Étude d'une architecture parallèle de processeur pour la transmission de données à haut débit

Abbas Ramazani

Laboratoire Interfaces Capteurs
et Microélectronique

École Doctorale IAEM - Lorraine
Département de Formation Doctorale Électronique - Électrotechnique

Thèse

Présentée à l'Université de Metz pour l'obtention du diplôme de
Docteur de l'Université de Metz
Discipline : Microélectronique

Etude d'une architecture parallèle de processeur pour la transmission de données à haut débit

Par

Abbas RAMAZANI

Soutenue le 19 juillet 2005 devant le jury composé de :

A. Dandache	Professeur à l'Université de Metz	Directeur de thèse
M. Sawan	Professeur à l'École Polytechnique de Montréal	Rapporteur
P. Garda	Professeur à l'Université Pierre et Marie Curie	Rapporteur
S. Weber	Professeur à l'Université Henri-Poincaré, Nancy 1	Examinateur
B. Lepley	Professeur à l'Université de Metz	Examinateur
F. Monteiro	Maître de conférence à l'Université de Metz	Examinateur
C. Diou	Maître de conférence à l'Université de Metz	Membre invité

LICM - 07 Rue Marconi, Technopôle, 57070 Metz, France
Tel : +33.(0)387.54.73.06/11 - Fax : +33.(0)387.54.73.07 - www.licm.sciences.univ-metz.fr

LABORATOIRE INTERFACES CAPTEURS
ET MICROÉLECTRONIQUE

École Doctorale IAEM - Lorraine
Département de Formation Doctorale Électronique - Électrotechnique

THÈSE

Présentée à l'Université de Metz pour l'obtention du diplôme de
Docteur de l'Université de Metz
Discipline : Microélectronique

ETUDE D'UNE ARCHITECTURE PARALLÈLE DE PROCESSEUR POUR LA TRANSMISSION DE DONNÉES À HAUT DÉBIT

Par

Abbas RAMAZANI

Soutenue le 19 juillet 2005 devant le jury composé de :

A. DANDACHE	Professeur à l'Université de Metz	Directeur de thèse
M. SAWAN	Professeur à l'École Polytechnique de Montréal	Rapporteur
P. GARDA	Professeur à l'Université Pierre et Marie Curie	Rapporteur
S. WEBER	Professeur à l'Université Henri-Poincaré, Nancy 1	Examinateur
B. LEPLEY	Professeur à l'Université de Metz	Examinateur
F. MONTEIRO	Maître de conférence à l'Université de Metz	Examinateur
C. DIOU	Maître de conférence à l'Université de Metz	Membre invité

LICM - 07 Rue Marconi, Technopôle, 57070 Metz, France
Tel : +33.(0)387.54.73.06/11 - Fax : +33.(0)387.54.73.07 - www.licm.sciences.univ-metz.fr

Remerciements

Je voudrais remercier profondément Monsieur **Bernard LEPLEY**, directeur du laboratoire LICM, de m'avoir accueilli dans son laboratoire. Qu'il trouve toute l'expression de ma sincère reconnaissance.

Je remercie Monsieur **Mohamad SAWAN**, professeur à l'école polytechnique de Montréal et Monsieur **Patrick GARDA**, professeur à l'université Pierre et Marie Curie, Paris, qui ont accepté d'être membres du jury et d'assumer la tâche de rapporteur. Pour sa participation au jury et en avoir assuré la présidence, je remercie Monsieur **Serge WEBER**, professeur à l'université Henri-poincaré, Nancy I.

De même, j'exprime ma plus plus grande gratitude à Monsieur **Abbas DANDACHE**, professeur à l'université de Metz pour avoir encadré mes travaux et pour son soutien moral et logistique.

J'exprime mes sincères et profonds remerciements à Monsieur **Fabrice MONTEIRO**, maître de conférences à l'université de Metz, et Monsieur **Camille DIOU**, maître de conférences à l'université de Metz, de m'avoir accordé beaucoup de leur temps, pour tous les conseils et les discussions.

Je remercie Monsieur Stanislaw J. Piestrak, pour ses conseils et ses aides.

Ma profonde reconnaissance à tous mes amis du laboratoire, pour leur aide et leur gentillesse. Je tien à remercier particulièrement : Amine, Gérald, Etienne, Ali, Mazen, Nassima.

Enfin, je remercie, mon père, ma mère, mon épouse bien aimée et mes deux filles qui m'ont toujours manifesté leur soutien et leur confiance inestimable.

Table des matières

Introduction générale

Contexte et objectifs

Avec l'arrivée des technologies de réseaux haut débit telles que les fibres optiques, le goulot traditionnel de la bande passante sur le support de transmission de données a disparu. Aujourd'hui c'est la vitesse à laquelle un processeur peut exécuter un protocole de réseau qui est un facteur de limitation des réseaux. Le temps d'accès à la mémoire est divisé par deux environ tous les 10 ans pour les DRAMs, tandis que la vitesse du traitement de données double tous les 18 mois. Cependant, la bande passante utilisée par Internet augmente à la même vitesse que la capacité de traitement, et le volume du trafic sur Internet double tous les six mois [ISC][Roberts00][Schaller97]. Par conséquent, les méthodes traditionnelles d'implantation de réseaux ne peuvent satisfaire les nouvelles conditions.

Plutôt que de choisir une solution fortement spécialisée, telle que l'approche ASIC, il existe la possibilité de développer une architecture ayant des niveaux élevés de flexibilité et de performance.

Des niveaux de flexibilité élevés et un bas coût de conception sont généralement atteints pour les applications par le développement logiciel tandis que le matériel est le choix normal pour implanter les algorithmes ayant de fortes contraintes temps réel.

Les « modems logiciels », par exemple, sont des interfaces de réseau où la conception logicielle est prédominante. Ils sont moins chers et plus faciles à mettre à jour et à configurer que les versions matérielles, mais offrent une vitesse réduite par rapport aux modems matériels. Quand flexibilité et vitesse sont exigées, un compromis efficace entre le logiciel et le matériel doit être trouvé. Dans ce processus de conception un effort particulier devrait être dédié à l'extraction du parallélisme et à sa transposition vers des blocs logiciels et matériels. Le parallélisme à grain fin ou à gros grain, la répartition du chemin de données, l'organisation de la mémoire et la synchronisation des échanges de données sont les aspects importants à analyser pour une bonne adéquation algorithme/architecture.

Les architectures de processeurs spécialisés avec des modules dédiés peuvent être de bons candidats pour remplir les conditions requises par le traitement des protocoles de réseau. Parmi ces architectures, les processeurs de réseau (*Network Processors*) sont en train de devenir un dispositif prédominant dans le domaine du matériel de réseau [Peyravian03]. Ils sont généralement utilisés dans les routeurs et les commutateurs. Avec l'émergence de nouveaux protocoles de réseau et l'augmentation de la vitesse de transmission des données, une deuxième génération de processeurs de réseau apparaît pour laquelle la recherche académique et industrielle est activement conduite vers de nouvelles techniques et méthodologies de conception et d'implantation.

Du côté des utilisateurs du réseau, il n'existe pas les mêmes contraintes de débit que pour le cœur du réseau. Cependant, le problème se posera bientôt également pour les

interfaces réseau des utilisateurs. Ces interfaces incluent deux groupes principaux : les modems et les cartes réseau. La vitesse de la ligne dans les réseaux locaux (LANs) augmente aussi rapidement que dans le cœur des réseaux et l'utilisation de réseaux Gigabit directement reliés aux machines des utilisateurs est pour bientôt. Afin de répondre à ces contraintes ainsi qu'aux exigences sur la flexibilité des protocoles, il est essentiel de trouver une nouvelle stratégie de traitement des protocoles, ainsi que le matériel approprié pour traiter ces protocoles dans les machines des utilisateurs.

Cette stratégie devrait être assez générique pour convenir à un grand nombre des protocoles dominants et à différents types de terminaux utilisateurs (les machines des utilisateurs du réseau). Pour cela, il faut identifier les caractéristiques communes des protocoles, puis construire une architecture satisfaisant ces conditions. Aujourd'hui, en raison de la diversité dans le domaine des protocoles, des types de données et des terminaux utilisateurs, le processeur de protocole est devenu un goulot assez complexe dans la conception des systèmes numériques de télécommunication. La nature temps réel du traitement des protocoles et les aspects liés au traitement du signal numérique dans certains protocoles (comme les réseaux sans fil) relient ce problème au domaine des architectures de processeurs pour le traitement du signal numérique.

Le grand défi dans la conception des processeurs de protocoles est de trouver une architecture qui soit un bon compromis entre celle d'un processeur généraliste et celle d'un circuit dédié ASIC. Les processeurs commerciaux disponibles dans ce domaine sont essentiellement les processeurs de réseau utilisés dans les routeurs et les commutateurs. Cependant, il y a quelques remarques à effectuer quant à cette approche. Premièrement, comment les architectures doivent-elles être choisies pour une bonne adéquation à une tâche particulière dans un protocole typique ? Deuxièmement, ces processeurs sont optimisés pour fonctionner dans les commutateurs plutôt que dans les terminaux utilisateurs. Troisièmement, le coût élevé de ces processeurs dû à leur technologie est un inconvénient important. En outre, il est nécessaire de disposer d'une méthode optimale pour adapter les algorithmes à des architectures possédant différents chemins de données. Ainsi, pour atteindre un bon niveau de performance, il faut recourir à des techniques logicielles spécifiques pour extraire le parallélisme intrinsèque des algorithmes.

Aujourd'hui, la plupart des recherches académiques et industrielles portent sur les architectures de processeurs de réseau. Excepté pour quelques protocoles particuliers [Hobson99][Kouks02], il n'existe pas beaucoup de projets académiques concernant les architectures pour le traitement des protocoles de couches basses dans les terminaux utilisateurs.

Le travail de cette thèse s'intègre dans un projet général plus vaste au sein du laboratoire LICM concernant la conception architecturale d'une chaîne de transmission fiable à haut débit. L'objectif global est de concevoir un processeur spécialisé dans le traitement rapide des algorithmes des divers protocoles présents dans les couches basses des modèles de références (OSI, Internet, ITU-T/ATM). Cette thèse est la continuation du travail effectué au sein du laboratoire LICM [Philip99] concernant le développement d'une architecture de processeur dédiée aux applications modem câble TV. Cette architecture était constituée d'un cœur DSP de type VLIW et de modules dédiés. Plusieurs travaux de recherche ont été consacrés au développent de modules dédiés au (dé)codage des codes correcteurs d'erreurs [Vallino99][M'sir03]. Dans la suite de cette thèse, nous proposons un modèle architectural et une méthodologie de conception pour un processeur de traitement des protocoles des couches basses (physique, liaison de donnés-MAC, et réseau) du modèle OSI et Internet.

Les contributions de ce travail peuvent être résumées par les axes suivants :

- Etude de divers protocoles de réseaux prédominant et extraction des tâches les plus communes et les plus critiques dans le traitement de ces protocoles.
- Proposition d'une nouvelle méthode d'estimation des performances des architectures de processeurs.
- Proposition d'un modèle architectural de processeur pour le traitement des protocoles.
- Définition de deux cœurs de processeurs spécifiques et implantation de ceux-ci sur FPGA.

Contenu de la thèse

Cette thèse est organisée en deux parties principales. La première comprend deux chapitres et donne un aperçu général de la thématique. La deuxième est composée de trois chapitres qui présentent le modèle et la méthodologie proposés ainsi que l'implantation des unités de traitement.

Première partie :

- Le premier chapitre présente les aspects généraux des réseaux informatiques et le traitement des protocoles. Les différents modèles de réseaux, les réseaux haut débit, et les caractéristiques du traitement des protocoles sont discutés.

- Le deuxième chapitre concerne les différentes architectures de processeurs. Les méthodes de parallélisme et les caractéristiques de différentes architectures sont étudiées de façon comparative, notamment les aspects architecturaux des processeurs DSP et de réseau.

Deuxième partie :

- Dans le troisième chapitre, la méthodologie proposée est présentée et ensuite une étude est menée pour extraire les tâches principales des protocoles dominants. Dans cette étude, les algorithmes les plus représentatifs sont choisis et une base de données est créée à partir des mesures effectuées sur l'exécution des algorithmes sur différentes architectures. Ensuite différentes architectures de processeurs sont évaluées à partir de cette base de données en utilisant des techniques d'analyse de données multi-dimensionnelle.

- Le quatrième chapitre présente le modèle architectural proposé pour un processeur de traitement de protocoles. Ce modèle est un modèle multi-processeurs composé d'un MU (*Master Unit*) et de différentes unités de traitement ou SPU (*Slave Processing Unit*) reliés par un réseau d'interconnexion. La performance de ce modèle est évaluée pour différents types de trafic, en se basant sur une simulation par événements discrets.

- Enfin, le cinquième chapitre concerne l'implantation de deux types de SPU sur FPGA : le SPU RISC pour les tâches générales et le SPU RISC-DSP pour les tâches orientées DSP.

Cette thèse se termine par une conclusion générale résumant l'ensemble de ce travail et deux annexes contenant les abréviations et la liste des publications.

Première partie

État de l'art : réseaux et architectures de processeurs

Chapitre 1

Réseaux

1.1 Introduction

Le traitement des protocoles de réseau se compose d'un nombre limité de types de tâches et donc il est concevable de lui dédier des processeurs spécialisés. La conception d'une architecture de processeur spécifique impose de connaître les caractéristiques des applications. Dans ce chapitre, nous expliquons les aspects généraux des réseaux informatiques et le traitement des protocoles.

La première partie du chapitre est consacrée aux trois modèles de référence des réseaux : le modèle OSI, le modèle TCP/IP et le modèle UIT-T. Le modèle OSI est tout à fait général et les fonctionnalités au niveau de chaque couche sont toujours très importantes. Les protocoles de modèle TCP/IP sont largement déployés. Le modèle UIT-T s'applique aux réseaux à commutation de cellules. Ce modèle est utilisé notamment par la norme ATM pour satisfaire les débits nécessaires au bon fonctionnement des applications multimédia.

La deuxième partie concerne différents réseaux haut débit. Les réseaux choisis sont généralement dominants dans le secteur des réseaux informatiques. Nous expliquons les aspects de ces réseaux en portant plus d'attention aux couches basses. Les aspects principaux des réseaux ATM, LAN sans fil, FDDI, DQDB, Ethernet haut débit sont discutés.

Dans la troisième partie sont introduites les caractéristiques du traitement des protocoles. Enfin, la dernière section détaille les techniques de parallélisme employés dans le traitement des protocoles.

1.2 Modèles de réseaux

Les ordinateurs sont reliés ensemble par des réseaux informatiques. Les langues utilisées par les ordinateurs pour échanger l'information s'appellent des protocoles. Le protocole spécifie tous les aspects des communications, y compris les détails syntaxiques tels que le format des paquets et les détails sémantiques tels que « comment faut-il traiter les paquets ? ». La communication est basée sur le transfert d'unités de données de taille limitée. Ces unités de données s'appellent des paquets, des trames ou des cellules selon le type de protocole. Dans ce mémoire, nous utilisons « paquet » dans le sens général d'unité de données. Chaque paquet se compose de deux parties : les données et l'en-tête. Les en-têtes contiennent par exemple les informations d'adresse et de contrôle. La figure 1.1 montre la structure générale d'un paquet.

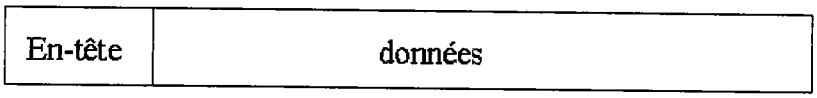


FIG. 1.1: La structure générale d'un paquet

Chaque protocole a sa propre structure d'en-tête. Les tailles des paquets peuvent être fixes ou variables. La plupart des réseaux sont organisés en couches ou niveaux, chacune étant placée au-dessus de la précédente. La communication entre deux machines utilise plusieurs protocoles appelés protocoles de couches. L'organisation de ces couches et des protocoles sont définis par le modèle de réseau. Il existe trois modèles importants, qui sont introduits dans la section suivante.

1.2.1 Modèle de référence (OSI)

OSI signifie *Open Systems Interconnection*, ce qui se traduit par « interconnexion de systèmes ouverts ». Ce modèle a été mis en place par l'ISO (*International Standardisation Organisation*) afin d'établir un standard de communication entre les ordinateurs d'un réseau, c'est-à-dire les règles qui gèrent les communications entre des ordinateurs. En effet, aux origines des réseaux, chaque constructeur avait un système propre (on parle de système propriétaire). Ainsi de nombreux réseaux incompatibles coexistaient. C'est la raison pour laquelle l'établissement de normes fut nécessaire [Tanenbaum03].

Le modèle OSI décrit en fait des niveaux de transmission mais non les protocoles proprement dits. Il divise l'ensemble des protocoles en sept couches indépendantes entre lesquelles sont définis deux types de relations : les relations verticales entre les couches d'un même système (interfaces) et les relations horizontales relatives au dialogue entre deux couches de même niveau (les protocoles). Les couches 1, 2, 3 et 4 sont orientées transmission et les couches 5, 6 et 7 sont orientées traitement.

- La couche physique (1) s'occupe de la connexion physique sur le réseau. En effet, cette couche se charge de la transmission de bits à l'état brut sur un canal de transmission. Les problèmes de conception de cette couche concernent les interfaces électriques, la synchronisation, ainsi que les specifications du support physique de transmission.
- La couche liaison (2) a pour but de transmettre les données sans erreur. Elle décompose les données sur l'émetteur en trames de données et les envoie en séquence. Différentes méthodes permettant de protéger les données contre les erreurs sont utilisées, comme le codage de détection et de correction d'erreur. Dans les réseaux à canal partagé, cette couche s'occupe aussi de contrôler l'accès au canal par une sous-couche MAC (*Medium Access Control*).
- La couche réseau (3) assure la commutation et le routage des paquets entre les nœuds du réseau. Il existe deux méthodes principales d'acheminement : la commutation de circuits et la commutation de paquets. Dans les cas où il y a beaucoup de paquets sur les nœuds, ou bien en cas de congestion, c'est la couche réseau qui doit résoudre le problème.
- La couche transport (4) permet l'établissement, le maintien et la rupture des connexions. Une tâche principale de cette couche est d'accepter des données de la couche supérieure et de les diviser en unités plus petites, ce qui constitue l'opé-

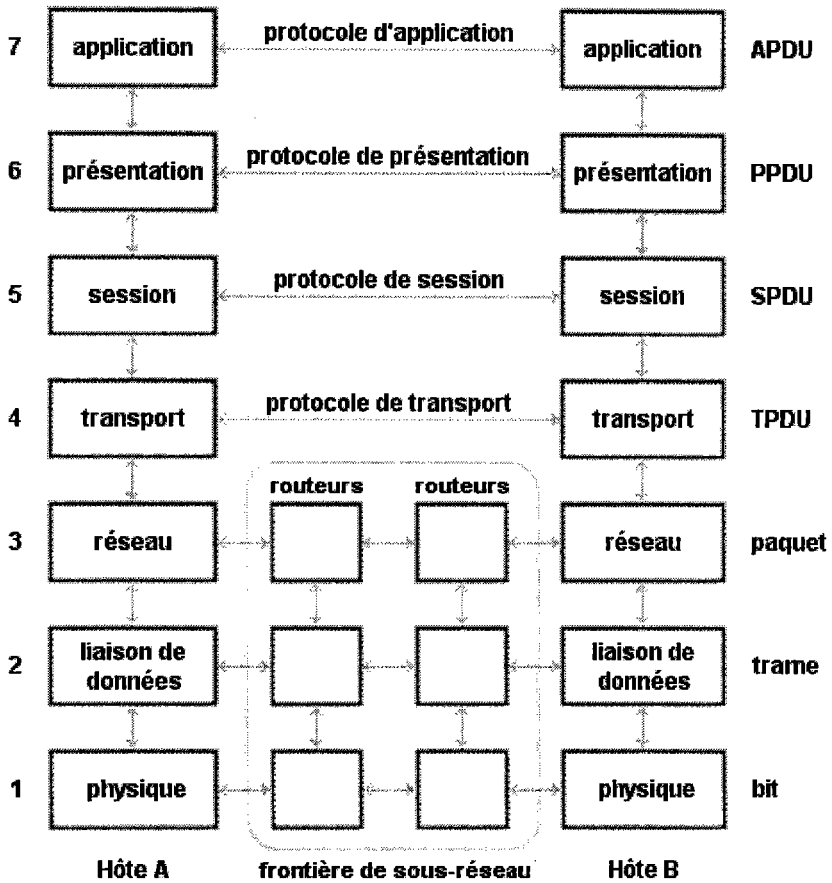


FIG. 1.2: Le modèle de référence OSI

ration de fragmentation. Elle offre un service réel de bout en bout de la source à la destination. C'est à dire qu'il existe toujours une conversation entre les machines source et destination par l'intermédiaire des en-têtes de messages et les messages de contrôle. Dans les couches plus basses, les protocoles établissent des relations entre une machine et ses voisins immédiats et non entre les machines source et destination. En effet, les couches 1 à 3 sont chaînées alors que les couches 4 à 7 sont de bout en bout.

