fallait définir une représentation cohérente pour les temps nécessaires à l'exécution des tâches micro-logicielles par le microcontrôleur, et pour les temps correspondants à la réalisation des commandes par les opérateurs matériels de NOÉ.

7.2.1. Modélisation du composant NOÉ

L'interface entre le programme s'exécutant sur le microcontrôleur et le composant NOÉ fait appel à l'utilisation de deux uniques fonctions : *noe_read* et *noe_write*. Nous avons décidé de modéliser NOÉ à l'intérieur de ces deux fonctions :

- ce modèle doit être d'assez haut niveau et n'a pas besoin de décrire explicitement les valeurs des données transportées. Il doit uniquement manipuler les informations utilisées par le protocole et situées dans l'en-tête paquet. Pour ce qui concerne le corps des paquets, seule la taille nous intéresse ;
- la modélisation n'a pas besoin d'être précise au cycle près : la réalisation d'une commande par un opérateur matériel de NOE peut prendre quelques dizaines de cycles, et une précision de quelques cycles est suffisante pour nos besoins.

Nous faisons l'hypothèse que la progression du temps est assurée par l'exécution du programme sur le microcontrôleur. L'unité de temps est donc le cycle du microcontrôleur. Le modèle de simulation représentant les opérateurs matériels de NOE gère un échéancier et un temps simulé. A chaque appel des fonctions *noe_read* et *noe_write*, le temps simulé est mis à jour.

Dans une première approche, la fonction *noe_write* se comportait comme un aiguillage vers un ensemble de fonctions. A chaque opérateur correspondait une fonction dont le résultat était le temps nécessaire (en nombre de cycles du microcontrôleur) à la réalisation de l'opération

commandée. Une fois ce temps calculé, il suffisait de l'ajouter à la date courante pour obtenir la date à laquelle le bit correspondant du status serait mis à 1.

En analysant cette solution, nous nous sommes rapidement rendu compte que cette approximation du fonctionnement des opérateurs ne permettait pas de représenter leurs dépendances mutuelles. Par exemple, un constructeur de paquet n'a terminé d'envoyer un paquet que lorsqu'un opérateur de DMA en lecture lui a fourni les données nécessaires. Lors de l'appel à la fonction modélisant le constructeur de paquet, il est donc impossible de déterminer la date de terminaison de l'émission du paquet.

Ces constatations nous ont amené à considérer que les opérateurs devaient être modélisés comme des automates possédant des états, et qu'un mécanisme devait gérer la progression de ces automates.

Nous avons choisi de modéliser le composant NOÉ comme un ensemble d'opérateurs reliés par des fifos qui modélisent les dépendances de données entre ces opérateurs, sous le contrôle d'un mécanisme de simulation de type « event driven ».

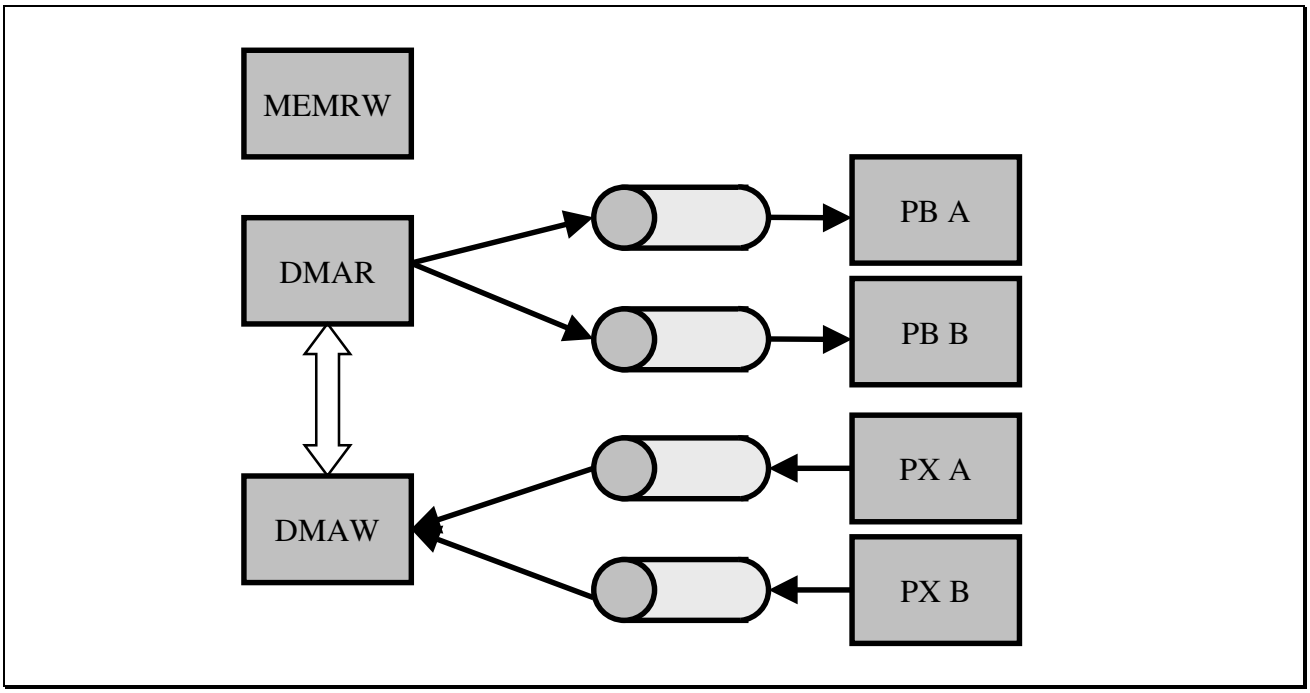


Figure 7.1 Opérateurs et dépendances.