- La couche session (5) permet d'établir une connexion logique entre deux applications. Elle assure l'organisation et la synchronisation du dialogue.
- La couche présentation (6) s'occupe de la syntaxe des données. Cette couche permet à deux machines de communiquer même lorsqu'elles utilisent représentations de données différentes. Elle gère les structures de données haut niveau pour accomplir cette tâche.
- La couche application (7) fournit les services et interfaces de communication aux utilisateurs [Tanenbaum03].

1.2.2 Modèle TCP/IP

Le modèle TCP/IP, inspiré du modèle OSI, reprend l'approche modulaire (utilisation des couches) mais contient uniquement quatre couches : la couche d'application (elle comprend les couches application, présentation, et session du modèle OSI), la couche transport (qui utilise les protocoles TCP ou UDP), la couche internet (la couche réseau) et la couche d'accès réseau (les couches liaison et physique d'OSI) [Comer00].

En effet, TCP/IP fournit un protocole standard pour résoudre le problème de connexion entre différents réseaux. TCP se charge du transport de bout en bout pour toute application alors que IP (*Internet Protocol*) est responsable du routage à travers le réseau.

La couche d'accès au réseau est la première couche du modèle TCP/IP, elle offre la capacité d'accéder à un réseau physique quel qu'il soit, c'est-à-dire les moyens à mettre en œuvre afin de transmettre des données via un réseau. Ainsi, la couche d'accès au réseau contient toutes les spécifications concernant la transmission de données sur un réseau physique, qu'il s'agisse de réseau local, de connexion à une ligne téléphonique ou de n'importe quel type de liaison à un réseau.

La couche Internet est la couche « la plus importante » (elles ont toutes leur importance) car c'est elle qui définit les datagrammes et qui gère les notions d'adressage IP. Elle permet l'acheminement des paquets de données vers des machines distantes. La couche Internet contient 5 protocoles : le protocole IP (*Internet Protocol*), le protocole ARP (*Address Resolution Protocol*), le protocole ICMP (*Internet Control Message Protocol*), le protocole RARP (*Reverse Address Resolution Protocol*) et le protocole IGMP (*Internet Group Management Protocol*). Les trois premiers protocoles sont les protocoles les plus importants de cette couche [Comer00].

La couche transport contient deux protocoles permettant à deux applications d'échanger des données indépendamment du type de réseau emprunté (c'est à dire indépendamment des couches inférieures...). Il s'agit des protocoles TCP (*Transmission Control Protocol*), un protocole orienté connexion qui assure le contrôle des erreurs, et UDP (*User Datagram Protocole*), un protocole non orienté connexion qui permet aux applications d'assurer elles-même le séquencement et le contrôle de flux. TCP fragmente le flot d'octets entrant en messages qu'il passe à la couche Internet. À l'arrivée, le TCP destinataire réassemble les messages reçus en un flot de sortie.

La couche application est la couche située au sommet des couches de protocoles TCP/IP. Celle-ci contient les " applications " réseau permettant de communiquer grâce aux couches inférieures. Les protocoles plus connus dans cette couche sont FTP (*File Transfer Protocol*), SMTP (*Simple Mail Transfer Protocol*), DNS (*Domain Name System*) et HTTP (*Hyper Text Transfer Protocol*). La figure 1.3 présente la structure de couches et un exemple des unités de données d'un réseau basé sur le modèle TCP/IP. Ethernet est utilisé comme couche physique et de liaison de données.

En bref, le modèle OSI est très utile pour analyser les réseaux informatiques, mais les protocoles OSI n'ont pas fait l'objet d'une large acceptation. Dans le cas de TCP/IP c'est tout le contraire : le modèle n'existe pour ainsi dire pas alors que les protocoles sont déployés partout [Tanenbaum03].

1.2.3 Modèle UIT-T

Les réseaux de communication de nouvelle génération utilisent des techniques de commutation de cellules. En relation avec cette commutation, l'UIT-T (Union Internationale des Télécommunications standardisation du secteur des Télécommunications) a

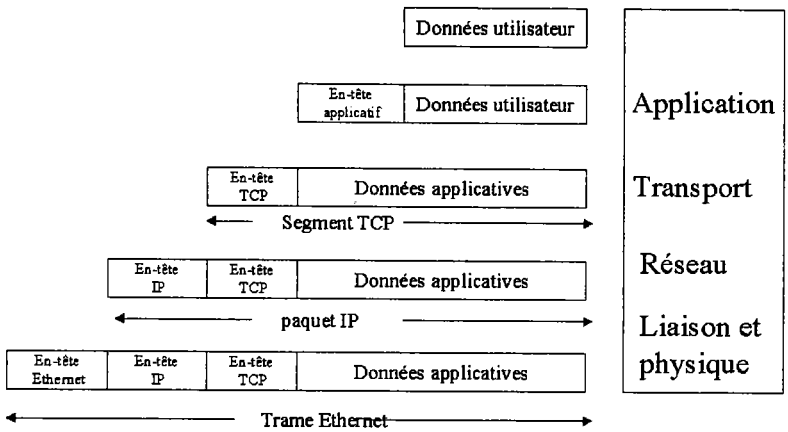


FIG. 1.3: Le modèle TCP/IP

développé un nouveau modèle de référence. Le modèle est composé de trois plans : le plan utilisateur, le plan contrôle, et le plan gestion, ainsi que de quatre couches. Le plan utilisateur est destiné au transport de l'information des utilisateurs. Le plan contrôle s'occupe de la signalisation. Le plan gestion offre les fonctions de surveillance du réseau, les fonctions de gestion de plan et les fonctions de gestion de couche. Les fonctions de gestion de plan permettent la coopération entre tous les plans et maintiennent le système. La gestion de couche s'occupe des flux OAM (*Operation And Maintenance*) de chaque couche et exécute les fonctions de gestion concernant les ressources et les paramètres des protocoles (cf. figure 1.5) [Tanenbaum03]. Nous expliquons ce modèle de façon plus détaillée dans la section consacrée au réseau ATM.

1.3 Classification des réseaux

Les réseaux informatiques peuvent être classés en quatre catégories différentes en fonction de la distance maximale séparant les points les plus éloignés du réseau.

Les PAN (*Personal Area Network*) sont les réseaux de plus petite taille. Ces réseaux réalisent des connexions sur quelques mètres entre les différents équipements personnels d'un même utilisateur. Une norme PAN très connue est la norme IEEE 802.15 intitulé WPAN (*Wireless Personal Area Network*). Trois groupes de services ont été définis dans cette norme. Chaque type de service présente les différents fonctionnalités pour particuliers et entreprises [Pujolle03].

Les réseaux locaux ou LAN (*Local Area Network*) sont constitués un ensemble d'éléments, connectés par des supports de transmission (câbles), qui offrent à des utilisateurs installés sur une surface limitée (quelques dizaines de mètres à quelques dizaines de kilomètres) les fonctionnalités nécessaires pour pouvoir relier des équipements informatiques. Un des objectifs des réseaux locaux est d'atteindre un niveau de transparence élevé vis-à-vis des utilisateurs : ceux-ci doivent détecter très peu de différences entre l'emploi d'un ordinateur autonome et celui d'un ordinateur connecté au réseau local.

Au niveau de la topologie, il existe trois types de réseaux locaux : en bus, en étoile et en anneau. Dans la topologie en bus, les machines sont connectées le long d'un seul câble,

la limite théorique est de 255 machines, ceci n'étant qu'une valeur théorique car la vitesse serait alors très faible. Dans la topologie en étoile, les machines sont connectées par des segments de câble à un composant central appelé concentrateur (hub). La topologie en anneau connecte les stations sur une boucle de câble continue et fermée. Les signaux se déplacent le long de la boucle dans une seule direction et passent par chacune des stations.

Les méthodes d'accès au média de communication, les réseaux LAN se divisent en deux groupes principaux : les méthodes d'accès CSMA (*Carrier Sense Multiple Access*) qui se divisent en différentes sous-méthodes, et les méthodes d'accès par jeton [Tanenbaum03].

Le protocole dominant des réseaux locaux (LAN) est Ethernet. Ethernet existe sous différentes formes et se développe constamment. C'est un réseau de type bus avec contrôle décentralisé qui fonctionne à des vitesses comprises entre de 10 Mbits/s et 10 Gbits/s. Ethernet définit les couches 1 et 2 du modèle OSI, c'est-à-dire les couches physique et liaison de données. Au dessus Ethernet, le Protocol IP (*Internet Protocol*) est le protocole couche réseau dominante.

Les couches 1, 2 et 3 composant Ethernet et IP constituent la base commune pour la plupart des LAN. Un LAN est relié au reste Internet par l'intermédiaire d'un ou plusieurs routeurs. Les couches 1 et 2 fournissent les communications point à point, bien que dans des versions précédentes d'Ethernet, tous les terminaux puissent intercepter toute communication.

Les réseaux métropolitains, ou MAN (*Metropolitan Area Network*), permettent l'interconnexion à haut débit des entreprises ou des particuliers sur un réseau spécialisé. Ces réseaux connectent des réseaux locaux situés à moins d'une centaine de kilomètres. La vitesse de transport doit atteindre un minimum de 100 Mbits/s. L'IEEE 802.6 ou le réseau DQDB (*Distributed Queue Dual Bus*) est une des normes de réseaux métropolitains. Il existe d'autres normes comme FDDI (*Fiber Distributed Data Interface*), qui permet de couvrir une distance de 100 kilomètres, ou les normes EPON (*Ethernet Passive Optical Network*) et IEEE 802.17 [Pujolle03].

Les réseaux étendus, ou WAN (*Wide Area Network*), sont destinés à transporter des données sur des distances à l'échelle d'un pays ou d'un continent. Ce sont soit de grands réseaux de fibre optique, soit des réseaux satellitaires.

1.4 Réseaux haut débit

1.4.1 ATM

ATM (*Asynchronous Transfer Mode*) est le nom d'une méthode de transmission de données mais est plus connu aujourd'hui comme un réseau qui utilise cette méthode. Dans ce réseau, une liaison numérique bidirectionnelle véhicule un flot de bits d'origine téléphonique, informatique ou audio-visuelle. La liaison numérique est multiplexée selon le principe du multiplexage TDM (*Time Division Multiplexing*). Elle utilise de petits paquets de données de taille fixe appelés cellules. L'ambition d'ATM est de véhiculer tout type d'information : voix, vidéo, données. en bref : « être un réseau multimédia ». Pour cela, il faut offrir un débit suffisant, parce que les applications multimédia ont besoin de liens avec des débits en Gigabits/s et une qualité de service (QoS) adaptés aux différents types de trafic. Cette diversité de trafic rend les différentes caractéristiques. Par exemple, certains applications temps réel (comme la voix et la video) tolèrent un trafic avec des pertes mais sans retard alors que d'autres applications (comme le transfert de fichiers) ne

peuvent admettre la moins perte, mais sont moins exigeantes du point de vue temporel.

Il existe deux méthodes générales pour transmettre des données sur des supports physiques : synchrone et asynchrone. En mode synchrone (STM : *Synchronous Transfer Mode*), il y a une allocation statique d'intervalles de temps pour chaque utilisateur. Dans ce cas, si dans un intervalle de temps le terminal n'a rien à envoyer, il y a une perte de bande passante et le débit disponible sur le réseau est spécifié par le débit de base. En mode asynchrone, il y a une allocation dynamique de canal, et lorsque une cellule est prête et peut être envoyyée, elle est envoyée : il n'y a pas de canal privé pour chaque utilisateur et l'en-tête de cellule spécifie l'adresse de la source et du destinataire. En mode asynchrone, la bande passante est utilisée plus efficacement. ATM utilise la méthode asynchrone pour l'envoie de cellules [Tanenbaum03].

Il existe cinq techniques principales de commutation : la commutation de circuits, la commutation de messages, la commutation de paquets, la commutation de trames et la commutation de cellules. La technique de transfert ATM utilise une commutation de cellules de taille fixe. Lorsque par exemple, la vitesse d'arrivée des cellules est de 155 Mbits/s, le commutateur doit traiter 365000 cellules par seconde (une cellule fait 53 octets) ce qui signifie que le temps de cycle du commutateur est de 2,7 μs. Un commutateur dispose de 16 à 1024 lignes d'entrées, normalement. Il y a donc un flot de 16 à 1024 cellules toutes les 2,7 μs. Des cellules de petite taille et de longueur fixe permettent de telles vitesses de commutation. Une caractéristique importante du réseau ATM est l'utilisation du mode avec connexion pour la transmission des cellules. Une cellule n'est transmise que lorsqu'un circuit virtuel est établi, ce circuit virtuel étant marqué à l'intérieur du réseau par des références précisées dans les tables de commutation placées dans chaque nœud traversé. De cette façon, deux machines qui veulent communiquer commencent par établir une connexion, avant d'envoyer leurs données avec un risque minimal de perte de cellules et une efficacité maximale pour leur traitement, ce qui permet notamment d'avoir une garantie sur le temps maximal qu'un paquet passera dans un commutateur [Pujolle03].

Chaque cellule ATM est de longueur constante : 53 octets comprenant 5 octets d'en-tête et 48 octets de données comme illustré dans la figure 1.4.

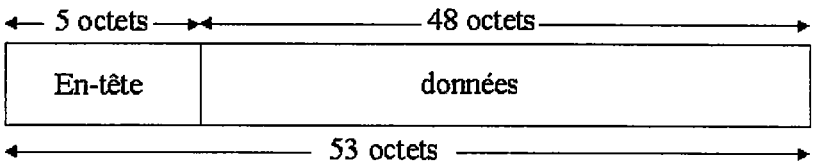


FIG. 1.4: La cellule ATM

VPI, VCI et HEC sont les parties les plus importantes de l'en-tête. VPI (*Virtual Path Identifier*) et VCI (*Virtual Channel Identifier*) sont les références permettant de commuter des cellules. Le cinquième octet de l'en-tête HEC (*Header Error Control*) permet à la fois de découvrir le début de la trame et de corriger une erreur dans l'en-tête. HEC est le reste de la division des 32 premiers bits par le polynôme $x^8 + x^2 + x + 1$ en ajoutant le nombre binaire de 01010101.

Les couches physique, ATM et AAL (*ATM Adaptation Layer*) sont les trois couches principales du réseau ATM. Ces couches sont constituées en un modèle tri-dimensionnel comme la figure 1.5 le montre.

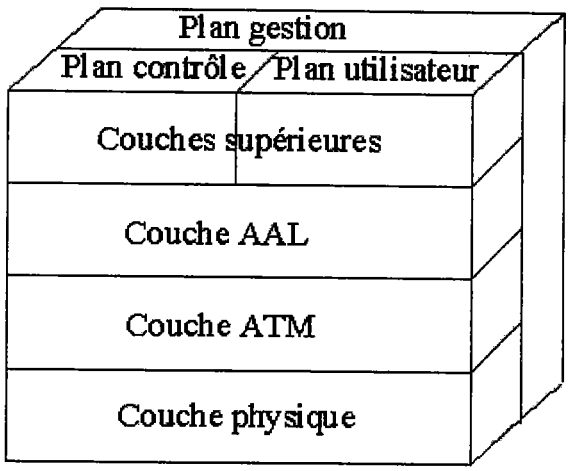


FIG. 1.5: Le modèle UIT-T

La couche la plus basse concerne les protocoles de niveau physique dépendants du médium (PMD*Physical Medium Dependent*). Cette couche est subdivisée en deux sous-couches : TC (*Transmission Convergence*) et PM (*Physical Medium*). La sous-couche TC s'occupe du découplage du débit cellule (adaptation du débit cellule au support de transmission), de la génération et vérification des en-têtes et de la génération des cellules. La sous-couche PM insère et extrait des bits sur le support de transmission.

La couche ATM réalise et supervise la transmission de cellules d'une source vers une destination et effectue aussi le routage et la commutation. Ces fonctionnalités sont typiques d'une couche réseau. De ce point de vue, on peut dire que la couche ATM est une couche de type réseau. La couche ATM est orientée connexion, aussi bien du point de vue des services offerts que de son mode de fonctionnement interne. L'élément de base de la couche ATM est une connexion logique d'une source vers une destination, appelée circuit virtuel. La couche ATM peut aussi établir une connexion multi-destinataires. La couche ATM ne délivre pas d'accusé de réception à la transmission des cellules (le support étant supposé très fiable). Dans la couche ATM, on trouve deux types d'interfaces : UNI (*User Network Interface*), et NNI (*Network Network Interface*). Il existe deux formats d'en-tête, l'un pour les cellules à interface UNI et l'autre pour les cellules à interface NNI. Dans un canal de transmission entre une source et une destination, plusieurs circuits virtuels peuvent être regroupés pour former un conduit virtuel (*virtual path*) : l'en-tête de cellule contient donc deux adresses VCI et VPI pour le routage des cellules. L'établissement d'un circuit virtuel utilise six messages et chaque message occupe une ou plusieurs cellules. Ces messages circulent d'un ordinateur vers le réseau ou du réseau vers un ordinateur. Le standard ATM ne définit aucun algorithme de routage particulier. C'est à l'opérateur réseau de choisir celui qui convient [Pujolle03].

Le but de la couche AAL (*ATM Adaptation Layer*) est de fournir des services utiles aux programmes d'application et de faire la segmentation et le réassemblage des données c'est-à-dire faire l'adaptation entre la couche ATM et les programmes d'application. Cette couche contient deux sous-couches : CS (*Convergence Sub-layer*) et SAR (*Segmentation And Reassemblage*). La CS prépare l'information des couches supérieures pour la conver-

Classe	Protocole
A : les services temps réel à débit constant et mode orienté connexion	AAL1
B : les services temps réel à débit variable et mode orienté connexion	AAL2
C/D : les services non temps réel à débit variable et mode orienté avec/sans connexion	AAL3/4
transfert de données	AAL5

TAB. 1.1: Les classes de service de la couche AAL

sion en cellules et assigne la classe de service. La SAR divise l'information en cellules et reassemble les cellules de nouveau. L'UIT a distingué quatre classes de services d'application. À ces quatre classes de service correspondaient quatre classes de protocoles. Cette division a été modifiée en 1993 par le regroupement des classes 3 et 4 et l'ajout de la classe 5 (cf. tableau 1.1) [Pujolle03].

1.4.2 Réseaux LAN sans fil

Comme les ordinateurs portables sont devenus de plus en plus populaires, le besoin de les relier facilement à différents types de réseaux s'est vite fait sentir. Du point de vue de l'utilisateur, la manière la plus facile de relier un tel dispositif à un réseau est d'employer une connexion sans fil. La norme IEEE 802.11 comprend deux générations de réseaux sans fil : les réseaux Wi-Fi 2, qui travaillent à la vitesse de 11 Mbis/s, et les réseaux Wi-Fi 5, qui travaillent à 54 Mbits/s. Les premiers se fondent sur la norme IEEE 802.11b et les seconds sur la norme 802.11a [Pujolle03].

Un aspect essentiel pour les réseaux qui utilisent un canal partagé est de déterminer qui, à un moment donné, est autorisé à mettre ses données sur le support physique. Beaucoup de méthodes ont été conçues pour resoudre ce problème. Le protocole LAN sans fil d'IEEE 802.11 utilise la méthode CSMA/CA (*Carrier Sense Multiple Access/Collision Avoidance*) pour l'accès au support physique. Cette méthode diffère de la méthode CSMA/CD (*CD : Collision Detection*) employé dans l'Ethernet (IEEE 802.3) dans le sens que le système évite les collisions de paquet au lieu de les détecter. Dans CSMA/CA une station écoute le support de transmission, si le support est libre, la station attend une période courte de temps appelée espace inter-frame (*IFS : Inter-Frame Space*) et continue d'écouter le support. Si le support est toujours libre après l'IFS (Les valeurs des différents IFS sont calculées par la couche physique), alors la station peut commencer à transmettre des données. Si le paquet est intact à la réception, la station réceptrice émet une trame ACK qui, une fois reçue par l'émetteur, met un terme au processus. Si la trame ACK n'est pas détectée par la station émettrice (parce que le paquet original ou le paquet ACK n'ont pas été reçus intacts), une collision est supposée et le paquet de données est retransmis après attente d'une durée aléatoire. En cas d'occupation du support lors de l'écoute initiale, la station doit attendre pendant une durée aléatoire avant d'essayer d'initialiser la transmission à nouveau [Pujolle03].

Les stations qui fonctionnent dans un réseau LAN de 802.11 constituent un ensemble de services de base (*BSS : Basic Service Set*). Chaque station et chaque BSS a sa propre

adresse de réseau. Deux modes de fonctionnement existent dans 802.11 : le mode infrastructure et le mode ad-hoc. Le mode infrastructure est défini pour fournir aux différentes stations des services spécifiques, sur une zone de couverture déterminée par la taille du réseau. Les réseaux en mode infrastructure sont établis en utilisant des points d'accès ou AP (*Access Point*), qui jouent le rôle de stations de base pour un BSS. Un réseau en mode ad-hoc est un groupe de terminaux formant un IBSS (*Independent Basic Service Set*), dont le rôle est de permettre aux stations de communiquer sans l'aide d'une quelconque infrastructure, telle qu'un point d'accès ou une connexion au système de distribution [Pujolle03].

Le réseau HiperLAN (*High Performance Radio LAN*) est l'autre normalisation de réseaux sans fil préparée par l'ETSI (*European Telecommunication Standard Institute*). Les bandes de fréquences se situent entre 5.15 et 5.30 GHz. Les vitesses de transfert devraient être comprises entre 19 et 25 Mbits/s. La distance entre les stations et un point d'accès peut atteindre plusieurs centaines de mètres. La modulation est de type GMSK (*Gaussian Minimum Shift Keying*) et la redondance nécessaire pour atteindre qualité typique d'un réseau local est obtenue par un code BCH (*Bose Chaudhuri Hocquenghen*). La technique d'accès au réseau local est une adaptation du CSMA/CD, appelée EY-NPMA (*Elimination Yield-Non Preemptive priority Multiple Access*), qui utilise cinq canaux avec un ordre de priorité. Il existe deux types étendus de la norme HyperLAN, le type 1 et le type 2. Dans [Grass01], une implantation du modem pour HyperLAN et 802.11a a été proposée.

1.4.3 Réseaux FDDI (*Fibre Distributed Data Interface*)

Le réseau FDDI est un réseau LAN ou MAN de type boucle en fibre optique à haute performance. Il utilise une technique d'anneau à jeton (*Token Ring*) et il possède la capacité d'auto-dépannage. La méthode d'accès est similaire à celle du réseau 802.5 version 16 Mbps. Chaque station doit posséder un jeton unique pour émettre puis pour générer un nouveau jeton. Il permet un débit de 100 Mbits/s sur une distance allant jusqu'à 100 kilomètres pour 1000 stations. Le réseau FDDI-II est une version de FDDI adaptée au transfert de données synchrones (par exemple, la voix numérique de type MIC : (modulation par impulsion et codage). Il utilise deux anneaux indépendants en fibre optique attachées trois types de stations : les stations de type A, qui possèdent une double connexion à chaque anneau, les stations de type B, qui possèdent une double connexion à un seul anneau, et celles de type C qui sont des concentrateurs connectés aux deux anneaux et pouvant relier les stations B. FDDI a une structure contrarotative : il permet donc d'assurer une meilleure fiabilité du réseau. En effet, si l'un des anneaux vient d'être coupé accidentellement, le second peut être utilisé. De même, si les deux anneaux viennent à être coupés au même point ils peuvent être reconfigurés de façon à former un nouvel et unique anneau [Pujolle03].

Il n'y a pas de station monitrice, chaque station participe à la surveillance de l'anneau. La distance maximale inter station (2 km), ainsi que la longueur totale de l'anneau FDDI ne permettent plus la synchronisation des stations à partir d'une horloge unique. Chaque station possède sa propre horloge, et une mémoire tampon permet de compenser les écarts entre l'horloge de réception et celle d'émission. C'est la capacité du tampon mémoire qui limite la taille de la trame à 4500 octets.

Les données sont séparées en deux flux, les données urgentes à contrainte de débit (classe synchrone) et les données sporadiques, sans contrainte particulière de débit (classe asynchrone). Une station peut toujours émettre des données synchrones lorsqu'elle possède le jeton. En revanche, elle ne peut émettre des données asynchrones que si le jeton est en avance (jeton temporisé).

La couche physique est scindée en 2 sous-couches. PMD (*Physical Medium Dependent*) adapte les caractéristiques des organes d'émission en fonction du support physique. PHY (*Physical layer protocol*) gère le protocole physique et s'occupe du codage et de la synchronisation. La couche FDDI-MAC est chargée des fonctions habituelles (gestion du jeton, temporisation). Le protocole SMT (*Station ManagemenT*) gère l'insertion et le retrait des stations, la configuration du réseau et le traitement des erreurs.

La couche physique de ce réseau n'utilise pas de codage Manchester (le codage utilisé dans la plupart des réseaux locaux et notamment dans les réseaux Ethernet) pour réduire le débit nécessaire. En effet en codage Manchester il faut doubler le débit pour effectuer le codage car il y a toujours une transition entre deux valeurs du signal pour un élément binaire. C'est le codage par bloc 4B/5B qui est utilisé. Dans ce codage, pour un groupe de 4 bits (un symbole), 5 bits sont tranmis sur le support. Il y a donc 32 combinaisons. 16 combinaisons sont utilisées pour les données et 16 combinaisons sont utilisées pour autre chose, comme par exemple la signalisation. Les bases du protocole FDDI sont très proches de la norme 802.5 (*Token Ring*) [Tanenbaum03]. Une différence majeure entre FDDI et 802.5 concerne l'émission de jeton. En effet, dans FDDI chaque station peut générer un nouveau jeton après la transmission de sa trame. Le format de la trame FDDI est peu différent des trames 802.5 : en plus des trames de données classiques (asynchrones) FDDI permet la transmission de trames synchrones qui sont générées toutes les 125 μs pour les systèmes MIC. Le protocole FDDI utilise un mécanisme de priorité similaire à celui du protocole 802.5 [Pujolle03][Kofman99][Robin99].

1.4.4 Réseaux DQDB (*Distributed Queue Dual Bus*)

Ce réseau est développé par une université australienne et soutenu par Telecom Australia. Il a été normalisé IEEE 802.6 et ISO 8802.6 comme norme de réseau métropolitain. Il a été crée parallèlement à ATM et utilise un format de cellule de 53 octets dont 48 de charge utile. DQDB permet des transferts isochrones (pour des données telles que la voix interactive qui nécessite un intervalle strict entre chaque échantillon) et asynchrones en mode connecté ou non pour des débits compris entre 45 et 155 Mbps.

Comme la figure 1.6 le présente, ce réseau utilise un double bus unidirectionnel. Sur chaque bus, une tête de bus (HoB, *Head of Bus*) génère une trame toutes les 125 μs contenant slots (cellules de 53 octets). Le nombre de slots dépend du débit du réseau. Les têtes de bus sont généralement situées sur une même station. Le premier bit de chaque slot (bit Busy) indique si le slot est libre ou occupé [Pujolle03].

Les stations peuvent lire ou écrire des données au vol dans une cellule mais c'est l'extrémité fixale de bus qui a la fonction d'absorption des trames. Chaque station écrit dans le bus qui correspond à la direction de la station avec laquelle elle veut communiquer, sauf en cas de broadcast où les données sont émises sur les deux bus. Chaque station ayant des données à émettre les dépose dans un slot vide de manière statistique (données asynchrones) ou pré-affectée (pour les données isochrone).

La pile de protocoles de ce réseau est composée de trois couches : la couche LLC (*Logical Link Control*), la couche MAC et la couche physique. La couche MAC comprend un ensemble de fonctions spécifiques à chaque type de transfert : MCF (*MAC Convergence Function*) pour un service asynchrone sans connexion (transfert de données entre ordinateurs), COF (*Connection Oriented Function*) pour les transferts asynchrones en mode connecté (applications conversationnelles) et, enfin, ICF (*Isochronous Convergence Function*) pour le transfert isochrone (débit constant pour la voix et la vidéo). L'accès au support partagé est géré par deux entités, celle qui s'occupe du trafic isochrone (*PA :*

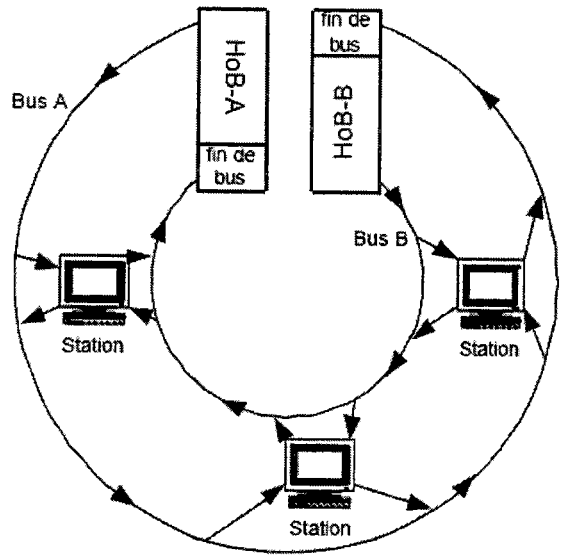


FIG. 1.6: Le réseau DQDB

Pre-arbitrated) et celle qui s'occupe du trafic asynchrone (*QA : Queue Arbitrated*).

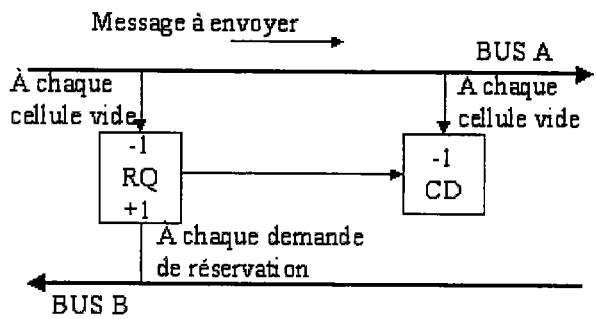


FIG. 1.7: La méthode d'accès de DQDB

La trame DQDB est composée de cellules préarbitrées, suivies de cellules arbitrées par la file d'attente. Les cellules QA utilisent une technique d'accès à deux compteurs, RQ (*Request Count*) et CD (*Count Down*). Les cellules contiennent deux bits spécifiques E (dont la valeur 0 indique une cellule vide) et R (dont la valeur 1 indique une cellule réservée). Une station désirant émettre fait une requête de réservation sur le bus allant dans la direction opposé à la station cible. Pour situer la station cible, elle lui adresse un message sur les deux bus, et en fonction du bus d'où arrive la réponse, elle sait de quel côté elle est. Si une station fait une requête sur le bus B pour obtenir N slots sur le bus A, la station suivante (et donc celle qui la précède dans l'autre sens) laisse passer N slots vide sur le bus A avant de déposer ses données dans le slot vide N+1. Lorsque une station veut émettre des données, elle attend que passe un bit R égal à 0 pour réserver la cellule correspondant et pour attribuer la valeur 1 au bit R. Ainsi elle transfère la valeur

du compteur RQ dans le compteur CD. Le compteur RQ est remis à 0 et est incrémenté de 1 à chaque fois qu'un bit R passe à 1. Pendant ce temps, le compteur CD est décrémenté de 1 à chaque fois qu'un bit E passe à la valeur 0, indiquant que la cellule est destinée à une station située à sa droite (cf. figure 1.7). Enfin, lorsque ce compteur indique la valeur 0, la prochaine trame dont le bit E est égal à 0 est la trame dans laquelle la station pourra enfin émettre ses données [Pujolle03].

1.4.5 Réseaux Ethernet haut débit

Ces réseaux sont basés sur la norme IEEE 802.3u que la plupart des spécialistes réseau appellent *Fast Ethernet*. Ils utilisent les anciennes spécifications des réseaux Ethernet classiques comme par exemple le format de la trame (cf. figure 1.8), les techniques et les interfaces d'accès.

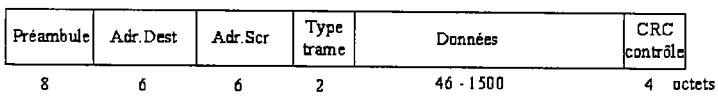


FIG. 1.8: Le trame Ethernet

Ils utilisent la technique CSMA/CD pour l'accès au support partagé mais la fenêtre de collision (temps minimal pendant lequel une station émettrice doit écouter le réseau pour détecter la collision la plus tardive) est réduite à 5,12 μs, ce qui fait un silence inter-trame (IFG : *InterFrame Gap*) de 0,96 μs (96 bits). Ceci induit de fortes contraintes sur le temps de propagation du signal et donc sur la distance maximale entre les deux stations les plus éloignées du réseau : la longueur d'un segment ne peut excéder 100 mètres.

Le temps d'un bit varie de 100 ns à 10 ns. On doit donc changer de type de câble parce que pour un débit multiplié par dix, la longueur de câble doit être divisée par dix sur le même câble. C'est pourquoi il existe trois types de réseaux Ethernet à haut débit selon le type de câble utilisé : le système 100Base-T4, le système 100Base-TX et le système 100Base-FX .

Le système 100Base-T4 utilise quatre paires torsadées. Une paire est affectée en permanence à la transmission de signaux dans le sens hub vers station et une seconde paire dans le sens station vers hub. L'affectation des deux autres paires est associée dynamiquement au sens de transmission courant des signaux entre le hub et la station. Afin d'obtenir le débit de 100 Mbits/s sur ce type de câble et sur 100 m, une technique d'encodage des signaux différente du code Manchester est utilisé. Dans cette technique, les signaux sont codés non pas de façon binaire (0 ou 1) mais de façon ternaire (un signal est défini sur trois états : 0,1 ou 2). Ainsi, avec trois paires torsadées et trois états par signal codé et par paire on peut représenter, donc transmettre, 27 symboles différents. Cela permet la transmission de 4 bits d'information à chaque impulsion d'horloge. En effet, avec une horloge à 25 MHz, on peut transmettre à 100 Mbits/s dans une direction. Le système 100Base-TX est basé sur un câble à deux paires torsadées de catégorie 5 qui peut supporter des signaux d'horloge jusqu'à 125 MHz. Une paire est affectée à chaque sens de transmission. On utilise un codage par bloc de type 4B /5B. Le dernier type de câblage possible, 100Base-FX, utilise deux fibres optiques multimodes, une dans chaque sens de transmission : c'est un réseau duplex symétrique à 100 Mbits/s. Ainsi, l'utilisation de la fibre optique autorise la transmission de signaux à 100 Mbits/s sur des distances de 2 km.

OC1	51.84 Mbits/s
OC3	155.52 Mbits/s
OC9	466.56 Mbits/s
OC12	622.08 Mbits/s
OC18	933.12 Mbits/s
OC24	1244.16 Mbits/s
OC36	1866.24 Mbits/s
OC48	2488.32 Mbits/s
OC96	4976.64 Mbits/s
OC192	9953.28 Mbits/s

TAB. 1.2: Les niveaux SONET

La norme gigabit Ethernet a été spécifiée en 1998 sous la désignation IEEE 802.3z. Le Gigabit Ethernet fonctionne en *full-duplex* dans le mode *switch-to-switch* (de commutateur à commutateur) et dans le mode *switch-to-end-station* (de commutateur à station) et en *half-duplex* pour les stations raccordées directement à un hub. Pour maintenir un diamètre de réseau suffisant en half-duplex (200 mètres), la fenêtre de collision a été modifiée, la trame minimale étant portée à 64 octets. l'IFG (*Inter Frame Gap*) reste à 96 bits.

1.4.6 Normes SONET et SDH

Ce sont les protocoles de la couche physique qui ont été définis au départ pour transporter un grand nombre de communications téléphoniques sur un support physique (fibre optique). SONET (*Synchronous Optical Netwok*) a été normalisée par l'ANSI, l'organisme de normalisation nord-américain et SDH (*Synchronous Digital Hierarchy*) a été normalisée par l'UIT-T. SDH est actuellement très utilisée : elle constitue même le standard d'ATM au niveau physique.

Un débit de 51.84 Mbits/s forme le premier niveau STS-1 (*Synchronous Transport Signal level 1*) et les niveaux supérieurs sont des multiples du niveau 1. Chaque 125 μs, une trame est émise et la longueur de cette trame dépend de la vitesse de l'interface du support physique. Le tableau 1.2 montre les différents niveaux de SONET classés selon la vitesse du support optique OC (*Optical Carrier*).

Les différences entre SONET et SDH sont mineures au point que certains niveaux de SDH et SONET sont compatible. Ainsi, avec certaines options, les communications et le transfert d'informations entre les réseaux de SONET et de SDH sont possibles. Les vi-

STM1	155.52 Mbits/s
STM3	466.56 Mbits/s
STM4	622.08 Mbits/s
STM6	933.12 Mbits/s
STM8	1244.16 Mbits/s
STM12	1866.24 Mbits/s
STM16	2488.32 Mbits/s
STM32	4976.64 Mbits/s
STM64	9953.28 Mbits/s

TAB. 1.3: Les niveaux SDH

tesses de transmission pour les réseaux sont semblables, bien que l'unité de transmission de base de SDH, appelé STM-1 (*Synchronous Transport Module level 1*), soit envoyée à 155,52 Mbps, et l'unité de transmission de base de SONET le soit à 51,84 Mbits/ps. En fait, en STM-1 trois signaux de SONET STS-1 sont multiplexés ; ainsi le SONET STS-3 est l'équivalent de SDH STM-1. La raison d'avoir pour STM-1 et STS-3 une vitesse de 155,52 Mbits/s et pour STS-12 et STM-4 une vitesse de 622,08 Mbits/s est assez évidente : ce sont les vitesses de transmission d'ATM par conséquent, donc les cellules d'ATM peuvent être envoyées en utilisant SONET ou SDH. Le tableau 1.3 représente les niveaux de SDH.

Puisque SDH est synchrone, les trames sont produites chaque 125 μs (dans STM-1) même s'il n'y a aucune donnée à envoyer. On observe facilement que 8000 trames par seconde, c'est equivalent du débit (64 Kbits/s) des systèmes de téléphonie PCM ou MIC (modulation par impulsion et codage). Une trame SDH est composée de 810 octets.