Comme nous le voyons sur ce schéma, NOÉ peut être modélisé par 7 opérateurs et 4 fifos, lesquelles matérialisent les dépendances entre opérateurs. Il existe une autre dépendance qui lie les opérateurs de DMA : ils ne peuvent pas accéder simultanément au bus PCI. La

synchronisation entre les opérateurs de DMA utilise un sémaphore interne au simulateur modélisant la disponibilité du bus PCI. Le compteur associé à ce sémaphore peut prendre trois valeurs :

- **1** : le bus PCI est disponible ;
- **0** : le bus PCI est utilisé par un opérateur de DMA ;
- -1 : le bus PCI est utilisé par un opérateur de DMA et l'autre opérateur de DMA est en attente du bus PCI.

7.2.2. Modélisation des opérateurs

Chaque opérateur est modélisé comme un automate. Ces automates sont constitués d'une boucle infinie possédant des états stables. A chaque état stable correspond un point de synchronisation matériel : soit avec le microcontrôleur (attente d'un ordre), soit avec un autre opérateur (dépendance de donnée).

7.2.3. Modélisation des fifos

Nous modélisons deux types de fifos dans NOÉ :

- les fifos de contrôle contenant des données pertinentes pour le protocole destinées à être utilisées par le microcontrôleur ;
- les fifos de données qui modélisent les échanges de données entre les opérateurs (celles représentées sur le schéma).

A chaque fifo est associée une structure de donnée qui modélise son état. L'accès à ces fifos est assuré par 4 fonctions. Chacune correspond à un mode d'accès (lecture ou écriture) et à un type de fifo.

Lorsqu'un opérateur veut écrire dans une fifo destinée à un autre opérateur, il indique : le nombre d'octets à écrire ainsi que le débit qu'il est capable de fournir pour cette écriture. Suivant sa requête et l'état de la fifo, trois cas sont possibles :

- la capacité disponible de la fifo est supérieure à la quantité de données que l'opérateur veut y écrire. Dans ce cas, l'automate modélisant l'opérateur est mis en attente jusqu'à la date de fin d'écriture, qui ne dépend que de son débit propre ;
- la quantité de données à écrire est supérieure à la capacité disponible dans la fifo mais une requête en lecture par un autre opérateur est en cours. Dans ce cas, l'opérateur souhaitant écrire dans la fifo est mis en attente jusqu'à la fin de l'écriture. La date de fin d'écriture est

calculée en prenant en compte les deux débits correspondant aux deux requêtes d'écriture et de la lecture ;

- la requête ne peut pas être satisfaite : dans ce cas, l'opérateur est mis en attente d'une requête de lecture dans cette fifo.

L'accès en lecture à une fifo reliant deux opérateurs fonctionne suivant le même principe que l'accès en écriture.

L'accès en écriture aux fifos contenant des données pertinentes ne peut donner lieu qu'à deux comportements distincts :

- la fifo n'est pas pleine. Dans ce cas, la donnée est écrite dans la fifo et l'opérateur continue son exécution ;
- la fifo est pleine. L'opérateur est alors mis en attente d'une lecture dans cette fifo.

La lecture dans ces fifos suit les mêmes principes que l'écriture.

7.2.4. Mécanisme général de simulation

Le programme destiné à être implanté sur le microcontrôleur est directement exécuté par le processeur de la station de travail utilisé pour la simulation. L'instruction RTSC du processeur PENTIUM [P5] nous permet de lire le compteur de cycle du processeur. Ce temps sert de base pour la progression de la simulation.

Du point de vue du programme qui s'exécute sur le microcontrôleur, les fonctions *noe_read* et *noe_write*, sont non bloquantes et instantanées. Le déroulement temporel du programme sur le microcontrôleur est indépendant de son environnement matériel : il ne reçoit aucune interruption venant du composant NOÉ. De sa propre initiative et par l'intermédiaire de la fonction *noe_read*, le programme du microcontrôleur peut avoir accès aux registres du composant NOÉ et éventuellement modifier sa propre exécution.

Il est donc naturel de mesurer le temps en termes du déroulement séquentiel du code qui s'exécute sur le microcontrôleur.