1.5 Équipements de réseaux

Il existe deux groupes principaux dans les équipements de réseaux : les équipements au cœur des réseaux et les équipements dans les machines des utilisateurs. En d'autres termes, il existe deux types d'équipements : ETTD et ETCD. L'ETTD (équipement terminal de transmission de données) est l'équipement sur lequel l'utilisateur travaille et par lequel il veut émettre ses données. L'ETCD (équipement terminal de circuit de données) est placé à chaque extrémité du support de transmission. Son rôle est d'adapter le signal à transmettre aux caractéristiques du support physique. Dans la section suivante, nous décrivons les routeurs comme l'équipement le plus important au cœur des réseaux. Nous nous concentrons ensuite particulièrement sur les interfaces de réseau des terminaux utilisateurs [Pujolle03].

1.5.1 Équipements au cœur des réseaux

Les routeurs sont les dispositifs qui se chargent de choisir le chemin qu'un message va emprunter. Ils sont de plus capable de manipuler les données (qui circulent sous forme de datagrammes) afin de permettre le passage d'un type de réseau à un autre (contrairement à un dispositif de type pont). Les routeurs ont donc la capacité de fragmenter les paquets de données pour permettre leur circulation sur différents réseaux. Certains routeurs sont capables de créer des cartes des itinéraires à suivre (tables de routage) en fonction de l'adresse visée grâce à des protocoles dédiés à cette tâche.

Les premiers routeurs étaient de simples ordinateurs ayant plusieurs cartes réseau (on parle de machines multihôtes) dont chacune était reliée à un réseau différent. Les routeurs actuels sont pour la plupart des matériels dédiés à la tâche de routage. Un routeur possède plusieurs interfaces réseau, chacune connectée sur un réseau différent. Un routeur possède ainsi autant d'adresses pour un réseau IP que de réseaux différents sur lesquels il est connecté.

Il existe deux types principaux de routeurs :

- Les routeurs de type « vecteur de distance » (*distance vector*) établissent une table de routage recensant le coût (en termes de nombre de sauts) de chacune des routes, puis transmettent cette table aux routeurs voisins. À chaque demande de connexion le routeur choisit la route la moins coûteuse.
- Les routeurs de type « état de la liaison » (*link state routing*) écoutent le réseau en continu afin de recenser les différents éléments qui l'entourent. À partir de ces informations chaque routeur calcule le plus court chemin (en temps) vers les routeurs voisins et diffuse cette information sous forme de paquets de mise à jour. Chaque routeur construit enfin sa table de routage en calculant les plus courts chemins vers tous les autres routeurs.

1.5.2 Équipements des terminaux

La carte réseau ou NIC (*Network Interface Card*) est le matériel le plus connu permettant de connecter un ordinateur au réseau. Comme pour les autres périphériques, la carte réseau est dirigée par le CPU (*Central Processing Unit*), c'est à dire que le CPU contrôle toutes les réceptions et les transmissions de paquets. Aujourd'hui, avec les réseaux haut débit, les capacités des CPU et les bus ne permettent pas d'utiliser toute la bande passante disponible. Pour cette raison, les cartes réseau récentes en utilisant des techniques d'optimisation exécutent une partie des tâches du CPU [Comer04]. On peut citer par exemple, les techniques suivantes :

- Reconnaissance et filtrage de trames sur la carte

  La carte vérifie les adresses de destination de toutes les trames, choisit les trames avec les adresses *unicast* et *broadcast* correspondantes, et abandonne les autres trames sans interruption du CPU.

- Stockage de trames sur la carte

  Le trafic réseau n'est pas toujours dans une situation stable, et il est même parfois éclaté. De plus, les resources partagées telles que les bus ne sont pas toujours disponibles. Pour ces raisons, une mémoire suffisante permet à la carte de récupérer les trames lors d'un trafic éclaté. La carte peut accepter et stocker le nouveau paquet et

en même temps envoyer les paquets précédents vers la machine.

- Accès direct à la mémoire ou DMA (*Direct Memory Access*)

  À l'origine, cette technique a été inventée pour optimiser le transfert de données vers les disques. Cependant, elle peut être utilisée par tous les périphériques qui transférent de larges volumes de données vers la mémoire. Tout d'abord, le CPU envoie un message à la carte de réseau indiquant l'adresse de la mémoire qui doit être utilisée pour stocker les paquets. Ensuite, la carte transfère le paquet vers la mémoire. Enfin, la carte génère une interruption pour informer le CPU de la fin de l'opération [Comer04].

Grâce aux méthodes d'optimisation, on peut trouver aujourd'hui, des cartes réseau qui fonctionnent à des débits d'environ 1 Gbits/s. Cependant avec ces cartes, beaucoup de fonctions sont traitées directement par le CPU principal. Par conséquent, il ne reste plus de temps au CPU pour traiter les protocoles de couches supérieures. Cela justifie le développement d'architectures dans ces interfaces permettant de décharger le CPU de la machine le plus possible. Afin de répondre à ces exigences ainsi qu'aux exigences de flexibilité concernant les protocoles, il est essentiel de trouver une nouvelle stratégie de traitement des protocoles (et le matériel correspondant) pour traiter les protocoles dans les machines utilisateurs.

1.6 Traitement des protocoles

1.6.1 Caractéristiques du traitement de protocoles

Le traitement de protocoles est un terme générique définissant n'importe quel type de traitement impliqué dans un réseau informatique, par exemple, l'expédition de paquets dans les routeurs, la génération de paquets dans les terminaux, le filtrage de paquets dans les pare-feu, le chiffrement de paquets dans les équipements de sécurité ou même la modulation et le codage dans les couches physique et liaison de données. Un point important est que le traitement d'un paquet ne dépende pas d'autres paquets, excepté dans quelques cas particuliers. Toute l'information nécessaire au traitement est présenté en un seul bloc dans les en-têtes de chaque paquet. Pour comprendre les caractéristiques du traitement du protocole, il est donc nécessaire de diviser les opérations de traitement du protocole en catégories plus restreintes.

Les opérations principales du traitement d'un protocole sont classées en deux groupes : le traitement de type contrôle et le traitement de données. Le traitement des données concerne des opérations de transmission de données entre les nœuds du réseau. Les opérations typiques incluent le transfert de données entre les mémoires, la génération de paquets, la vérification de données de contrôle, l'ordonnancement de paquets, les opérations telles que le codage et le décodage ainsi que les opérations au niveau bit de la couche physique telles que la modulation. D'autre part, le traitement de type contrôle comprend la gestion des connexions et la régulation du transfert de données par l'échange d'informations de contrôle et de synchronisation. Du côté de la réception, divers champs de contrôle sont analysés et les opérations correspondantes sont effectuées. Les opérations de traitement de données avec une structure de communication pipeline exigent une performance adaptée au temps réel, tandis que les opérations de type contrôle sont moins critiques au niveau de la performance. Le tableau 1.4 présente cette classification pour un réseau sans fil typique [[Rabaey00].

Couches	Opérations de contrôle	Opération de données
Application Transport	Algorithmes de localisation	Chiffrement Compression
Réseau	Gestion de la topologie Routage Classification des paquets	
Liaison MAC	Algorithmes d'accès au canal	Codage Décodage
Physique	Synchronisation Temporisateurs Fragmentation	Modulation Démodulation

TAB. 1.4: Les couches et les opérations dans un réseau sans fil

Une tâche très importantes pour caractériser le traitement de protocoles est d'identifier les opérations fréquentes et communes (ou peu différentes) parmi les divers protocoles cibles. Certaines de ces opérations exigent un traitement critique du point de vue temporel. Elles peuvent être implantées dans les modules matériels dédiés. En conséquence celle qui demandent plus de flexibilité peuvent être implantées dans des modules logiciels spécialisés tels que les cœurs de processeurs.

Notre travail concerne plutôt le traitement de protocoles dans les terminaux de réseau. Les terminaux dans un réseau peuvent être de différents types. On peut citer, par exemple, les ordinateurs de bureau ou portables, les téléphones IP, les systèmes de video conférence ou les serveurs de fichiers. Les besoins en traitement sont donc différents : par exemple, dans un système fonctionnant avec des données de type *stream* comme la voix et le vidéo, la latence est plus importante que dans un système de transfert de fichiers. Par contre, le transfert de fichiers demande un débit sortie élevée (*high throughput*).

Il existe deux différences principales entre le traitement de protocoles dans un terminal utilisateur et dans les nœuds au cœur du réseau, tels que les routeurs : dans les terminaux, le nombre de connexions est moindre et elles sont plus stables ; en outre, les terminaux doivent traiter plus de couches que les routeurs. La première différence simplifie les tâches de traitement dans un terminal tandis que la deuxième la complique. En d'autres termes, dans les terminaux le problème vient du volume de traitement alors que dans les routeurs il vient plutôt du volume de données.

1.6.2 Fonctions de base du traitement de protocoles

Dans cette section, nous introduisons les fonctions de base du traitement de protocoles qui sont communes à un grand nombre de protocoles. En effet, l'étude de ces fonctions peut nous aider à trouver la justification et les principaux aspects d'une architecture de processeur adaptée au traitement de protocoles [Comer04].

Nous étudions ici les dix fonctions suivantes, opérations communes dans le traitement de protocoles des réseaux informatiques :

- **Traitement numérique de données pour la transmission**

  La transmission de bits sur un canal de communication demande certaines opérations afin de s'assurer qu'un bit à 1 envoyé à une extrémité arrive à 1 de l'autre côté. Certaines limites physiques existent pour chaque type de canal. Pour cela, les protocoles de la couche physique utilisent des techniques de traitement de données et des signaux, telles que la modulation, la démodulation et le filtrage. La sélection des techniques requises pour ces opérations dépend très fortement des caractéristiques du canal.

- **Recherche d'adresse pour acheminer les paquets vers leur destination**

  Cette fonction consiste à rechercher dans un tableau, les adresses de destination des paquets ou les trames. C'est par exemple, la recherche de l'adresse MAC d'une trame dans un terminal ou la recherche d'adresses IP dans un routeur pour envoyer les paquets vers leur destination. La méthode de recherche et la puissance de traitement requise dépendent du point dans le réseau où la recherche se fait. Par exemple, un pont exige un assortiment exact de l'adresse MAC tandis que, dans un routeur un assortiment du préfixe le plus long est exigé.

- **Détection et correction d'erreurs**

  Lorsqu'un paquet traverse un réseau, des bits erronés sont produits par différentes sources d'erreurs. Afin de permettre la détection ou la correction des erreurs du côté du récepteur, l'émetteur ajoute des bits complémentaires aux paquets. Les bits ajoutés ne changent pas beaucoup le volume de données à envoyer, mais la détection et surtout la correction d'erreur en temps réel du côté du récepteur exige une puissance de traitement assez élevée. Les méthodes les plus communes sont le code CRC (*Cyclic Redundancy Check*) et la somme de contrôle (*checksum*). Afin d'atteindre la performance requise dans la plupart des réseaux, le CRC est effectué par des unités matérielles dédiées.

- **Fragmentation et réassemblage**

  Chaque réseau imposant une taille maximale aux paquets ou cellules qu'il achemine la plupart des protocoles offrent la possibilité de fragmenter les messages ou paquets de taille limitée au niveau de l'émetteur et de réassembler les fragments au niveau du récepteur. Les charges utiles maximales vont généralement de 48 octets pour les cellules ATM à 65515 octets pour les paquets IP, mais dans les couches supérieures elles sont souvent plus importantes. Fragmenter les messages en morceaux plus petits est plus facile que de reconstituer le message d'origine. Le récepteur doit conserver les fragments jusqu'à ce qu'ils soient tous arrivés. Cette opération exige un système de tampons et un mécanisme de gestion des cas où certains fragments seraient perdus. Le choix de la taille maximale de ces tampons est une décision architecturale importante.

- **Démultiplexage de trames et protocole**

  Démultiplexer signifie trouver un protocole (ou un élément de protocole) dans la couche adéquate selon le type de trame ou le paquet. Par exemple, lorsqu'une trame arrive, le type de trame détermine le protocole requis, comme ARP ou IP. Ainsi le

type de datagramme IP détermine le protocole de la couche supérieure tel que UDP ou TCP. En effet, l'émetteur insère le type de données dans les paquets et le récepteur utilise cette information pour démultiplexer les paquets entre les protocoles dans chaque couche.

- **Classement de paquets**

  Le classement s'effectue suivant une ou plusieurs valeurs contenues dans l'en-tête du paquet (exemple : adresse source destination, port source destination, protocol ID, ...). Contrairement au démultiplexage, qui utilise une méthode traditionnelle basée sur les couches, la classification effectue plusieurs vérifications de l'en-tête en même temps. En effet, au niveau du démultiplexage, une compression de différentes couches se fait par classement. Pour classer un paquet, une comparaison au maximum est nécessaire pour chaque champ du critère de classement (par exemple en notation de langage C, on peut avoir un critère comme : `if ((type_trame==0x0800) && (type_IP==6) && (port_TCP==80)) then classifie(paquet))`.
  Cependant, si le nombre maximum de comparaisons pour un classement est fixe, mais le nombre moyen de comparaisons dépend de l'ordre des tests. Le nombre minimum est obtenu quand la comparaison débute par le test qui élimine le nombre maximum de paquets. Dans une implantation de traitement de protocoles sur les processeurs conventionnels, le classement se fait de façon logiciele mais il existe des modules dédiés pour faire cette opération dans des architectures spécifiques. Certains processeurs réseau tels que les processeurs Intel, utilisent un langage spécial, appelé NCL (*Network Classification Language*), pour la classification [Comer04].

- **Gestion de files d'attente**

  Les systèmes de traitement de paquets fonctionnent d'une manière *store-and-forward* : quand certains paquets sont en train d'être traités, les autres sont en attente dans une file d'attente. La gestion des files d'attente est liée aux méthodes de stockage et de sélection des paquets en vue de leur traitement. De manière générale, une file d'attente a une structure FIFO (*First-In-First-Out*). La taille de la FIFO est naturellement déterminée par la longueur maximum des salves du trafic. Dans les systèmes avec plusieurs interfaces, on peut avoir une FIFO commune ou plusieurs FIFO avec un mécanisme de priorité. La gestion des files d'attente se complique surtout dans les systèmes avec différentes règles de priorité. Le choix des règles de priorité dépend des caractéristiques du réseau et des charges utiles des paquets. En cas de saturation des tampons (à cause de leur taille limitée), les systèmes utilisent différents mécanismes pour abandonner certains paquets entrants et respecter les critères de QoS (*Quality of Service* : qualité de service).

- **Authentification**

  Il existe des systèmes de traitement des paquets tels que les terminaux sûrs qui utilisent des mécanismes d'authentification pour valider l'identité de l'émetteur, l'intégrité des données et leur confidentialité. L'authentification est basée sur les algorithmes de chiffrement des données. Si, la charge ajoutée aux paquets à cause de ces algorithmes est relativement faible, le traitement requis pour l'authentification est élevé. Il existe deux raisons à cela : premièrement, l'authentification est effectuée sur des paquets entiers ; deuxièmement, les caractéristiques des algorithmes de

chiffrement et de déchiffrement demandent un traitement intensif. Ces opérations sont effectuées dans la plupart des processeurs réseau par des modules dédiés.

- **Mesurer le trafic**

  Des mesures du trafic sur le réseau peuvent être utilisées dans les systèmes d'analyse de trafic. Pour faire ces mesures, une copie de chaque trame qui traverse le réseau est vérifiée et les informations concernant le trafic (par exemple : le nombre de paquets, le pourcentage de trames à diffusion, le pourcentage d'utilisation du réseau) sont mises à jour. Ces mesures sont effectuées à la fois aux nœuds intermédiaires du réseau et au niveau des terminaux et peuvent être utilisées par exemple pour vérifier la réalisation réelle d'un contrat de service entre un client et un fournisseur de réseau à distance. La réalisation des mesures en temps réel est un aspect important de ces mesures, ce qui demande une puissance de traitement élevée.

- **Gestion des temporisateurs**

  La gestion du temps est un aspect fondamental dans le traitement des protocoles. La plupart des protocoles utilisent des mécanismes de requête et d'attente pour leur fonctionnement. Le temps d'expiration d'une attente est spécifié par un temporisateur. La diversité et la nature dynamique des valeurs du temps d'expiration, ainsi que le traitement simultané de fonctions de protocoles gênèrent des problèmes dans la gestion des temps dans le traitement de protocoles. Ainsi, la sélection de différentes granularités du temps dans les temporisateurs est très importante.

1.6.3 Tâches de traitement dans le terminal d'utilisateur

Dans cette section nous décrivons les tâches de traitement dans les terminaux de réseaux. Deux remarques importantes doivent être prise en compte. Premièrement, une attention particulière doit portée aux tâches accomplies sur les paquets entrants, car les données arrivent de manière non équilibrée et un terminal reçoit plus de données qu'il n'en transmet. Par conséquent, il est plus intéressant d'optimiser le processus de réception des paquets que celui de transmission des paquets. Deuxièmement, les tâches décrites sont pour le cas particulier où le terminal utilise les protocoles IP et TCP (ou UDP) comme protocoles des couches 3 et 4 respectivement.

Lorsqu'un paquet arrive au niveau terminal du utilisateur, il doit être manipulé tout d'abord par la couche physique. Les protocoles de couche physique diffèrent beaucoup selon le support de transmission, comme par exemple le câble coaxial, la fibre optique ou les ondes radio. En outre, la distance et le débit influencent le protocole. Dans cette couche les tâches typiques sont [Comer00][Busby00] :

- Modulation/démodulation
- Codage/décodage
- Filtrage
- Entrelacement

Le train de bits (*bitstream*) de la couche physique est ensuite fourni à la couche liaison de données. Dans les terminaux qui utilisent Ethernet comme couche liaison de données et physique, les tâches typiques à effectuer sont :

- Vérifier l'adresse de destination
- Déterminer la longueur du paquet

- Calculer et vérifier une somme
- Démultiplexer le train de paquets selon le protocole de la couche 3
- Décompresser l'en-tête et/ou les données

Quand l'en-tête de la couche 2 a été extrait, le paquet est envoyé à la couche réseau. Ici IPv4 et IPv6 sont les protocoles dominants de la couche réseau depuis une longue période. Dans le cas de réseaux Ethernet, un protocole de résolution d'adresse (ARP) est employé pour retrouver l'adresse Ethernet correspondante à une adresse d'IP.

ARP peut également être vu comme une partie de la couche réseau, bien qu'il soit souvent attribué à la couche "2,5" : Parce qu'il fournit un service à IP, il est un protocole de couche 2 ; parc qu'il utilise des services fournis par Ethernet, il est un protocole de couche 3. De même, le protocole de contrôle de message d'internet (ICMP) et le protocole de gestion (IGMP) pourraient être considérés comme appartenant à la couche 3,5. Ils appartiennent cependant à la couche réseau. Dans la couche réseau, les tâches de traitement sont les suivantes [Comer00][Busby00] :

- Contrôler le code opération de ARP
- Mettre à jour la table ARP
- Déclencher la réponse ARP
- Vérifier l'adresse IP destination
- Vérifier la version d'IP
- Calculer la somme contrôle de l'en-tête
- Demultiplexer le train de paquets dépendant du protocole de couche 4
- Vérifier le type et la version d'ICMP
- Déclencher la réponse ICMP
- Vérifier le type et la version d'IGMP

Dans la couche 4, tout comme IP pour la couche 3, TCP et UDP sont devenus totalement inconfortables. Ils représentent deux types différents de protocoles de transport. TCP est orienté connexion et fournit un train d'octets entre l'expéditeur et le récepteur, ces octets devant être reçus dans l'ordre où ils ont été envoyés. UDP est un mode sans connexion et fournit un service efficace d'échange de datagrammes entre l'expéditeur et le récepteur. TCP est typiquement employé pour la communication point à point et le transfert de données de manière fiable. D'autre part, UDP est employé pour l'émission et le multicast des communications en temps réel et sensibles au retard aux protocoles lourds comme TCP ne sont pas adaptés. Le traitement de la couche transport est semblable à celui des couches 1 à 3, sauf que TCP requiert de la manipulation de l'état puisqu'il offre des connexions fiables qui sont maintenues avec des nombres de séquences et des acquittements. Les tâches sont de deux sortes : le calcule des sommes de contrôle qui requièrent un traitement intensif orienté calculs, et le reste, qui exige un traitement orienté contrôle [Comer00][Busby00] :

- Vérifier la somme de contrôle
- Démultiplexage de trains de paquets et livraison à l'application
- Déclencher un paquet d'acquittement
- Gestion de l'état de connexion
- Mise à jour de la taille de fenêtre (en TCP)

1.6.4 Parallélisme dans le traitement de protocoles

Le trafic réseau est une catégorie de connections (flux) indépendantes. L'indépendance entre les flux différents peut être exploitée par parallélisme entre les tâches de traitement des paquets. De plus, les paquets traversent les différentes étapes basiques de manière

séquentielle : démultiplexage, traitement et acheminement (*forwarding*). On peut donc utiliser du pipeline pour améliorer les performances.