L'appel des fonctions *noe_read* et *noe_write*, constitue un point de synchronisation entre le programme du microcontrôleur et le composant NOÉ. Durant l'exécution de ces fonctions, le comptage du temps est interrompu. A chaque appel de ces fonctions, on déclenche l'évaluation du simulateur du composant NOÉ. Un échéancier est utilisé pour superviser la progression des différents automates modélisant les différents opérateurs matériels du composant NOÉ. Comme c'est le déroulement du programme du microcontrôleur qui fait

progresser les dates, le temps simulé du matériel est toujours en retard par rapport au temps du microcontrôleur.

D'une façon générale, la progression du temps pour les différents éléments du système suit la séquence suivante :

- exécution du programme du microcontrôleur à partir du temps T0 ;
- appel à *noe_read* (ou *noe_write*) au temps T1 ;
- progression de l'échéancier du simulateur VNOÉ du temps T0 au temps T1 ;
- exécution du programme du microcontrôleur du temps T1 au temps T2 ;
- …

Suivant cette méthode de simulation, le programme du microcontrôleur est maître de son exécution. Elle est adaptée à notre architecture qui n'utilise pas les interruptions.

7.2.5. Modélisation de l'environnement

Le contrôleur de réseau NOÉ réalise l'interface entre le bus PCI et le réseau HSL. Il peut recevoir des événements provenant de ces deux sources : le processeur hôte et le réseau.

7.2.6. Le processeur de la station de travail

Le processeur de la station de travail est modélisé par les écritures et les lectures dont il est initiateur, et dont l'opérateur MEMRW de NOÉ est la cible. L'ensemble de ces requêtes est contenu dans un fichier. A chaque ligne de ce fichier correspond une transaction PCI représentée par les informations suivantes : la date absolue, le type de transaction (lecture ou écriture), l'adresse et la donnée dans le cas d'une écriture.

Ce fichier est interprété par l'opérateur modélisant MEMRW.

7.2.7. Le réseau HSL

Les paquets arrivant du réseau HSL sont décrits dans deux fichiers (un par lien). Ces fichiers sont interprétés par les opérateurs de réception de paquets PXA et PXB. Les opérateurs se mettent en attente jusqu'à la date indiquée dans la description des paquets à recevoir.

Du point de vue des émisssions, le réseau est modélisé par un débit `HSL_DEB`. Le modèle ne prend pas en compte les engorgements possibles du réseau, car on considère que le goulot d'étranglement est le contrôleur lui-même. La description des paquets émis est placée dans deux fichiers dont la syntaxe est la même que celle décrivant les paquets reçus du réseau. Cette caractéristique permet de reboucler l'émission et la réception.

7.2.8. La mémoire

La mémoire est modélisée par un tableau qui est initialisé lors de l'initialisation du simulateur VNOÉ.

En fin de simulation, l'état de la mémoire peut être écrit dans un fichier. L'ensemble des transactions PCI (dont NOÉ est l'initiateur) est également décrit dans un fichier spécifique.

L'ensemble de la plate-forme peut être résumé sur ce schéma.