Un point important, dans le traitement parallèle de protocoles basé sur les modèles multiprocesseurs à mémoire partagée, est la façon d'utiliser les ressources partagées. L'autre point est le niveau de granularité de parallélisation du traitement. Un parallélisme à gros grain génère un léger surcoût mais aux dépens de moins parallélisme. L'inverse es produit pour un parallélisme à grain fin.

Dans [Björkman98] sont décrites quatre manières de paralléliser le traitement de protocoles qui constituent un bon aperçu de toutes les approches qui ont été proposées. Un jeu de processeurs identiques a été employé pour exécuter le traitement des protocoles. Le point clé est comment répartir le traitement entre les processeurs. Le tableau 1.5 représente les caractéristiques des quatre principales méthodes.

- **Un processeur par tâche**

  Le choix d'un processeur par tâche implique que chaque processeur exécute une ou plusieurs tâches d'un ou de plusieurs protocoles. Cette méthode a une granularité fine mais exige que les paquets et les états de connexion puissent être partagés par plusieurs processeurs. Dans cette méthode, un contrôleur identifie pour chaque paquet les exigences du traitement et envoie le paquet vers les processeurs concernés par ces tâches. Cette approche est trop coûteuse du point de vue de l'implantation. Les avantages viennent du fait que plusieurs tâches peuvent être exécutées en parallèle et qu'une réduction de la latence de traitement d'un paquet est possible.

- **Un processeur par message**

  La méthode utilisant un processeur par message, suggère d'avoir un processeur pour chaque paquet qui arrive. Un cas particulier et simple est celui où chaque paquet de données est assigné à un processeur de traitement de paquets de données et les paquets de contrôle sont assignés au processeur de traitement des paquets de contrôle. En général, cette méthode a les avantages du parallélisme gros grain et un bon équilibrage des charges. Elle permet également une flexibilité dans le nombre de processeurs assignés pour le traitement de protocoles. Si le réseau n'est pas fortement utilisé pendant une certaine période, quelques processeurs peuvent être assignés à d'autres tâches pendant ce temps.
  En cas de forte utilisation ou d'augmentation du débit, le niveau de parallélisme peut être facilement augmenté par l'ajoute de processeurs. C'est le contraire du cas d'un processeur par tâche où le niveau de parallélisme est limité par la possibilité d'extraire des fonctions essentiellement indépendantes. Les inconvénients sont que les états de la connexion doivent être mis en commun entre plusieurs processeurs, particulièrement pour des protocoles avec un état de connexion complexe, comme TCP. Ceci rend l'approche d'un processeur par message coûteuse.

- **Un processeur par connexion**

  Un processeur par connexion signifie que chaque connexion est assignée à un processeur. Après l'établissement de chaque connexion, les programmes correspondants de chaque connexion sont transférés vers la mémoire locale du processeur qui doit manipuler toutes les tâches pour tous les paquets qui appartiennent à cette connexion. L'avantage par rapport à l'approche précédente est que l'état de la connexion n'a pas besoin d'être mis en commun entre plusieurs processeurs. L'in-

Approche	Les données partagés	Inconvénients	Avantages
processeur-par-message	l'état de la connexion	l'état de la connexion partagée	flexibilité, bon équilibrage de charge
processeur-par-connexion	non	saturation	pas de données partagées
processeur-par-protocole	paquet	paquet partagé	possibilité de spécialisation
processeur-par-tâche	paquet et l'état de connexion	paquet et l'état de la connexion partagés	réduction de la latence

TAB. 1.5: Caractéristiques des approches de parallélisme

convénient est qu'une connexion avec beaucoup de salves de paquets saturent un processeur tandis que d'autres processeurs peuvent être libres et ne peuvent pas décharger le processeur saturé car ils ne peuvent pas accéder à l'état de la connexion. Ceci se produit notamment lorsque les connexions ont beaucoup de salves comme par exemple lors du transfert de fichiers. D'autre part, le coefficient d'utilisation des processeurs dépend du nombre de connexions.

- **Un processeur par protocole**

  Un processeur par protocole est une approche pour laquelle chaque processeur est assigné à un protocole. Les avantages sont que les processeurs peuvent être spécialisés et la taille du code peut être petite pour chaque processeur. Le problème est que les paquets doivent se déplacer de processeur en processeur et la granularité est trop petite pour des protocoles avec de petits besoins de traitement.

Toutes ces approches, sauf le processeur par connexion, ont des surcoûts et une possibilité de blocage parce que les processeurs doivent utiliser plusieurs ressources partagées. Une implantation multiprocesseurs a été proposée dans [Björkman98]. L'approche d'un processeur par message s'est trouvée être la meilleure dans l'environnement utilisé.

Il existe aussi des méthodes pipeline pour traiter les protocoles. Puisque les modèles de réseaux sont constituées de différentes couches traitées logiquement en série, chacune des couches fonctionne indépendamment des autres et fournit des services à la couche supérieure. Dans une méthode pipeline, on peut assigner les tâches de chaque couche à un processeur. L'avantage de cette méthode est la facilité d'implantation mais, comme pour les autres méthodes pipeline, le débit sortie (*throughput*) global du système est limité par la vitesse du processeur de couche le plus lent.

Il existe deux types de traitement de paquets dans un système de traitement de protocoles. Dans le premier cas, les paquets (ou une partie des paquets) sont traités sans mémorisation, qu'on appelle traitement en ligne. L'avantage de cette méthode est la réduction du temps d'accès à la mémoire mais elle exige une bonne synchronisation et une vitesse de traitement élevée. Dans le deuxième cas, les paquets sont enregistrés puis sont traités. De cette manière, le système peut effectuer les opérations de traitement et le stockage dans un pipeline.

En résumé, toutes les manières présentées peuvent augmenter potentiellement la performance du système de traitement de protocoles. Comme pour toutes les méthodes de traitement parallèle afin d'atteindre la performance maximale il faut utiliser le parallélisme de manière à ce que l'utilisation des processeurs soit maximale et que la synchronisation nécessaire soit minimale.

1.7 Conclusion

Dans ce chapitre, les réseaux ont été décrits de manière générale. Les trois modèles standards de réseaux ont été introduits et leurs principes de fonctionnement expliqués. Les différents types de réseaux, notamment les réseaux haut débit, ont ensuite été introduits.

L'étude des protocoles et de leurs fonctions montre qu'il existe des opérations fréquentes ou peu différentes entre les divers protocoles. L'identification de ces opérations peut tracer la voie à suivre pour mettre au point des architectures dédiées pour le traitement des protocoles.

Il existe deux catégories principales dans le traitement des protocoles : le traitement de protocoles de type contrôle et le traitement de protocoles de données. Chaque type possède ses caractéristiques propres.

Les opérations de traitement dans les routeurs et les commutateurs sont différentes en comparaison avec les opérations dans les terminaux. Dans les terminaux utilisateurs, le nombre de connexions est moindre et celles-ci sont plus stables que dans les routeurs. D'autre part, les terminaux doivent traiter plus de couches que les routeurs. La première différence simplifie les tâches de traitement dans les terminaux tandis que la deuxième la complique.

Dans la continuation de ce chapitre, en se basant sur la classification faite dans [Björkman98], les différentes méthodes de parallélisme ont été introduites et leurs avantages et inconvénients ont été discutés. Toutes les approches présentées permettent d'augmenter potentiellement la performance du système de traitement de protocoles. Cependant, il est souhaitable de choisir une combinaison de méthodes réalisant un compromis entre les surcoûts de chaque technique (comme la communication entre les processeurs et la synchronisation) et le facteur d'utilisation des ressources.

Chapitre 2

Architectures de processeurs

2.1 Introduction

Après une étude du domaine d'application des réseaux et de leurs caractéristiques, nous étudions dans ce chapitre différentes architectures de processeurs. Globalement, on peut distinguer deux types d'architectures de processeurs : les processeurs généraux et les processeurs spécialisés. Les processeurs généraux utilisent une architecture générale et un jeu d'instructions assez général qui leur permet d'avoir une bonne performance sur la plupart des applications. Les processeurs spécialisés sont conçus exclusivement pour un domaine d'application défini. Il existe plusieurs approches architecturales dans le domaine des architectures de processeurs. Ces approches se partagent en deux classes principales : les approches parallèles et les approches non-parallèles.

Ce chapitre se divise en trois parties. Dans la première partie, nous nous intéressons aux architectures scalaires. Nous décrivons les architectures générales CISC et RISC de manière comparative. Dans la deuxième partie, nous présentons les méthodes architecturales parallèles. Tout d'abord, nous introduisons la taxonomie de Flynn sur les architectures parallèles. Ensuite, les parallélismes d'instructions, de données et de processus sont décris. Enfin, après des notions sur les architectures configurables, les caractéristiques principales de deux architectures spécifiques, les DSP et les processeurs de réseaux (NP : *Network Processor*) sont étudiées.

2.2 Architectures scalaires

2.2.1 CISC

CISC est l'acronyme de *Complex Instruction Set Computer* qui définit les architectures de processeurs faciles à programmer et permettant d'utiliser efficacement la mémoire. Puisque les premières machines programmées en langage machine et les mémoires associées étaient lentes et chères, l'architecture CISC a été généralement implantée un grand nombre de processeurs. En effet, l'utilisation d'instructions complexes diminue la taille du code. La conception de la plupart des processeurs généralistes, comme la série Intel 80x86 et la série Motorola 68K, a longtemps suivi la philosophie CISC [Tokhi95]. Mais les évolutions récentes des technologies logicielles et matérielles ont forcé à réexaminer le paradigme CISC et beaucoup de processeurs CISC modernes sont hybrides et mettent en application beaucoup de principes RISC (*Reduced Set Instruction Computer*).

Les contraintes de conception qui ont mené au développement des CISC (mémoires lentes et en petites quantités ayant comme conséquence que la plupart des premières machines étaient programmées en langage assembleur) donnent les caractéristiques communes des jeux d'instructions de ces architectures. On peut notamment citer les caractéristiques suivantes :

- Format avec deux opérandes où les instructions ont une source et une destination, (registre à mémoire, mémoire à registre et registre à registre) avec plusieurs modes d'adressages.
- Instructions de longueur variable où la longueur change souvent selon le mode d'adressage.
- Instructions qui exigent plusieurs cycles pour s'exécuter.
- Petit nombre de registres généraux. C'est la conséquence directe de l'existence d'instructions qui peuvent fonctionner directement avec la mémoire. La majorité de la surface est donc dédiée au décodage des instructions, à l'exécution, et au stockage du microcode.
- Un registre drapeau, ou registre d'état, qui reflète les effets secondaires de la plupart des instructions notamment lorsque certaines conditions d'erreur se produisent.

Des avantages existent pour ce type d'architecture, surtout à l'époque du développement de ce type d'architecture. Parmi eux, on peut citer :

- L'emploi de la micro-programmation (aussi simple que le langage assembleur), et beaucoup moins cher que celui d'une unité de commande câblée.
- La facilité de micro-programmation de nouvelles instructions permet de concevoir des machines CISC hautement compatibles.
- Lorsque plusieurs opérations sont intégrées dans une instruction de processeur, celui-ci exige moins d'instructions pour exécuter une tâche. Par conséquent, il y a une utilisation plus efficace d'une mémoire relativement lente au niveau des temps d'accès.
- Puisque le jeu d'instructions micro-programmé peut être construit selon les structures de langages de haut niveau, le compilateur n'a pas besoin d'être aussi compliqué que pour un RISC.

Il y a aussi des inconvénients :

- Chaque génération d'une famille de processeurs apporte généralement un nouveau sous-ensemble d'instructions ; par conséquent, le jeu d'instructions et le matériel deviennent plus complexes à chaque version.
- Beaucoup d'instructions spécialisées ne sont pas utilisées assez fréquemment pour justifier leur existence. Seulement approximativement 20% des instructions disponibles sont employées dans un programme typique.
- L'unité de commande est toujours implantée sous forme microprogrammée, ce qui est beaucoup plus lent qu'un système câblé. De plus, un CISC possède habituellement des formats d'instructions des différentes longueurs, ce qui rend la tâche de décodage encore plus difficile.

2.2.2 RISC

RISC est l'acronyme de *Reduced Instruction Set Computer* et décrit un type de processeur contenant un nombre relativement limité d'instructions. L'idée du RISC est de n'utiliser que instructions fréquemment employées (dans les CISC) comme jeu d'instructions pour obtenir la même performance qu'avec un jeu d'instructions beaucoup plus complexe.

Les caractéristiques principales de l'architecture RISC sont :

- Jeu d'instructions réduit
- Instructions moins complexes que pour CISC
- Unité de commande câblée
- Peu de modes d'adressage pour les opérandes en mémoire avec seulement deux instructions de base, *LOAD* et *STORE*
- Beaucoup de registres symétriques organisés en bancs de registres

Un des avantages de l'architecture RISC est la simplicité de la structure de processeur démontrée par le nombre réduit de transistors intégrés. Cependant, en rendant le matériel plus simple, les architectures de RISC ont reporté le problème sur le logiciel.

Une caractéristique très importante est l'unité de commande câblée (les instructions sont câblées). Ceci signifie que dans un processeur RISC, l'unité d'exécution n'est plus commandée à l'aide des microcodes. Au lieu de cela, l'opération entière est implantée sous forme de logique câblée. Ceci accélère considérablement l'exécution des instructions.

Les RISC possèdent un grand nombre de registres généraux, ce qui réduit la fréquence des accès à la mémoire et, par conséquent, augmente la performance globale.

La structure simple des RISC permet une utilisation performante de la technique pipeline. On trouve généralement cinq étages de pipeline pour exécuter les instructions (lecture, décodage, exécution, accès mémoire et écriture). Dans certains processeurs, quelques étages sont combinés en un seul étage. Dans d'autre, les étages de pipeline d'instructions peuvent être encore subdivisés, pour avoir des étages très fins et rapides. Une telle stratégie appelée *super-pipelined* génère beaucoup d'étages de pipeline (dix ou plus). Cette technique est employée dans le MIPS R4000 [Patterson96].

Les CISC et les RISC récents utilisent de plus en plus des méthodes architecturales communes. Aujourd'hui plusieurs processeur RISC contiennent autant d'instructions que les CISC et les processeurs CISC emploient beaucoup de techniques autrefois liés aux RISC.

2.3 Architectures parallèles

Dans cette section, nous expliquons les principes de base des architectures parallèles. Flynn a proposé un modèle simple pour classer les architectures parallèles [Hwang93]. Selon sa classification il existe quatre groupes :

- SISD (*Single Instruction Single Data*) : un seul flot d'instruction et un seul flot de données (cas monoprocesseur).
- SIMD (*Single Instruction Multiple Data*) : plusieurs processeurs exécutent la même instruction en utilisant plusieurs flots de données. Dans ce cas, il existe un seul processeur de contrôle qui lit et lance les instructions.
- MISD (*Multiple Instruction Single Data*) : plusieurs flots d'instructions et un seul flot de données
- MIMD (*Multiple Instruction Multiple Data*) : plusieurs flots de données et plusieurs flots d'instructions, chaque processeur exécutant ses propres instructions sur son propre flot de données (système multiprocesseurs).

Les modèles MIMD et SIMD ont fait l'objet de plus d'attention, surtout dans le domaine des processeurs généraux [Patterson96].

Les machines MIMD sont regroupées, selon l'organisation de la mémoire, en deux catégories : MIMD, à mémoire partagée centralisée, et MIMD, à mémoire distribuée. Dans le premier cas, une mémoire centrale est partagée entre les processeurs, le temps d'accès pouvant être uniforme pour tous les processeurs (UMA : *Uniform Memory Access*) ou non-uniforme (NUMA : *Non-Uniform Memory Access*)(cf. figure 2.1). Pour avoir un grand nombre de processeurs, la mémoire doit être distribuée plutôt que centralisée [Patterson96].

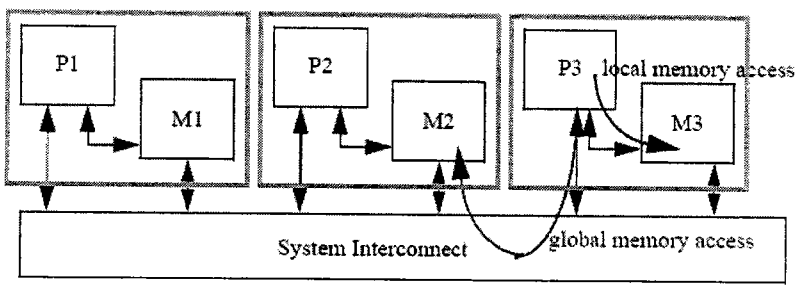


FIG. 2.1: Architecture MIMD-NUMA

Les MIMD à mémoire distribuée possèdent deux avantages principaux. Premièrement, c'est une manière économique d'étendre le débit mémoire. Deuxièmement, il réduit la latence pour les accès à la mémoire locale. L'inconvenient de cette méthode est la complexité de la communication des données entre les processeurs.

Selon la méthode de communication des données entre les processeurs, on dsitingue deux groupes : les machines à mémoire partagée distribuée et les machines à mémoire totalement distribuée. Dans le premier cas, qui utilise implicitement la communication de données via les instructions LOAD et STORE, les mémoires séparées physiquement peuvent être adressées dans un espace d'adressage partagé logiquement. C'est-à-dire qu'une référence à toute case mémoire peut être faite par tout processeur, en supposant qu'il possède les bons droits d'accès. Dans le deuxième cas, qui utilise la méthode de passage de messages pour communiquer les données, l'espace d'adressage est constitué de plusieurs espaces d'adressages privés [Hwang93](cf. figure 2.2).

Un système redimensionnable (*scalable*) de multiprocesseurs, utilisant une hiérarchie de mémoire partagée, a des problèmes de cohérence : comment et quand les caches locaux sont mis à jour. Si les données partagées sont-ils stockées dans de multiples caches des techniques de cohérence de caches doivent être employées pour éviter les conflits.

Deux défis importants rendent le traitement parallèle difficile. Le premier vient de la nature des applications : il s'agit du parallélisme limité disponible initialement dans les programmes. Ce problème peut être partiellement résolu par l'emploi d'algorithmes qui extraient le parallélisme intrinsèque disponible dans les applications. Le deuxième provient du coût élevé des communications entre les processeurs parallèles. Ce problème doit être combattu par des mécanismes de réduction de la fréquence des accès lointains, comme l'emploi de caches pour les données partagées [Patterson96].

On peut considérer le parallélisme global d'une application comme la somme de deux composantes :

- le parallélisme de tâches qui englobe le parallélisme au niveau instruction (ILP ou *Instruction- Level Parallelism*) et au niveau processus (*multi-threading*).