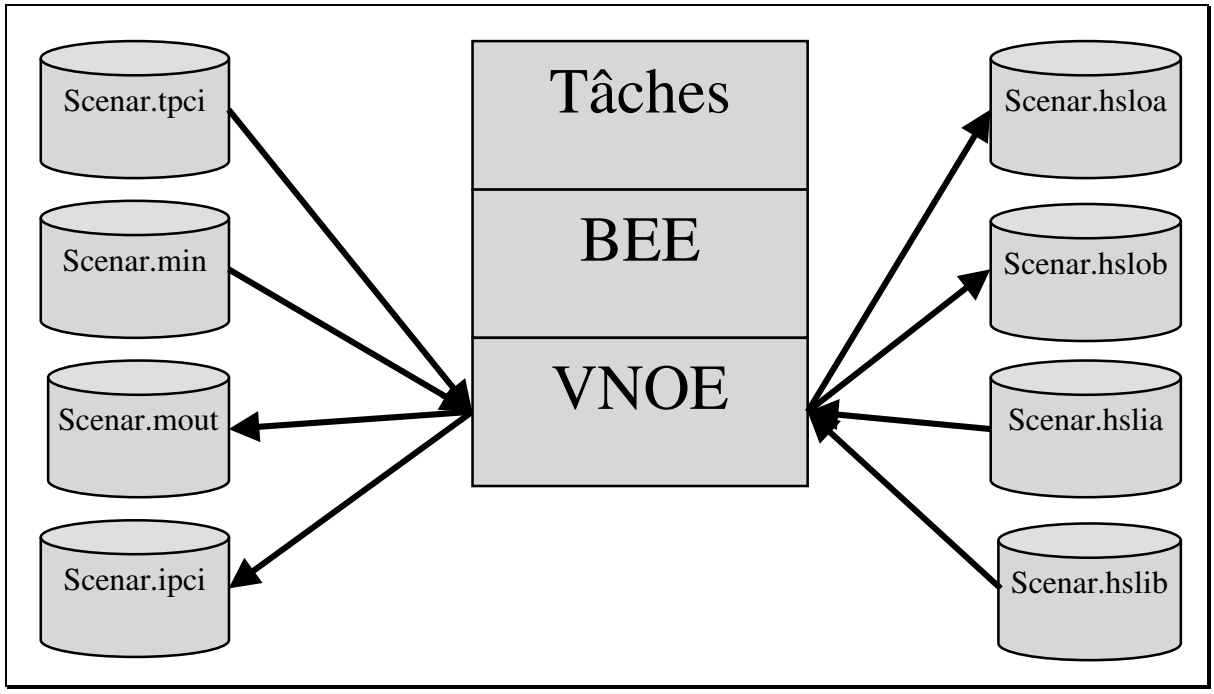


Figure 7.2 La plate-forme BEE/VNOÉ

7.2.9. Exemple de modélisation d'un opérateur

L'exemple suivant correspond à la modélisation d'une version simplifiée de l'opérateur de DMA en lecture. Cette version ne prend pas en compte la destination des données.

```
void noe_dmar()
    {

    while (1)
        {
        command = fifo_read(&dmar_com);
        adr_lo = fifo_read(&dmar_adr);
        dest_fifo = (command >> 24) & 0xf;
        data_length = command & 0x3ffff;

        ppci();
        pci_lat = AV_PCI_LAT+((rand()*SD_PCI_LAT)/RAND_MAX)-SD_PCI_LAT/2;
        wait_for(pci_lat);

        vfifo_write(&dmar2pb_a, data_length, PCI_DEB);
```

```
        vpci();
        wait_for(DMAR_LAT);
        vnoe_reg[NOE_DMAR_DONE_CNT>>2]++;
        }
    }
```

La latence d'obtention du bus PCI est générée de façon aléatoire en utilisant les paramètres `AV_PCI_LAT` et `SD_PCI_LAT` fournis par un fichier de configuration. Les fonctions *ppci* et *vpci* correspondent à la consultation et la mise à jour du sémaphore modélisant le partage du bus par les deux opérateurs de DMA.

7.3. DDSLR, un protocole élémentaire

Pour réaliser l'évaluation des performances, nous avons écrit les trois tâches (PCI, TXA/B et RXA/B) correspondant à la mise en œuvre d'un protocole réseau semblable à celui réalisé par le composant PCIDDC. Ce protocole est basé sur l'utilisation de deux listes de descripteurs LPE[46] et LMI[47]. La LPE est la liste des descripteurs correspondant aux messages à émettre, et la LMI la liste des descripteurs des messages reçus. Ces deux listes sont stockées dans la mémoire du processeur hôte.

7.3.1. La tâche PCI

Cette tâche interprète les écritures dont NOÉ est cible. Lorsqu'elle reçoit une écriture correspondant à la notification de l'ajout d'un descripteur, elle fait un V sur le sémaphore `LPE_ENTRY`.

7.3.2. La tâche de transmission

La tâche de transmission assure l'émission des paquets sur le réseau. Il existe une instance de cette tâche associée à chaque lien HSL. Après l'initialisation, cette tâche est bloquée sur le sémaphore `LPE_ENTRY`. Après le V effectué par la tâche PCI, elle initie la commande de DMA en lecture, pour lire le descripteur du message à transmettre. Enfin, elle analyse ce descripteur et commande l'émission du paquet.

7.3.3. La tâche de réception

Une tâche de réception est associée à chaque lien HSL. Après l'initialisation, cette tâche se met en attente de la réception d'un en-tête de paquet, en faisant un P sur le sémaphore `HEADER_EXTRACT`. Lorsqu'un en-tête a été reçu, il est analysé par la tâche. Elle en déduit où

[46] List of Page to Emit

[47] List of received Messages Identifiers

doivent être déposées les données, dans la mémoire de la station de travail. Elle commande l'opérateur de DMA en écriture pour les données et lorsque l'opération est terminée, elle écrit le descripteur du message dans la LMI.

7.4. Les Performances

De manière à pouvoir être comparées avec celles des autres architectures, les performances de l'architecture NOÉ ont été mesurées par simulation avec les hypothèses suivantes :

- un bus PCI 32 bits fonctionnant à 33 Mhz ;
- un microcontrôleur Motorola 8240 fonctionnant à 266 Mhz ;
- un réseau HSL fonctionnant à (2*66Mo/s).

Ces choix auront principalement un impact sur le débit maximum atteint. Ces performances ont été évaluées pour le protocole DDSLR. Il s'agit des premiers résultats obtenus : le programme du microcontrôleur n'a subi aucune optimisation pour accroître ses performances.

7.4.1. Le débit

Le débit est mesuré dans le cas d'émissions et de réceptions simultanées sur les deux liens. Les deux « packets builders » sont utilisés simultanément et les paquets reçus sur les deux liens HSL ne sont séparés par aucun temps mort. Ce contexte correspond à une sollicitation maximale du microcontrôleur.