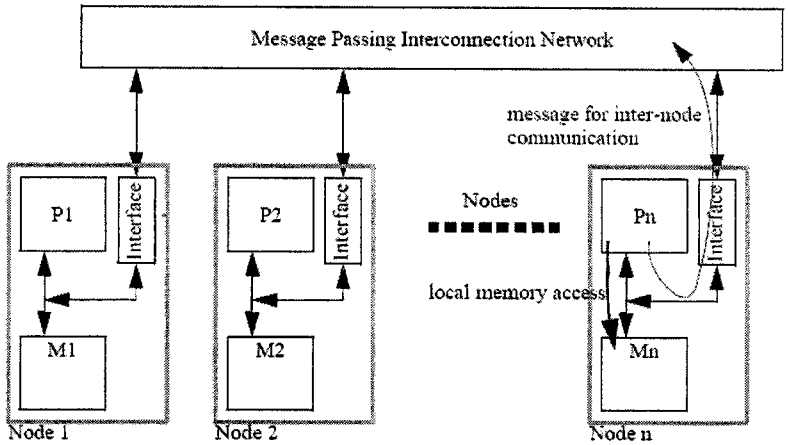


FIG. 2.2: Architecture MIMD avec communication par passage de messages

- le parallélisme de données qui traduit la tendance du programme à appliquer les mêmes traitements à plusieurs données différentes.

Dans les sections suivantes, on explique ces différents types de parallélisme.

2.3.1 Parallélisme de processus

Pour exécuter les applications sur un système multi-tâches, il est nécessaire de réécrire les applications sous forme d'un ensemble de tâches indépendantes pouvant s'exécuter simultanément. Malheureusement, l'extraction du parallélisme de tâches des applications décrites de manière séquentielle est un processus très complexe, et il n'existe aucune méthode automatique donnant des résultats satisfaisants. L'extraction manuelle est actuellement la seule solution efficace, sous réserve que la nature de l'application elle-même puisse offrir de tels niveaux de parallélisme, ce qui n'est pas forcément évident pour toutes les applications, surtout dans le domaine des télécommunications.

En fait, le meilleur moyen pour utiliser efficacement du *multi-thread* est que, dès le départ, les applications soient conçues et décrites sous forme parallèle. Les notions d'orienté-objet et de multi-tâches commencent à apparaître dans les applications complexes. Ainsi, la norme de compression multi-média MPEG-4 modélise les flux d'images comme un ensemble d'objets indépendants possédant leurs caractéristiques et leurs déplacements propres. Par exemple, deux personnages dans une scène seront modélisés par deux objets différents. La composition d'une séquence d'images animées revient alors à calculer les caractéristiques et les déplacements des divers objets constituant la scène et à les superposer pour former l'image finale. Dans ce cas, chaque traitement d'un objet particulier peut être géré par un processus indépendant, et l'animation de l'ensemble devient ainsi une application multi-tâches [Silc00].

Pour gérer le multi-tâches, les processeurs les plus récents (par exemple : Pentium IV-HT (*Hyper Threading*)) se contentent d'inclure des dispositifs matériels permettant une meilleure gestion de la pile logicielle et la sauvegarde automatique de certains registres (afin de diminuer le temps nécessaire aux changements de contexte). Pour l'implantation d'applications multi-tâches complexes, une solution plus répandue consiste à utiliser plu-

sieurs processeurs indépendants et spécialisés sur une structure de type SoC (*System On Chip*). Dans ce cas, toute la difficulté consiste à répartir les tâches sur les différents processeurs et à définir les architectures respectives de ceux-ci (exemple : Intel Dual Core (Pentium D et Pentium EE)).

2.3.2 Parallélisme d'instruction

Ce type de parallélisme est mis en œuvre dans les architectures VLIW (*Very Long Instruction Word*) et superscalaires. Alors que les processeurs RISC exécutent les instructions une à une de manière séquentielle, les processeurs VLIW (ou superscalaires) exécutent ces mêmes instructions élémentaires en parallèle, sous réserve qu'il n'existe pas de contraintes entre les différentes instructions. Le principe consiste à augmenter le nombre des unités de taritement afin de pouvoir exécuter une instruction sur chacune d'entre elles, augmentant ainsi l'IPC (nombre d'instructions par cycle) du processeur. Idéalement, un processeur possédant 4 unités d'exécution devrait pouvoir consommer quatre instructions par cycle, soit un IPC de 4. Dans la réalité, deux facteurs principaux limitent la performance de ces architectures : le parallélisme intrinsèque de l'application, et les ressources matérielles du processeur [Patterson96].

Le degré de parallélisme maximal exploitable est obtenu par une architecture parallèle idéale (avec les ressources matérielles infinies). Pour cela, il existe toujours des limites pour atteindre le parallélisme existant. À l'origine, les premières architectures exploitant l'ILP (*Instruction Level Parallelism*) étaient destinées aux microprocesseurs grand public exécutant un large spectre d'applications de nature parfois radicalement différentes du point de vue de l'ILP. Les nombreuses études sur l'ILP montrent que les différentes applications présentent différents degrés de parallélisme intrinsèque [Bagnordi97][Jouppi89].

Le parallélisme intrinsèque est limité par deux types de contraintes issues de la structure du CDFG (*Control Data Flow Graph*) de l'application. Une instruction produisant un résultat doit obligatoirement s'exécuter avant l'instruction qui consommera ce résultat. On parle de dépendance de données entre les deux instructions : c'est le type de contrainte la plus forte et qui ne peut être supprimée qu'en modifiant profondément la structure de l'algorithme. Les dépendances de contrôle interviennent entre des instructions n'appartenant pas au même bloc de base. Elles traduisent une rupture dans le déroulement séquentiel du programme, due à la présence dans le programme source d'une structure de contrôle évoluée (if-then-else, boucles for et while, appels de fonction, etc.). A priori, ce type de dépendance limite le parallélisme exploitable aux instructions appartenant au même bloc de base, puisqu'on ne sait pas à priori comment se fera l'enchaînement des instructions entre les blocs de base [Patterson96].

Les problèmes de ressources matérielles surviennent lorsque plusieurs instructions devant s'exécuter simultanément veulent accéder à la même ressource (unité d'exécution, registre, emplacement mémoire). Ces contraintes sont évidemment diminuées en redimensionnant l'architecture : duplication des unités fonctionnelles, augmentation du nombre de registres et de la bande passante mémoire. Le matériel requis est cependant très dépendant de l'application : par exemple, une application tirera grand parti de deux ALU et de 16 registres, tandis qu'une autre se contentera de 8 registres mais nécessitera deux multiplieurs pour une accélération maximale. Il n'existe pas de configuration universelle, c'est pourquoi les processeurs à architecture parallèle générale ne peuvent rivaliser avec les processeurs dédiés à une classe d'application.

Dans la section suivante, la discussion concernera les deux types généraux d'implantations de l'ILP : VLIW et superscalaire.

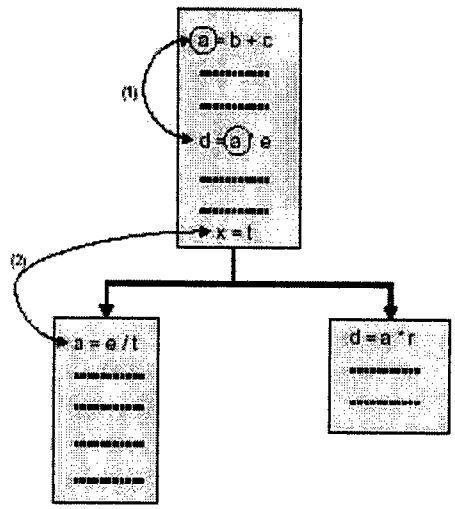


FIG. 2.3: Dépendances de données

2.3.3 Architectures superscalaires et VLIW

On exploite le parallélisme d'instructions dans deux architectures générales : superscalaire et VLIW. Pour encoder des instructions dans un processeur VLIW, des instructions élémentaires (typiquement des instructions RISC), correspondant chacune à des unités fonctionnelles, sont collectées pour former une seule instruction large. Cette opération est réalisée par le compilateur.

Dans les processeurs superscalaires, le parallélisme est analysé dynamiquement par le processeur lui-même. Le programme est stocké en mémoire sous forme d'une séquence d'instructions élémentaires de type RISC. Dans cette représentation, il n'existe encore aucune expression réelle de l'ILP. Lors de l'exécution, les instructions sont chargées par le processeur dans un tampon d'instructions. Des mécanismes matériels complexes effectuent une analyse des contraintes entre les différentes instructions pour déterminer lesquelles peuvent être exécutées en parallèle. Il s'agit d'évaluer les dépendances de données (cf. figure 2.3) et de contrôle entre les différentes instructions et l'occupation des resources du processeur. Dans cette phase, les instructions peuvent être réordonnées pour minimiser les contraintes. Enfin, les instructions terminent de manière séquentielle dans un tampon appelé tampon de réordonnancement.

L'inconvénient majeur de l'architecture superscalaire est la complexité du circuit de contrôle qui augmente la surface et la consommation. Dans le monde des DSP actuel par exemple, la plupart des constructeurs préfèrent l'utilisation de VLIW, sans doute à cause des problèmes liés au principe des superscalaires : l'exécution des instructions avec ordonnancement dynamique est problématique pour l'implantation d'applications soumises à de fortes contraintes temps réel. Cependant, les outils de génération de code pour les superscalaires sont moins complexes que pour les VLIW. D'autre part, l'architecture superscalaire permet la compatibilité binaire du code entre deux générations de processeurs, ce qui constitue son principal avantage [Abnous95].

Les deux solutions souffrent des mêmes inconvénients : un coût matériel et une consommation électrique supérieurs à cause des unités fonctionnelles plus nombreuses

et des bancs de registres multi-ports.

2.3.4 Parallélisme de données

Le parallélisme de données est le parallélisme existant entre des données d'un programme peu ou pas entre elles. Il peut à priori exister entre n'importe quelles données, mais on le trouve plus régulièrement entre des données « éloignées » dans le domaine spatial (applications 2D comme le traitement d'images) ou temporel (application 1D ou « flux » comme les traitements audio). L'indépendance de ces données permet de leur appliquer des traitements en parallèle. Le terme SIMD traduit la capacité d'un processeur à exploiter le parallélisme de données.

Une approche possible est d'augmenter la largeur du chemin de données du processeur pour augmenter la performance. Si, par exemple, la largeur du chemin de données augmente de 32 à 64 bits, la fréquence de fonctionnement peut être réduite d'un facteur deux s'il est possible d'employer le même nombre d'instructions. Cependant, ceci augmente la surface et la complexité du chemin de données. Si les instructions et les opérandes ne sont pas appropriées pour le taritement de 64 bits, cette technique ne permet pas d'augmenter la performance. Dans le traitement de protocoles, il y a beaucoup d'opérations qui ne sont pas appropriées aux chemins de données de 64 bits, étant plus courantes, les instructions basées sur les bits et les octets.

2.4 Architectures dédiées et configurables

Contrairement aux processeurs généraux dont le but est de pouvoir exécuter un spectre très large d'applications, les processeurs spécialisés sont conçus pour un domaine d'application particulier, voire parfois même pour une seule application. L'intérêt est d'obtenir des processeurs possédant les performances minimales requises par l'application en terme de puissance de calcul (et éventuellement de consommation) et dont le coût est inférieur à celui des processeurs du commerce.

Le niveau de la spécialisation est un problème. En effet, on ne peut trouver un modèle de processeur universel capable de traiter efficacement n'importe quel type d'applications, tant leurs contraintes respectives peuvent être différentes et réclamer des solutions parfois totalement divergentes. Même les microprocesseurs généraux sont en fait conçus pour un domaine d'application particulier, celui des applications informatiques générales.

Une architecture configurable fournit une manière d'étendre le jeu d'instructions, et les ressources matérielles et la mémoire, ainsi qu'un jeu d'outils pour implanter une architecture spéciale. À l'heure actuelle, la tendance est à une plus grande généralisation des architectures, qui passent du niveau de spécialisation « Application » à celui de « Domaine d'Application » pour permettre une plus grande flexibilité.

Dans les sections suivantes, nous discutons des architectures configurables et spécialisées.

2.4.1 Architectures configurables

Dans les architectures configurables, le concepteur peut conduire le matériel dans la direction nécessaire pour mieux supporter l'application. Les outils appropriés de compilation et d'analyse permettent au concepteur d'optimiser rapidement l'architecture pour l'application spécifique, en éliminant les éléments matériels inutiles, de manière à réduire la taille et le coût. Par exemple, *ArchC* est un langage de description matérielle qui permet

aux utilisateurs d'évaluer les modèle architecturaux des processeurs [ARCHC].

Puisque le système de conception est guidé par l'application, le support de nouvelles normes et les mises à jour sont rapides. En utilisant les outils appropriés, ceci peut être accompli sans ajouter de circuit spécifique, et la performance de l'architecture peut être améliorée par un simple changement de configuration.

Il existe deux étapes principales dans le flot de conception des architectures configurables : l'exploration et la génération. L'intérêt dans la phase d'exploration est de déterminer la valeur optimale des paramètres de configuration de l'architecture. Elle est basée sur un ensemble d'outils logiciels (assembleur, simulateur d'instructions et éventuellement compilateur) dont le comportement tient compte des paramètres et fournit au final une estimation précise de la performance des applications sur la configuration proposé. Si les chiffres obtenus satisfont les contraintes du cahier des charges, on peut alors passer à la réalisation matérielle de l'architecture correspondant aux valeurs courantes des paramètres. Dans le cas contraire, le modèle courant doit être modifié par le changement de la valeur d'un ou plusieurs paramètres.

La phase de génération permet ensuite d'obtenir l'image matérielle de l'architecture qui sera utilisée pour la réalisation physique du circuit. Cette description matérielle est produite à partir du modèle paramétré. Par exemple, pour les processeurs, le niveau de description obtenu est variable selon les cœurs et dépend du type de marché visé, celui des cœurs logiciels (*soft*) ou des cœurs matériels (*hard*). Les cœurs logiciels sont fournis sous forme de *netlist* RTL synthétisable, en général au format VHDL ou Verilog. Leur grand intérêt provient de leur indépendance vis-à-vis de la technologie, contrairement aux cœurs matériels qui sont fournis sous forme de macro-cellules optimisées pour une technologie donnée.

Pour avoir un temps de développement le plus court possible il faut utiliser le flot de conception le plus direct et le plus automatisé possible. Le flot de conception basé sur la variation des paramètres de l'architecture impose aux outils d'être génériques vis-à-vis de ces paramètres pour pouvoir tester rapidement l'influence de leurs variations sur les performances.

Les processeurs configurables les plus évolués permettent de faire varier le nombre et la nature des unités fonctionnelles (UF) intégrées au chemin de données. La plupart d'entre eux imposent une structure de base fixe composée d'UF prédéfinites à laquelle peuvent s'ajouter une ou plusieurs UF utilisateur. L'ajout d'unités fonctionnelles utilisateurs accroît la complexité du processeur de plusieurs manières. Les outils de génération de code doivent gérer les nouvelles instructions de commande des unités. La topologie du chemin de données étant modifiée, c'est une grande partie de la description matérielle qui doit être régénérée. La phase de génération matérielle devient alors particulièrement critique. Le processeur « *Metacore* » a poussé la flexibilité jusqu'à rendre configurable la connectivité du chemin de données [Yang00]. La famille *Xtensa* de *Tensilica* propose des outils de développement pour générer automatiquement une architecture ainsi que les outils optimisés pour cette architecture [XTENSA]. Dans le domaine du traitement de protocoles, une méthodologie de conception d'une architecture configurable a été proposée pour traiter des protocoles de réseau sans fil dans [Tuan01].

2.4.2 DSP

A Caractéristiques des DSP

L'architecture des processeurs DSP est directement issue de la structure des filtres numériques comme le filtre FIR (*Finite Impulse Response*).

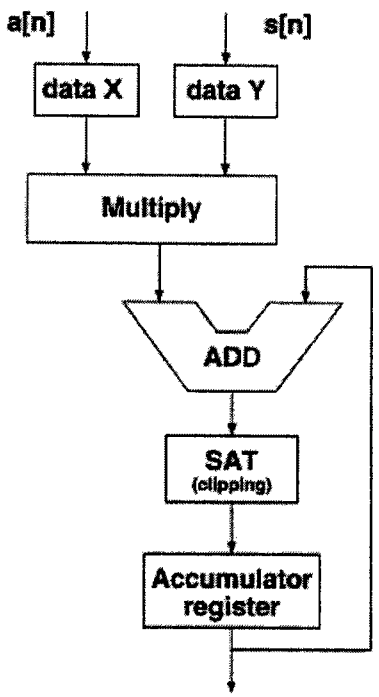


FIG. 2.4: Flot de données dans une instruction MAC

Comme le nom de DSP (*Digital Signal Processing*) l'indique, leur conception les dédie au traitement numériques des signaux. Dans le domaine du traitement du signal numérique, le calcul suivant est fondamental :

$$g[n] = \sum_{k=0}^{N-1} a[k]s[f(n,k)] \tag{2.1}$$

Dans cette équation, l'échantillon de sortie est obtenu par N multiplications et N-1 additions. À un instant donné, les N-1 derniers échantillons reçus résident dans la ligne à retard et N est la longueur du filtre. Cela signifie qu'il faut faire à chaque cycle d'échantillonnage, N multiplications et N-1 additions. Cette opération se fait dans une structure de base appelée MAC (*multiply-accumulate*) (cf. figure 2.4).

B Jeu d'instructions

Aujourd'hui, la plupart des microprocesseurs classiques utilisent un jeu d'instructions de type RISC dont les fonctionnalités sont générales, une instruction correspondant à une seule opération. À l'inverse, les DSP possèdent un jeu d'instructions spécialisé avec

des opérations arithmétiques complexes. De plus, certains instructions encodent plusieurs opérations en parallèle comme l'instruction MAC du filtre FIR.

L'architecture conventionnelle des DSP utilise un faible nombre d'unités d'exécution et un jeu de registres hétérogène. Le nombre de bits nécessaires pour encoder une instruction est habituellement inférieur à celui requis pour une architecture plus générale et homogène.

C Les registres

Dans les microprocesseurs RISC classiques la mémorisation interne des données se fait au moyen d'un banc de registres central à l'architecture, souvent multi-ports, chaque registre ayant la même visibilité vis-à-vis des unités fonctionnelles. Du point de vue de la programmation cela signifie que les registres sont complètement interchangeables et forment un jeu de registres homogènes.

La stratégie utilisée dans les DSP est différente. Certains registres sont dédiés à des opérateurs particuliers et ne peuvent être accédés que comme opérandes source et résultat des instructions associées à l'opérateur. Historiquement, la raison d'être de ce type d'architecture avec des registres hétérogènes se justifie par le coût matériel minimum.

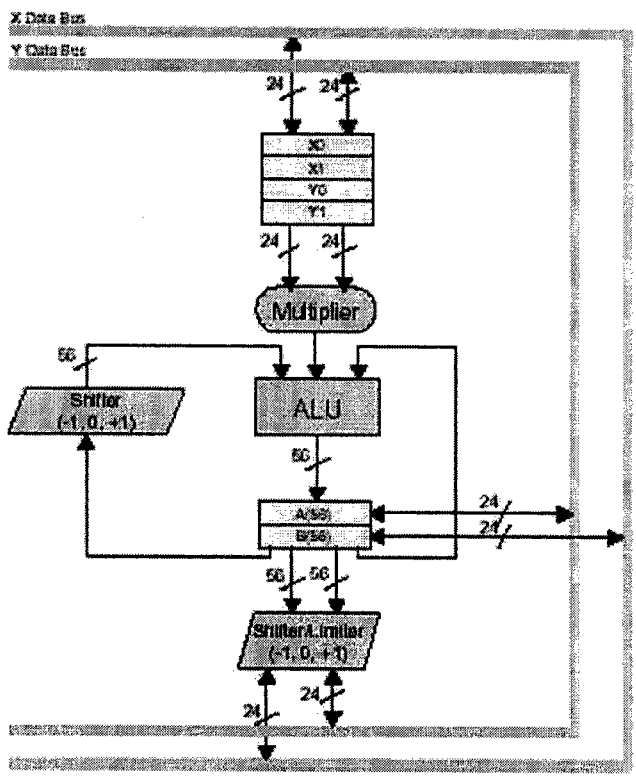


FIG. 2.5: Chemin de données du DSP 5600x de MOTOROLA

La figure 2.5 montre une architecture à jeu de registres homogènes équivalent à celle du DSP 5600X. Elle utilise un banc de registres multi-ports (3 ports d'entrées et 5 ports de sorties) permettant de fournir suffisamment de données pour réaliser en parallèle une multiplication, une addition et un décalage [Motorola98]. À nombre de registres égal, la

solution banc de registres homogène est beaucoup plus coûteuse en performance et en consommation.