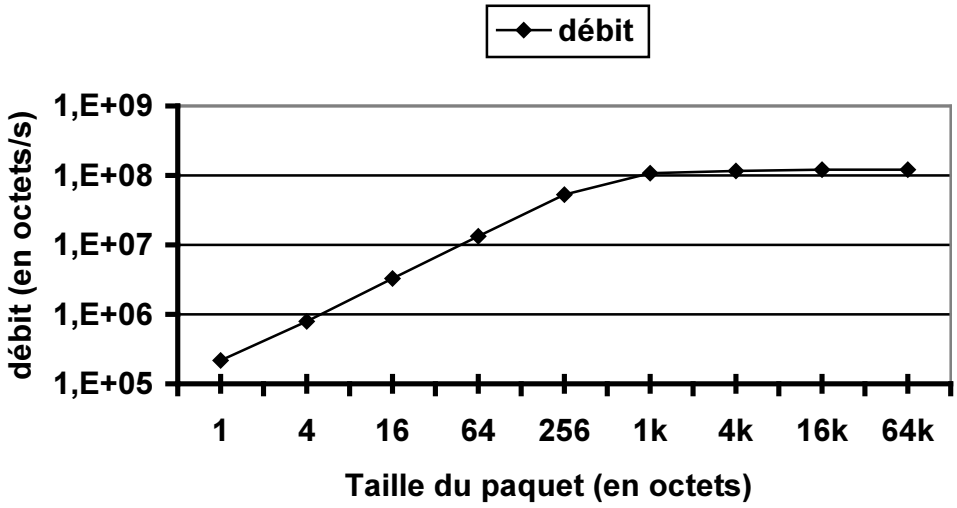


Figure 7.3 Débit NOÉ

La valeur maximale du débit mesuré est de 121 Mo/s, soit 92% du débit maximum offert par le bus PCI. Le demi-débit est atteint pour des paquets d'une longueur égale à environ 512 Octets.

7.4.2. La latence

La latence est mesurée « one way » comme pour la carte Myrinet. Aucune réception n'est en cours durant l'émission et, aucune émission durant la réception.

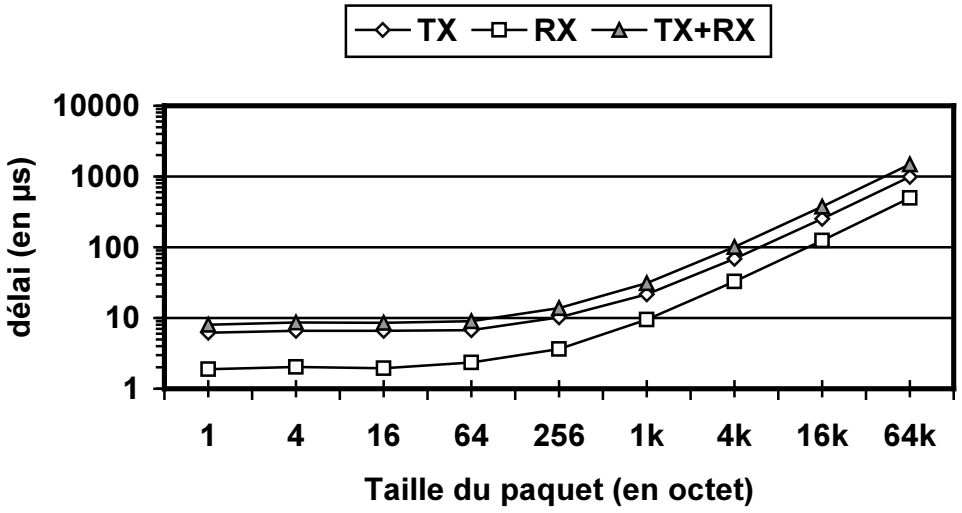


Figure 7.4 Latence NOÉ

La latence mesurée pour un paquet de taille minimale est égale à 8 µs (respectivement 6µs pour l'émission et 2µs pour la réception). Comme nous pouvons le constater sur ce graphique, la latence pour les petits paquets est essentiellement due aux traitements effectués par le microcontrôleur. En effet, jusqu'à 64 octets, le délai de transmission reste constant.

7.4.3. Incidence de la charge sur le contrôleur de réseau NOÉ

La latence présentée au paragraphe 7.4.2 a été mesurée alors que le contrôleur de réseau NOÉ n'effectue qu'une opération à la fois (soit une émission, soit une réception). Il s'agit d'un cas très favorable et il est intéressant d'évaluer l'incidence de la charge du contrôleur sur les performances. Nous nous sommes placés dans le « pire cas » qui correspond à une charge maximale du contrôleur (2 émissions et 2 réceptions simultanées). Cette mesure est intéressante, car elle nous montre la capacité du microcontrôleur et du micronoyau à traiter plusieurs émissions et réceptions simultanées. Si nous divisons le temps pris par l'ensemble des transmissions par leur nombre, nous obtenons un délai moyen de transmission égal à 9.2µs. Si nous comparons ce résultat à la latence obtenue « à faible charge », nous constatons qu'en moyenne la gestion des 4 activités simultanées n'accroît que de 15% la latence. Cet accroissement est du à la nécessité pour les tâches de partager les opérateurs PCI ainsi qu'au nombre plus important de commutations de tâches.