Les concepteurs ont donc choisi d'utiliser des architectures hétérogènes minimisant le matériel et favorisant la performance. L'inconvénient de ce type d'architecture est son manque de souplesse au niveau de la programmation. En d'autres termes, la circulation des données dans ce type d'architecture est plus complexe que dans les architectures homogènes parce que, par exemple, tel registre n'est connecté qu'à telle unité, tel résultat doit aller obligatoirement dans tel registre et pas un autre,etc. C'est pourquoi la programmation de ce type d'architecture est difficile et demande beaucoup d'attention et d'expérience.

D Unités fonctionnelles

Les DSP utilisent des unités fonctionnelles spécialisées pour effectuer les calculs mathématiques plus rapidement et plus précisément que les processeurs classiques. On peut trouver trois grandes catégories d'unités fonctionnelles dans les DSP :

- **Unité arithmétique et logique (UAL)**

  Cette unité effectue des fonctions classiques comme : l'addition, la soustraction et des opérations logiques élémentaires (ET, OU, OU-exclusif, NON). Les opérations plus spécifiques telles que la gestion de la précision se font dans cette unité, comme par exemple les fonctions de saturation ou d'arrondi. Il existe d'autres fonctions dans cette unité comme la valeur absolue, le maximum/minimum de deux valeurs, etc.(cf. figure 2.5)

- **Multiplieur-accumulateur (MAC)**

  Dans cette unité, une multiplication et une addition sont réalisées en une seule instruction. La plupart des DSP exécutent ces opérations en un cycle machine. Grâce à l'UAL et à l'unité MAC, les DSP sont capables d'effectuer une opération MAC à chaque cycle. Il existe deux types de MAC : purement combinatoire (cas du 5600x), et pipeline, au moyen d'un registre « produit » entre le multiplieur et l'accumulateur (registre p0 et p1 des DSP Lucent16xxx (cf. figure 2.6))

- **Décaleurs (*shifters*) et unité de manipulation de bits (BMU)**

  Les opérations de décalage et de test au niveau bit sont réalisées dans cette unité. Les opérations de décalage sont utilisées généralement pour normaliser les valeurs dans les calculs à virgule flottante. Les opérations de manipulation de bits sont utiles dans les applications incluant du contrôle, ainsi que dans les algorithmes de traitement de données sous forme de flux de bits de longueur variable.
  Cette classification n'est pas universelle et la complexité, le type et le nombre d'unités fonctionnelles varient selon les processeurs. Certains DSP disposent en plus d'unités fonctionnelles spécialisées, pour un domaine d'application particulier, qui leur permettent d'accélérer l'exécution de certains fonctions critiques. La plupart des DSP destinés à la téléphonie mobile, par exemple, disposent d'un dispositif matériel pour le décodage de Viterbi utilisé dans la norme GSM.

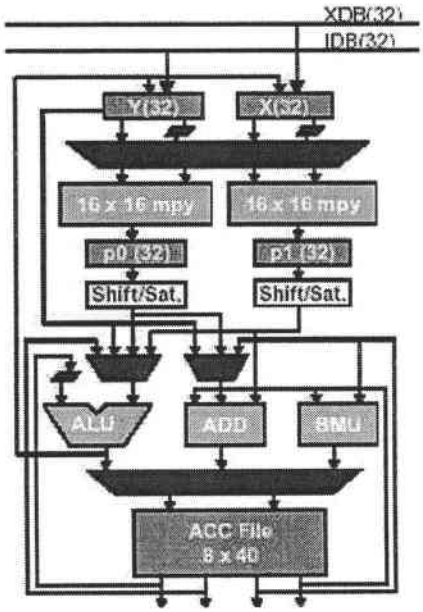


FIG. 2.6: Chemin de données du DSP 16xxx [Lucent]

E Architecture mémoire

Traditionnellement, les architectures de microprocesseur sont du de type Von Neumann qui utilise un bus unique pour l'accès aux données et aux instructions, mais cette approche est trop limitée dans le cas des applications de traitement du signal. En effet, la plupart des algorithmes DSP requièrent une bande passante mémoire supérieure à celle que peut fournir l'architecture Von Neumann. Par exemple, dans les opérations de filtrage RIF, le processeur doit pouvoir effectuer une multiplication et une addition et accéder à la mémoire plusieurs fois dans la même instruction, alors que l'architecture Von Neumann permet juste un accès par cycle. C'est pourquoi les processeurs DSP utilisent généralement une architecture mémoire différente appelé Harvard [Bajot01].

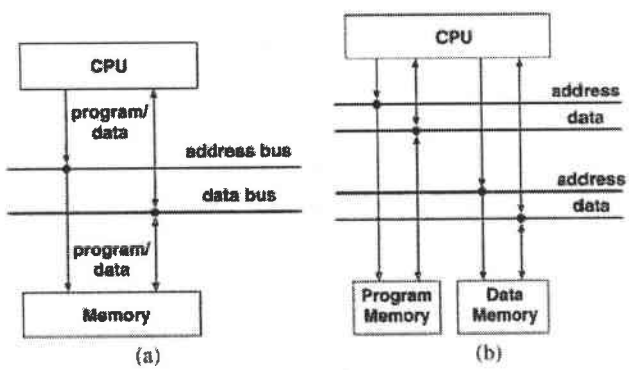


FIG. 2.7: Architectures Von Neumann (a) et Harvard (b)

Dans cette architecture, deux voire plusieurs bus d'adresse et de données sont utilisés

et permettent d'effectuer plusieurs accès simultanés à la mémoire par cycle. Dans l'architecture Harvard classique il existe un bus programme et un bus données mais dans les architectures plus avancées, dites enhanced Harvard (Harvard amélioré), il existe un plus grand nombre de bus afin d'offrir des bandes passantes supérieures, comme par exemple, 2, 4 ou 8 données par cycle.

Dans le DSP5600X de MOTOROLA par exemple, on dénombre trois espaces mémoires distincts et leurs bus associés : un pour les instructions et deux pour les données (bus X et bus Y). Ce processeur peut donc à chaque cycle charger une instruction et deux données.

Contrairement aux microprocesseurs, la plupart des DSP n'utilisent pas de mécanisme de mémoire cache, ni pour les données, ni pour les instructions. En effet, la taille des algorithmes de traitement du signal est généralement faible (quelques dizaines de lignes de C) et le code résultat peut tenir dans des espaces mémoire limitées. De ce fait, la philosophie dominante pour les DSP est de placer la mémoire instruction directement sur le circuit, en général sous la forme de mémoire rapide de type SRAM. Un espace est aussi réservé sur le circuit pour des blocs de mémoires données à accès en un cycle, donc à débit maximal. L'accès aux données se fait souvent via des mémoires double accès, plus souples d'utilisation mais plus coûteuses. Il existe cependant des DSP qui utilisent de la mémoire cache mais en quantité très limitée [Bajot01].

DSP			horloge	caches			interne	
famille	proc.	type	MHz	LIP Ko	LID Ko	L2 Ko	RAM Ko	ROM Ko
TMS320C6xx VLIW 8 × 32b inst.	C62x	fix.	150 – 300	0 ou 4	0 ou 4	0 ou 64	64 – 512	0
	C64x	fix.	300 – 720	16	16	256 – 1024	0	0
	C67x	flo.	100 – 200	16	16	256 – 1024	0	0
TMS320C5xx IPC = 2	C54x	fix.	40 – 160	0	0	0	16 – 512	0 – 256
	C55x	fix.	144 – 200	0, 16, 24	0	0	32 – 320	32 – 64
TMS320C2xx CPI = 1 à 2	C24x	fix.	20 – 40	0	0	0	1 – 5	0 – 32
	C28x	fix.	150	0	0	0	36	128 – 256
DSP56xx	563x	fix.	80 – 240	0 ou 3	0	0	9 – 100	0
	568x	fix.	80 – 120	0	0	0	4 – 128	0 – 2
ADSP21xx SIMD/SISD	211x	flo.	80 – 100	< 1	0	0	128 – 512	0
	210x	flo.	40 – 60	< 1	0	0	64 – 512	0
TigerSHARC	TS101	2	300	0	0	0	768	0

FIG. 2.8: Caractéristiques de quelques DSP actuels

Les processeurs DSP intègrent toujours des interfaces pour des mémoires externes qui permettent d'augmenter la capacité de mémorisation du système.

F Unité de contrôle

L'unité de contrôle des DSP gère le flot des instructions à exécuter, et les modes basiques de contrôle des processeurs tels que l'exécution séquentielle, les sauts et les branchement conditionnels, les appels de sous-programmes, et les interruptions. Dans un processeur classique, une boucle logicielle est codée à l'aide d'une instruction de branchement conditionnel, une décrémentation et un test de compteur de boucle. Dans un DSP, il suffit d'indiquer le début et la fin de la boucle, ainsi que le nombre de répétitions : toutes les opérations seront effectuées par le matériel sans perte de cycle machine. Ce mécanisme

s'appelle *zero overhead loop*. Certains DSP autorisent plusieurs niveaux de boucles imbriqués dont le nombre de niveaux est variable et limité par le nombre de registres internes mémorisant les caractéristiques des différentes boucles [Berkeley00].

Pour sauvegarder les adresses de retour des sous-programmes, certains DSP utilisent une pile matérielle afin de permettre un retour rapide en un cycle. Dans certains DSP, il existe des registres cachés (*shadow register*) pour sauvegarder les adresses de retours.

G Chemin de données

Du point vue du type de données, il existe deux catégories bien distinctes de DSP : les processeurs « entiers » et les processeurs « flottants ». Les premiers sont les plus utilisés, principalement pour une question de coût. De plus, l'arithmétique entière est plus rapide que la flottante. Dans les processeurs « entiers » ou à virgule fixe les nombres sont codés en complément à 2 et peuvent être de deux types : entiers et fractionnaires. Un nombre flottant utilise trois champs distincts : un bit de sign, une mantisse et un exposant. La dynamique du signal (le rapport entre la plus petite et la plus grande valeur representable) est beaucoup plus grande en flottant (1535 dB en 32 bits) qu'en entier (187 dB en 32 bits).

La raison pour laquelle les processeurs flottants sont moins utilisés est qu'ils sont plus coûteux matériellement : les opérations flottantes sont plus complexes, consomment plus, occupent plus de silicium, et sont plus lentes que leurs équivalentes entières.

Il faut noter aussi la largeur du chemin de données. La largeur du chemin de données native d'un processeur est la largeur maximum des données pouvant circuler sur ses bus. La plupart des DSP fonctionnent en 16/32 bits, ces deux chiffres correspondant à des données en simple/double précision. Certains processeurs fonctionnent aussi en 24/48 bits. La largeur des données influe directement sur la précision arithmétique et la dynamique maximale du signal et surtout sur le coût matériel et sur la consommation du circuit. La sélection d'une largeur de données dépend des applications. La téléphonie mobile, par exemple, et les applications de type modem utilisent 16 bits, tandis que les applications de traitement audio demandent une dynamique plus large et utilisent 24 bits [Bajot01].

La plupart des algorithmes DSP utilisent principalement l'opération MAC qui consiste à accumuler le résultat d'un produit avec un résultat précédent. Le produit de deux termes de N bits donnant un résultat sur 2N bits, les registres et les unités fonctionnelles dans l'opération MAC sont surdimentionnées pour ne pas perdre de bits significatifs.

H Unité de calcul d'adresse

La plupart des processeurs DSP sont capables d'effectuer l'opération MAC et de modifier les pointeurs d'adresses en un cycle machine. C'est-à-dire qu'ils doivent faire 3 opérations en un cycle : l'opération MAC et les deux incrémentations de pointeurs. Les calculs d'adresses sont effectués par des unités spécialisées (AGU : *Address Generation Unit*). Ces unités travaillent en parallèle avec les autres unités fonctionnelles.

En plus des modes d'adressage classiques tels que l'adressage immédiat ou l'adressage direct, a plupart des DSP utilisent trois modes spécifiques additionnels :

- l'auto incrémentation et décrémentation de pointeurs
- l'adressage modulo
- l'adressage bit-inverse

Les deux premiers sont utilisés dans l'exécution d'algorithmes, tel que le filtrage FIR. Le troisième est un type d'adressage particulier utilisé dans les algorithmes de transformée de Fourier rapide (FFT) qui manipule les échantillons dans un ordre différent de l'ordre

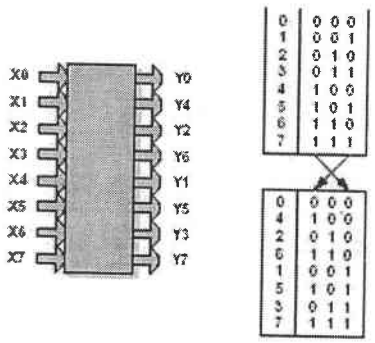


FIG. 2.9: L'adressage bit-inverse dans une FFT 8 points

séquentiel. On trouve cette fonctionnalité dans toutes les AGUs [Berkeley00].

2.4.3 Processeurs de réseau (NP)

Un processeur de réseau intervient typiquement dans les systèmes routeurs, entre les lignes physiques de communication et la matrice de commutateurs (*switch fabric*). Dans ces routeurs, les tâches sont séparées en deux « plans » : le plan « contrôle » et le plan « données ». Le plan « contrôle » gère l'information se rapportant au paquet de données (en tant qu'entité), tel que l'origine, la destination, le type, les statistiques, c'est-à-dire des tâches de routage qui ne sont pas trop critiques en temps. Le plan « données » occupe du traitement des bits internes du paquet, l'exécution de ces tâches devant se faire à la vitesse de transmission. Un composant a le qualificatif de *wire speed* s'il est capable de traiter les données au même débit que celui entrant. Les NP (*Network Processors*) font donc partie de cette catégorie devant travailler sur les paquets à la vitesse de transmission sur les lignes [Comer04].

Les architectures de routeurs ont évolué au cours des années et deux types subsistent actuellement. La plupart des routeurs sont construits autour d'une carte mère de commutation, qui est un *crossbar* pour faire transiter les paquets de données. Les cartes d'interfaces la ligne sont rattachées à cette carte mère. Chaque carte d'interface peut avoir un ou plusieurs ports, par exemple Ethernet, ATM, et ADSL. Le traitement se fait sur les cartes de ligne pour les paquets entrants et, quand le port de sortie est déterminé, (par une consultation de l'adresse IP) le paquet est envoyé via la carte mère à la carte de ligne correspondante, qui stocke le paquet jusqu'à ce que le port de sortie soit disponible. L'autre manière de construire les commutateurs ou les routeurs est celle d'employer une mémoire partagée de vitesse élevée (au lieu d'une carte mère de commutation) et des tampons locaux au niveau des ports de sortie. Ce dernier type d'architecture peut être intégré sur une puce unique.

La fonctionnalité fondamentale d'un commutateur ou d'un routeur est simple, mais l'exigence de haute performance concernant l'exécution rendent la conception complexe. L'autre facteur qui contribue à la complexité est la demande croissante de qualité de service (QoS). Cela signifie que différents types de paquets doivent être traités différemment. Par exemple, les trains de paquets avec des contraintes de temps réel, telles que audio et vidéo, devraient obtenir une priorité plus élevée que des paquets de transfert de fichier et de courrier électronique qui tolèrent des petits retards. En tous cas, les commutateurs et les routeurs sont des systèmes temps réel où les paquets doivent être traités dès qu'ils

arrivent.

Les conditions de QoS augmentent la charge de traitement par paquet de manière significative, parce que le contenu des paquets doit être inspectés afin de déterminer leur type. Ceci est désigné sous le nom de classification de paquet : normalement, des champs de l'en-tête des paquets sont employés pour faire cette classification. Une fois les paquets classés, ils doivent être traités différemment, ce qui est accompli par un système de gestion de file d'attente. Un autre problème pour les routeurs et les commutateurs est la demande croissante de sécurité dans les réseaux informatiques. Ceci conduit à chiffrer beaucoup de paquets [Comer04].

La multiplication de tâches dans les commutateurs et les routeurs est la raison du développement des NP qui sont des processeurs consacrés au traitement de ces tâches. La plupart des systèmes NP sont constitués d'un processeur principal et de plusieurs co-processeurs, par exemple, pour le chiffrement, la classification de paquet et la gestion de la file d'attente. Il existe des interfaces normalisées concernant la manière de relier les co-processeurs au processeur principal [Bhugra04]. Les contraintes sur la conception sont normalement liées au débit en sortie et à la consommation d'énergie.

A L'architecture générale des NP

La plupart des architectures de processeur de réseau sont basées sur une unité de traitement constituée d'un banc de processeurs fonctionnant en *multi-threads*. Une telle architecture est basée sur le fait que le trafic du réseau dans un intervalle de temps court est compose de paquets (indépendants la plupart du temps) qui peuvent être traités en parallèle. Le *multi-threading* permet alors la pleine utilisation de la capacité de traitement disponible pour compenser la latence des différents accès à la mémoire. Plusieurs études [Bux01] confirment la meilleure performance d'un processeur *multi-threads* pour les applications de réseau.

Les architectures de processeur de réseau peuvent différer dans l'arrangement global entre le chemin de contrôle et le chemin de données [Peyravian03]. Dans les cas dits « architecture en-ligne » (figure 2.10.a), toutes les données traversent l'unité de traitement et, par conséquent, la capacité de traitement dépensée sur un paquet est proportionnelle à sa longueur. D'autre part, le traitement du paquet est généralement limité à une partie seulement du paquet (l'en-tête du paquet) et, par conséquent, les exigences de traitement sont en réalité indépendantes de la longueur du paquet. Autrement dit, l'architecture parallèle bénéficie de cette caractéristique en chargeant l'unité de traitement seulement avec la charge utile du paquet contenant une partie complète ou un extrait de l'en-tête du paquet (figure 2.10.b).

Un autre aspect intéressant de l'architecture de processeur de réseau est la manière dont la charge de travail est distribuée entre les différentes unités du traitement. L'exigence de base pour tous les composants du réseau est de préserver l'ordre de transmission des paquets. L'approche la plus simple est de modéliser le processeur comme une FIFO, où chaque unité de travail est assignée à une unité de traitement puis libérée dans un ordre FIFO. Une telle approche mène à une utilisation faible de la capacité de traitement disponible, parce qu'il y a des différences significatives dans la durée de traitement nécessaire pour des unités consécutives. L'approche alternative est de concevoir une logique de réordonnancement qui préserve l'ordre entre les paquets seulement dans des cas particuliers. Une approche possible est décrite dans [Bux01]. L'inconvénient évident d'une telle conception est la complexité plus élevée et le fait que, dans les cas extrêmes, le comportement est identique à l'approche FIFO.