7.4.4. Taille minimale des paquets

Cette mesure est destinée à évaluer la capacité du contrôleur de réseau NOÉ à extraire les paquets du réseau. Bien que la complexité des traitements à effectuer par le microcontrôleur lors de la réception d'un paquet soit fortement dépendante du protocole réalisé, il est intéressant d'estimer la taille minimale des paquets permettant le complet recouvrement du traitement de l'en-tête du paquet n+1 par le microcontrôleur avec la transmission des données du paquet n sur le bus PCI. Pour le protocole DDSLR, le temps de réception d'un paquet de taille minimale est de 2µs. En faisant l'hypothèse que ce temps soit entièrement dû aux traitements effectués par le microcontrôleur (hypothèse pessimiste), cette taille minimale de paquet est égale à environ 132 octets. En deçà de cette limite, le traitement des en-têtes devient le facteur limitant.

7.5. Les limites de la méthode de mesure

Ces résultats ont été obtenus par simulation. Le contexte n'est pas le même pour la simulation (effectuée sous Linux), que pour l'utilisation réelle du microcontrôleur (utilisation d'un processeur Pentium à la place d'un microcontrôleur Motorola, pas de mémoire virtuelle donc pas de problème de gestion de TLB, forte localité des données et du programme) et nous pensons que les résultats obtenus sont pessimistes.

Lors d'une simulation, l'évaluation du simulateur du composant NOÉ constitue une part importante des traitements effectués par le processeur de la station de travail. Lors de ces simulations, il est fait appel au système d'exploitation pour accéder aux différents fichiers de commande et de trace. Bien que le temps correspondant à ces appels ne soit pas pris en compte, la modification du contenu des caches a un effet négatif artificiel, et nous avons constaté une forte incidence de ces traitements sur les résultats mesurés.

Nous avons évalué le temps d'exécution des différentes fonctions destinées à être implantées sur le microcontrôleur, lorsqu'elles étaient sorties de leur contexte. Les temps obtenus sont très inférieurs à ceux obtenus dans le contexte de la simulation (de l'ordre d'un facteur 2). Ces temps sont actuellement utilisés pour la simulation dans l'objectif d'obtenir une évaluation moins pessimiste des performances de cette architecture.

Il est fort probable que ces différences sont dues à des mises à jour des caches du fait des changement de contextes. Lors de l'exécution réelle par le microcontrôleur du programme réalisant le protocole, ces phénomènes ne se produiront pas.

Il apparaît que notre méthode d'évaluation des temps nous conduit à obtenir des temps supérieurs à ceux que nous obtiendrons en réalité, et les résultats que nous présentons pour la latence ainsi pour que le « demi-débit » sont des approximations pessimistes de la réalité.

7.6. Conclusion

La plate-forme VNOÉ nous a permis de mettre au point le micronoyau BEE. Elle nous a été très utile pour la mise au point des tâches DDSLR. L'objectif de réaliser une plate-forme de développement pour le micro-logiciel embarqué est donc atteint.

Le micronoyau BEE permet de gérer efficacement le parallélisme contenu dans le composant NOÉ (seulement 15% de surcoût en latence).

Nous pouvons comparer maintenant nos résultats aux autres machines :

	Débit	Latence	Surcoût Processeur	Extensibilité	Fiabilité	Programmabilité
The HIVE	Faible	Elevée	Elevé	Faible	Corrigée	Nulle
BIP/Myrinet	Elevé	Faible	Moyen	Moyenne	Bonne	Moyenne
MPC0	Elevé	Faible	Moyen	Excellente	Corrigée	Faible
NOÉ	Elevé	Moyenne	Faible	Excellente	Bonne	Excellente

Les premiers résultats obtenus concernant la latence ne sont pas très satisfaisants par rapport à la latence de la machine MPC0 qui réalise le protocole DDSLR de façon « cablée ».

Le principal gain de cette architecture est de décharger le processeur de la station de travail du traitement des communications, et de permettre la mise en œuvre des protocoles de communication sans appels systèmes tels que RWU.

CHAPITRE 8

Conclusion

Notre objectif était de concevoir un contrôleur de réseau adapté aux machines parallèles de type « grappe de PC ».

Pour atteindre cet objectif, nous avons été amenés à préciser quelles étaient les contraintes spécifiques à ce type d'architecture de machine parallèle. La structure particulière du nœud de calculs (une carte mère de PC avec bus d'entrée/sortie PCI) a orienté notre étude vers les architectures de communication du type « à passage de messages ».

L'étude des machines existantes nous a montré que, pour réduire la latence, les appels au système d'exploitation étaient à proscrire lors des communications. Nous avons également constaté que pour la mise en œuvre de bibliothèques de passage de message telles que PVM, jusqu'à 99 % du temps de latence était du aux traitements effectués par le processeur de la station de travail. La conséquence immédiate de ces traitements est une perte de temps de calcul pour les applications, et une impossibilité de recouvrement des temps de communication.

Cet ensemble de constatations nous a amené à définir le protocole RWU. Ce protocole permet de rendre le réseau directement accessible aux processus, tout en garantissant l'intégrité du système et des autres applications.

La définition du protocole RWU a été utile pour la spécification de l'architecture du contrôleur de réseau NOÉ.

L'architecture de ce contrôleur de réseau est basé sur l'utilisation d'un microcontrôleur dont la fonction est de mettre en œuvre le protocole de communication. L'ensemble des transferts de données est assuré par un second circuit spécifique NOÉ. Ce circuit contient des opérateurs spécifiques à la gestion des paquets, et des opérateurs permettant les accès directs à la mémoire de la station de travail (via le bus PCI). NOÉ dispose également d'une interface cible

sur le bus PCI. Elle permet au microcontrôleur de recevoir les ordres venus du processeur de la station de travail. L'ensemble de ces opérateurs est commandé par le microcontrôleur. Le circuit VLSI NOÉ a été réalisé en technologie ST 0,25µ et contient 1.3 Millions de transistors. Le circuit NOÉ est capable de gérer simultanément deux émissions et deux réceptions. La gestion de ces quatre activités par le microcontrôleur nécessite une structure de programmation particulière. Pour supporter cette structure spécifique, nous avons développé un micronoyau multitâche adapté à notre architecture.

Le micronoyau BEE offre les primitives minimales permettant la communication et la synchronisation entre les tâches. BEE contient un ordonnanceur permettant aux tâches de se partager le microcontrôleur. Un ensemble de sémaphores permet aux tâches de se partager l'accès aux opérateurs de NOÉ.

Une première version du micronoyau BEE a été développée pour l'architecture i386.

Aucune carte permettant de tester le circuit NOÉ n'étant disponible, il nous a été nécessaire de développer une plate-forme de simulation pour évaluer les performances de cette architecture.

Les simulations effectuées sur cette plate-forme nous ont montré que l'architecture NOÉ satisfaisait aux objectifs que nous nous étions fixés. Associé au micronoyau BEE, cette architecture offre une très grande facilité de programmation tout en assurant aux applications de profiter des performances du réseau. Elle permet ainsi de décharger le processeur de la station de travail de la gestion des communications. D'une façon générale, les premières performances mesurées sont encourageantes.

La plate-forme constitue un environnement de mise au point des tâches très confortable, et elle peut être utilisée pour le développement de nouveaux protocoles.

Nous ne pouvons pas garantir aujourd'hui la disponibilité d'une carte NOÉ-HSL composée du circuit NOÉ et d'un microcontrôleur. Néanmoins, cette architecture ne devrait pas tarder à être disponible sous la forme d'un circuit programmable intégrant les opérateurs de NOÉ et le microcontrôleur, et devrait permettre la mise au point rapide de premiers prototypes.

Cette architecture va pouvoir servir de base à la réalisation d'une nouvelle génération de machine MPC utilisant le protocole RWU.

Notre réalisation ouvre la porte à des recherches portant sur les protocoles de communication de bas niveau.

Les solutions que nous avons apportés au problème de la gestion simultanée des opérateurs par le microcontrôleur, devraient également être applicables dans le contexte des « system on

a chip », pour résoudre le problème de la coordination de plusieurs coprocesseurs matériels spécialisés par un unique processeur central.

Bibliographie

[Alt99] http://www.alteon.com/products/ace180-data.html
Alteon. NIC Data Sheet. WWW page,
August 1999.

[And95] T. E. Anderson, D. E. Culler, D. A. Patterson
A Case for NOW (Network Of Workstations)
IEEE Micro, p. 54-64 ;
February 1995.

[Arches] http://www.pact.srf.ac.uk/arches/

[Bhoe98] R. A. F. Bhoedjang, T. Rühl, H. E. Bal
User-Level Network Interface Protocols
IEEE Computer, p. 53-60 ;
November 1998.

[BIP] http://lhpca.univ-lyon1.fr/bip.html

[Cap98] F. Cappello, O. Richard
Architectures parallèles à partir de réseaux de stations de travail :
Réalités, Opportunités et enjeux
Calculateurs Parallèles, Vol. 10 – n°1/1998, p.9-34
Février 1998.

[Cran00] C. D. Cranor, R. Gopalakrishman, P. Z. Onufryk
Architectural Considérations for CPU and Netwok Interface Integration
IEEE Micro, p. 18-26
January/February 2000.

[Dunn98] D. Dunning, G. Reignier, G. McAlpine, Don Cameron, B. Shubert, F. Berry
A. M. Merritt, E. Gronke, C. Dodd
The Virtual Interface Architecture
IEEE Micro, p. 66-76 ;
March/April 1998.

[Eick92] T. von Eicken, D. E. Culler, S. C. Goldstein, K. E. Schauser
Active Messages : a Mechanism for Integrated Communication and Computation
Proceedings of the 19th Internationnal Symposium on Computer Architecture,
ACM Press ;
Gold Coast Australia May 1992.

[Eick98] T. von Eicken, W. Vogels
Evolution of the Virtual Interface Architecture
IEEE Computer, p. 61-68 ;
November 1998.

[Euro] http://www.1355-association.org/projects/europro.htm

[Fenyo] A. Fenyö
Conception et réalisation d'un noyau de communication sécurisé pour la machine parallèle MPC.
Thèse de doctorat de l'Université de Paris VI ;
A soutenir.

[Grap] J-L. Pazat
http://www.irisa.fr/grappes/

[Grei98] A. Greiner, P. David, JL. Desbarbieux, A. Fenyö, JJ.Lecler, F. Potter, V. Reibaldi, F. Wajsburt, B. Zerrouk
La Machine MPC
Calculateurs Parallèles, Vol. 10 – n°1/1998, p.71-84
Février 1998.

[Herb98] M. Herbert, F. Naquin, L. Prylli, B. Tourancheau, R. Westrelin
Protocole pour le Gbit/s en réseau local
L'expérience Myrinet
Calculateurs Parallèles, Vol. 10 – n°1/1998, p.35-54
Février 1998.

[HIVE97] J. E. Dorband
http://newton.gsfc.nasa.gov/thehive/

[HSL] IEEE
IEEE Draft Std 1355 Standard for Heterogeneous Interconnect (HIC) Low Cost Low Latency Scalable Serial Interconnect for Parallel System Construction. ;
August 1994

[Lanai] http://www.myri.com/scs/L3/documentation.html

[Lee98] W. S. Lee, W. J. Dally, S. W. Keckler, N. P. Carter, A. Chang
An Efficient Protected Message Interface
IEEE Computer, p. 69-75 ;
November 1998.

[Li89] K. Li and P. Hudak
Memory Coherence in Shared Virtual Memory Systems
ACM trans. Computer Systems, Vol. 7, No. 4, p.321-359
November 1989.

[Mark96] E. P. Markatos and M.G.H. Katevenis
Telegraphos : High-Performance Networking
Parallel Processing on Workstation Clusters
Procedings of HPCA-2 ;
San Diego, Ca, Feb. 1996.

[Mark97] E. P. Markatos and M.G.H. Katevenis
User-Level DMA without Operating System Kernel Modification
Proccedings of the Third Internationnal Symposium on High Performance Computer Architecture (HPCA-3) ;
San Antonio, TX, Feb. 1997.

[Méh98] JF. Méhaut, I. Demeure
Environnements d'exécution parallèle : PVM et MPI
Calculateurs Parallèles, Vol. 10 – n°5/1998, P.493-508
1998.

[MPC] A. Fenyö, O. Gluck
http://mpc.lip6.fr/

[Myr] http://www.myri.com

[P5] http://developer.intel.com/design/mobile/manuals/243191.htm
Intel Architecture Software Developer's Manual,
Volume 2: Instruction Set Reference Manual

[PCI] PCI Special Group of Interest
The PCI Local Bus Specifications. Revision 2.1
PCI-SIG ;
July 1995.

[Potter] F. Potter
Conception et réalisation d'un réseau d'interconnexion à faible latence et haut débit pour machine multiprocesseur
Thèse de doctorat de l'Université de Paris VI ;
1996.

[Pryl97] L. Prylli
Bip user reference manual.
Technical Report TR97-02, LIP/ENS-LYON ;
Septembre 1997.

[Prylli] L. Prylli
Réseaux et systèmes de Communication Locaux : Etude des couches logicielles de base.
Thèse de doctorat de l'Ecole Normale Supérieure de Lyon ;
Janvier 1998.

[PTW99] L. Prylli, B. Tourancheau, and R. Westrelin.
The design for a high performance MPI implementation on the myrinet network.
EuroPVM/MPI'99, number 1697 in LNCS, pages 223-230
Barcelone, Spain, 1999.

[Reibaldi] V. Reibaldi
Conception et réalisation d'un routeur de paquets à hautes performances
Thèse de doctorat de l'Université de Paris VI ;
1997.

[Renault] E. Renault
Etude de l'impact de la sécurité sur les performances dans les grappes de PCs .
Thèse de doctorat de l'Université de Versaille Saint Quentin ;
A soutenir.

[RWU] P. David, J-L. Desbarbieux, A Fenyo, J-J. Lecler, E. Renault, F. Wajsburt
Proposition pour un *Remote Write* en mode Utilisateur
1998.

[Tan] A. Tanenbaum
Systèmes d'Exploitation, systèmes centralisés, systèmes distribués
Ed. Prentice Hall
Juillet 1994.

[Wels98] M.Welsh, A. Basu, X. W. Huang, T. von Eicken
Memory Management for User-Level Network Interface
IEEE Micro, p. 77-82 ;
March/April 1998.

[Zerr95] B. Zerrouk, V. Reibaldi, F. Potter, A. Greiner and A. Derieux
Un routeur de messages adaptatif
Actes du 4ème Symposium sur les Architectures Nouvelles de Machines
Rennes, 13-14 décembre 1995.