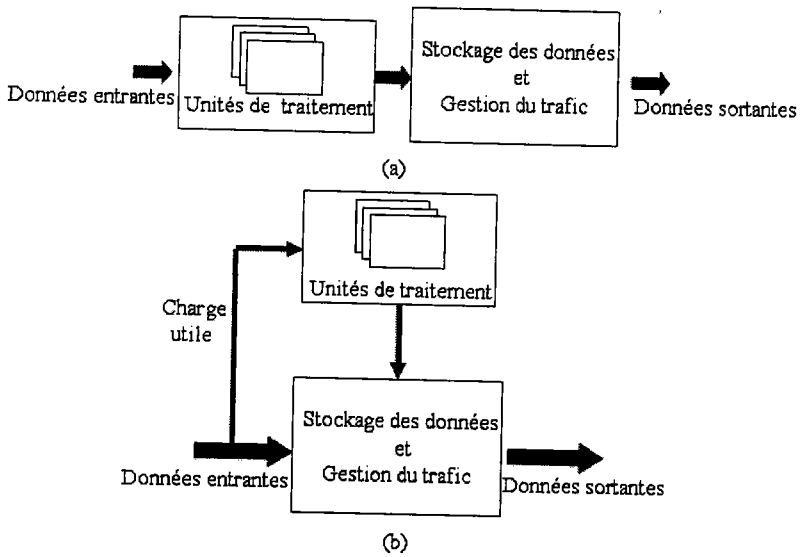


FIG. 2.10: Architecture de processeur réseau (a) en-ligne (b) parallèle

Il est très important de s'assurer que le processeur de réseau peut gérer la situation où le traitement de certains paquets exige une puissance plus élevée que la puissance de traitement disponible. Une ressource très importante pour réaliser ceci est une file d'attente devant l'unité de traitement capable d'absorber les variations de la durée du traitement. Évidemment, la taille de la file d'attente dépend, non seulement des conditions externes (" quelles sont les tailles des salves(*burst*) du trafic ?" "quelle est la variation de la durée du traitement ?", etc.), mais également des aspects internes de l'architecture du processeur de réseau déjà mentionnés (en ligne ou arrangement parallèle), ainsi que de l'algorithme de distribution de la charge de travail [Comer04].

L'organisation de la mémoire est l'autre aspect important pour les processeurs de réseau. En général, une hiérarchisation adéquate de la mémoire avec une hiérarchie adéquate permet d'augmenter la performance d'un processeur de réseau. Cependant, à cause de la différence dans le traitement des paquets entre les caractéristiques de données et les programmes , l'organisation mémoire de ces processeurs exige des méthodes spécifiques [Chiueh99][Jones03].

Le tableau 2.1 présente les caractéristiques de quelques exemples de processeurs de réseau. Des quelques sociétés qui fabriquent des processeurs réseaux, Intel est le leader en commandes passées, avec sa famille IXP. Sa popularité est essentiellement liée à la souplesse de son produit et au support matériel et logiciel par des tierces-parties. La société AMCC (*Applied Micro Circuits Corporation*) vient en second. Elle a évolué plus vite qu'Intel vers les processeurs réseaux hautes performances, mais son succès vient beaucoup de ses composants d'entrée de gamme (PHY, framers, contrôleurs d'accès média...) qui fonctionnent parfaitement avec ses NPU. AMCC a annoncé une cinquième génération de NPU, la famille nP 5. Ces composants, 5 Gbits/s et 10 Gbits/s, full-duplex, intègrent des unités de gestion du trafic et de recherche, et possèdent des interfaces standard et non plus propriétaires. Motorola vient ensuite grâce à son C-5 qui est unique dans sa capa-

Référence	Fabricant	Architecture	Fréquence	Débit	Fonctions câblées
IXP12X0	Intel	6 RISC multithreaded et un cœur stronbArm	166, 200 MHz	1 Gbits/s	transformation CRC , ERC (error correction code)
IXP2400	Intel	8 RISC multithreaded et un cœur XScale	600, 400 MHz	2.4 Gbits/s	1 module hash unit 16 Ko mémoire scatchpad
IXP2800	Intel	16 RISC multithreaded et un cœur XScale	1.4 GHz, 700 MHz	10 Gbits/s	Non
nP7110	AMCC	1 nPcore 64 bits avec un jeu d'instructions optimisée	200 MHz	2 Gbits/s	transformation des paquets statistiques
nP7120	AMCC	2 nPcore 64 bits	200 MHz	5 Gbits/s	transformation classification statistiques
nP7510	AMCC	6 nPcore 64 bits	333 MHz	10 Gbits/s	transformation des paquets statistiques
c-5	Motorola	16 micromoteurs processeurs channel	166, 200 MHz	5 Gbits/s	interface avec switch fabric, look-up table, gestion de buffers
c-3e	Motorola	8 micromoteurs(channel) 8 micromoteur service	180 MHz	3 Gbits/s	idem c5
NP4GS3 (Rainer)	IBM	16 micromoteurs RISC 32 bits 1 cœur powerPC405	133 MHz	5 Gbits/s	recherche filtrage, modification transmission
NP2G	IBM	12 micromoteurs RISC 32 bits 1 cœur powerPC405	133 MHz	2 Gbits/s	recherche filtrage, modification transmission

TAB. 2.1: Les caractéristiques de quelques processeurs de réseau [Trézéguet02]

cité à supporter toutes les connexions : Ethernet, POS (Packet Over Sonet), ATM, HDLC et Fibre Channel. Une stratégie qui réussit aussi à ses successeurs, le C-5e et le C-3. Motorola ne propose pas de produits interfaces de ligne ou unité de commutation, mais ses PowerPC sont largement utilisés dans le plan contrôle. IBM commercialise aussi des PowerPC avec ses NPU PowerNP. Le PowerNP Rainier est très apprécié dans les applications multiservices (*DiffServ*) au débit OC-48 (*Optical Carrier Level, 2.488 Gbits/s*). Les leaders du marché, AMCC et Intel, sont plus optimistes mais ils sont aussi assez solides pour s'offrir une gamme complète de NPs entre 1 et 10 Gbits/s. Dans la section suivante nous décrivons un processeur connu dans ce domaine, le IXP2400 d'Intel.

B Le processeur IXP2400 Intel

Les deux parties principales de l'architecture d'IXP (cf. figure ?? sont un cœur de processeur XScale et quelques processeurs RISC appelés « micromoteurs » ou ME (*Micro-Engines*). L'architecture de ce processeur de réseau est basée sur une division des opérations du traitement de protocoles. Dans n'importe quelle application de traitement de paquet, deux types d'opérations doivent être exécutées, les opérations de chemin rapide et les opérations de chemin lent. Les opérations de chemin rapide sont effectuées typiquement sur chacun des paquets et doivent être accomplies aussi rapidement que possible pour respecter le débit de ligne. Un exemple d'opération de chemin rapide est l'acheminement (*forwarding*) IPv4 qui exige de rechercher l'adresse IP de destination dans un tableau d'expédition pour déterminer l'interface de sortie par laquelle un paquet IP doit expédiée puis, de mettre à jour de l'en-tête du protocole IPv4, et enfin d'expédier le paquet sur l'interface sélectionnée. Une opération de chemin lent est nécessaire seulement dans des cas exceptionnels et prend plus de temps pour s'exécuter qu'une opération de chemin rapide. Un exemple d'opération de chemin lent se produit quand un composant

de réseau reçoit un paquet IPv4 avec certaines options d'en-tête et le traitement de ces options prend beaucoup plus de temps que l'opération habituelle de chemin rapide. Indépendamment des opérations de paquet, chaque composant de réseau ou carte interface de ligne doit exécuter certains protocoles de plan de contrôle tels que les protocoles de signalisation [Intel01].

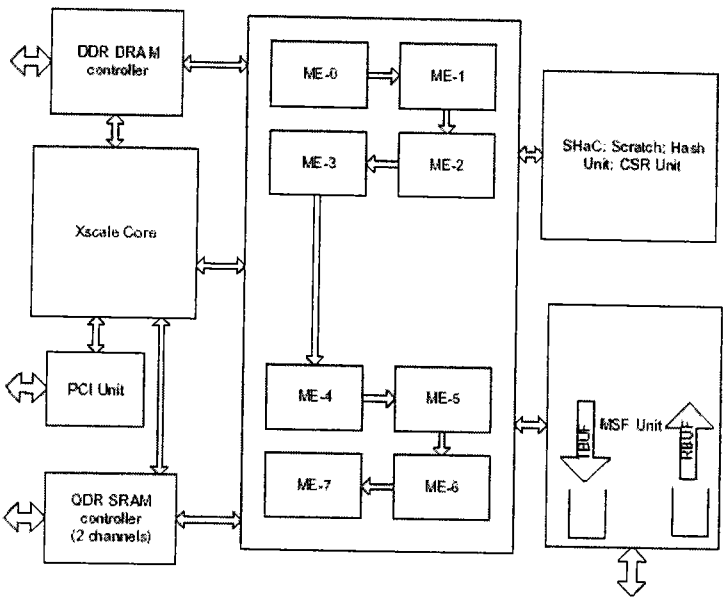


FIG. 2.11: Architecture du IXP2400

Les deux modules principaux de l'architecture IXP sont conçus pour soutenir le traitement des paquets et le traitement du protocole de contrôle. Le cœur Xscale est conçu pour le traitement du chemin lent et du protocole de contrôle, tandis que les ME sont conçus pour le traitement des paquets de chemin rapide. Le cœur XScale d'Intel est un processeur généraliste avec une architecture RISC 32 bits pour initialiser et contrôler le NP. Il est utilisé pour la manipulation des exceptions, le traitement du chemin lent et les tâches de plan contrôle.

Quand un NP doit traiter un train de paquets entrants à la vitesse de ligne, il doit achever le traitement d'un paquet dans un intervalle de temps court. Par exemple, avec le débit de ligne OC-48 (2,4 Gbps), un paquet POS (*Packet Over SONET*) de taille minimum arrive approximativement chaque 100 cycles. Cependant, un traitement simple exige quelques opérations de consultation et d'accès à la mémoire. Avec une estimation de latence d'accès SRAM de 80 à 100 cycles et d'accès SDRAM de 150 à 200 cycles, il est impossible d'achever le traitement d'un paquet avant que le prochaine n'arrive. L'architecture IXP emploie des ME de traitement où chacun possède huit *threads* matériels. Assigner chaque paquet à un *thread* séparé sur un ou plusieurs ME permet le traitement de multiples paquets en parallèle. Chaque ME fournit les opérations *multithreads* contrôlées par le logiciel. Supposons que pendant la durée du cycle d'accès à la mémoire externe un thread d'exécution soit souvent bloqué à attendre l'opération mémoire. La disponibilité de plusieurs *thread* en même temps permet d'intercaler des *threads*, il y a souvent au moins un thread prêt à être exécuté pendant que d'autres sont bloqués. Ceci améliore l'utilisation des ressources des ME.

Les MEs sont des « moteurs » entièrement programmables et d'usage universel qui peuvent être programmés pour le traitement arbitraire de paquets ou pour d'autres fonctions. Cependant, le traitement des paquets requiert certaines fonctions bien définies , telles que le hachage (*hashing*), le CRC, le chiffrement, etc. Pour faciliter la programmation de telles fonctions, l'IXP inclut des unités dédiées pour le brouillage et le calcul de CRC.

L'IXP2400 supporte deux types de mémoires externes, la QDR (*Quad Data Rate*) SRAM pour l'accès aux petites structures de données à faible latence utilisées dans les opérations de recherche (*lookup*), et la DDR (*Double Date Rate*) SDRAM pour la grande capacité de stockage qui est nécessaire pour stocker des paquets. Les espaces d'adresse SRAM et SDRAM sont mis en commun entre tous les ME et les données dans ces mémoires sont accessibles à chaque *thread* de ME. En outre, le processeur inclut une mémoire SRAM sur puce (16 KB) commun à tous les ME, et une petite mémoire locale (640 mots de 32 bits) par ME. Il y a 128 registres généraux et 640 registres de transfert de données disponibles dans chaque ME. Les MEs incluent également le matériel consacré aux fonctions suivantes : CRC de 16 et 32 bits, génération de nombres aléatoires, les temporisation matériels avec une granularité assez fine, multiplication et mémoire de type CAM (*Content Address Memory*) [Intel01].

C Les défis de la programmation

Les applications de traitement de protocoles de réseaux sont ciblées pour des débits de données spécifiques. Afin de satisfaire aux exigences de débit de sortie, un NP doit terminer les tâches de traitement d'un paquet avant qu'un autre n'arrive. Les conditions de pire cas, pour le débit de sortie, correspondent typiquement l'arrivée de paquets de taille minimum. Par exemple, pour le débit OC-48 (ou 2,4 Gbps), les paquets de taille minimum (46 bytes) arrivent chaque 100 cycles. Pour supporter l'arrivée un par un de paquets de taille minimum à la vitesse de la ligne, le NP doit achever le traitement du paquet en 100 cycles. De telles contraintes et leurs solutions potentielles posent quelques défis de programmation intéressants [Venkatachalam03] :

- **Respecter une limite temporelle déterministe pour le traitement des paquets** :

  La contrainte de traiter des paquets à la vitesse de la ligne affecte la programmation de manière significative. En effet, le temps maximum pour traiter un paquet de taille critique (taille minimale) est borné le temps nécessaire pour recevoir un autre paquet de taille minimale sur la ligne. Ceci signifie que nous avons besoin de concevoir le logiciel de telle manière que le nombre de cycles d'horloge pour traiter le paquet n'excède pas une limite haute. Par conséquent, il est important que le logiciel emploie les structures de données adéquates et soit conçu de manière à respecter la limite.

- **Masquer la latence de la mémoire et des E/S** :

  En plus des structures de données et des blocs de traitement rapides pour satisfaire contraintes de vitesse de ligne, un programmeur doit considérer la latence de la mémoire et des I/O, qui est beaucoup plus élevée que la quantité de temps de traitement disponible pour chaque paquet. Par conséquent, l'autre défi important en programmation d'un NP est l'utilisation de *multi-threads* matériels efficaces pour masquer les latences des E/S et de la mémoire. Plusieurs techniques telles que le *pipelining* et le *multi-processing* peuvent être utilisées pour cacher efficacement ces latences.

- **Mettre des paquets en ordre malgré le traitement parallèle** :

  L'autre défi significatif en programmation de NP est de réordonnancer les paquets après un traitement parallèle. Ceci peut être réalisé en utilisant deux techniques : numéroter les paquets et mettre en œuvre l'ordonnancement des *threads*. Si nous nous assurons que les *threads* s'exécutent fonctionnent dans un ordre exact, alors des paquets sont assignés aux *threads* dans l'ordre de leur arrivée et, par conséquent, le traitement des paquets s'achèvera dans l'ordre.

Sur le marché des processeurs de réseau, certains concepteurs préfèrent optimiser profondément une architecture pour en tirer le profit maximal. D'autres sont plus concernés par le temps de mise sur le marché et opteront pour un développement plus rapide en privilégiant une programmation de haut niveau efficace. Pour répondre à ces deux attentes, la plupart des fournisseurs proposent la panoplie d'outils suivante : une bibliothèque de sous-programmes parmi les plus couramment employés, un ensemble « assembleur-simulateur-débogueur » et un compilateur pour une programmation en langage C. La qualité de tous ces outils est très variable et c'est évidemment un point important à vérifier avant de choisir un NP. En outre, même si les outils sont bons, la tâche du programmeur sera d'autant plus laborieuse que l'architecture est complexe. Par exemple, programmer le Power NP4GS3 d'IBM demande de coordonner 16 micromoteurs et plusieurs coprocesseurs spécialisés, c'est si compliqué que, souvent, IBM écrit la majeure partie du code pour les utilisateurs. Néanmoins, sur ce plan-là aussi, les fournisseurs de NP progressent très rapidement, les concepteurs systèmes sont de plus en plus soutenus et la programmation gagne en efficacité (en temps et en performance). Il existe aussi des langages, dits de quatrième génération, qui permettent de travailler à un plus haut niveau d'abstraction. C'est le cas du langage FPL (*Functional Programming Language*) pour le processeur FPP d'Agere qui peut, en une seule instruction, extraire tous les paquets à partir d'une série d'adresses Internet [Trézéguet02]. Un autre exemple de tels langages est le NCL (*Network Classification Language*) d'Intel pour programmer les algorithmes de classement de paquets.

2.5 *System On Chip*

2.5.1 Multi-processeurs mono-puce

Le terme multi-processeurs mono-puce ou *Multiprocessors system on chip* représente les produits intégrant plusieurs processeurs tels que RISC, DSP, ASIC, et des réseaux de communication complexes tels que les bus hiérarchiques, les bus avec protocole TDMA, les connexions point à point, et même les réseaux de commutation de paquets. Pour réduire les coûts de production et améliorer la performance, ils sont intégrés dans une seule et même puce. Il est prévisible que ce type de production devienne l'orientation principale de l'industrie des semi-conducteurs, dans un proche futur.

La figure 2.12 montre une architecture multi-processeur typique de SoC (*System On Chip*) avec des processeurs hétérogènes, la communication se faisant par le réseau d'interconnexion. Une différence principale avec l'architecture classique des ordinateurs est que ce modèle distingue deux types de processeurs (en fonction de leur utilisation) : ceux qui exécutent les applications (ou fonction) générales et ceux dédiés à l'exécution de fonctions spécifiques spécialement implantées dans le matériel. La programmation et l'interface de ces deux types de processeurs sont tout à fait différentes.

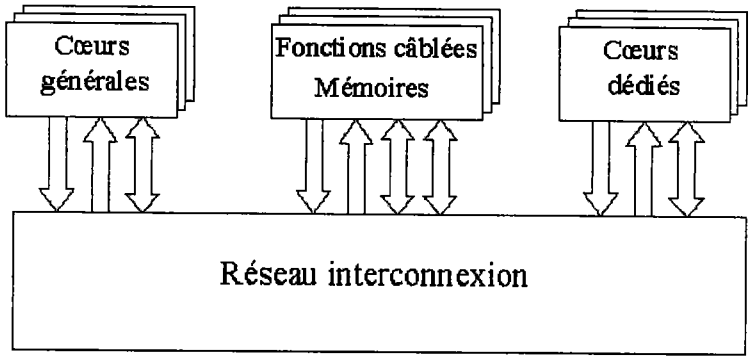


FIG. 2.12: Architecture de multi-processeurs SoC

De plus, les processeurs dédiés exigent la communication spécifiée par l'application de gestion de la mémoire pour optimiser la performance. Cette optimisation spécifiée par l'architecture est généralement reliée au domaine d'application. Par exemple, des interfaces spécifiques sont exigées pour respecter les contraintes de coût/performance (par exemple : surface, consommation d'énergie, temps d'exécution) [Cesario02].

Le grand problème de ces architectures, est d'en maîtriser leur complexité lors de conception. Des approches de co-design ont été introduites ces dernières années pour faire face à ce défi. Il s'agit de méthodologies et d'outils de CAO qui permettent la synthèse automatique des parties matérielles, des parties logicielles, et de la communication en partant d'une spécification de haut niveau du système complet.

Ainsi les SoC exigent des interfaces matériel/logiciel spécifiques pour implanter le réseau d'interconnexion dédiée aux applications. Ces interfaces permettent la communication par les protocoles appropriés entre les parties matérielles telles que les cœurs de processeurs, les bancs de registres ou les mémoires, et les parties logicielles comme le système d'exploitation. Afin d'augmenter la flexibilité du système global les SoCs intègrent des processeurs avec différents jeux d'instructions pour exécuter les fonctions dédiées. La complexité d'un tel code peut être plus élevée que celle du matériel.

Dans les SoC multi-processeurs, plusieurs processeurs et composants mémoires peuvent communiquer les uns avec les autres. Pour cela des moyens de stockage intermédiaire (et temporaire) des données, comme le tampon de communication sont requis. Dans certains cas, les tampon peuvent prendre une partie significative de la surface du circuit.

Dans le cas, par exemple, de systèmes multimédia MPEG2 et 4, les besoins de communication de données de grande taille peuvent conduire à des surcoût importants pour l'implantation des tampons. Par conséquent, pour réduire la surface, la réduction de la taille des tampons de communication est requise. Les algorithmes d'ordonnancement et de partage des ressources de stockage entre les différentes transactions de communication peuvent résoudre une partie du problème [Cho03].

La synchronisation de futurs SoC à partir d'une source d'horloge unique aves une dérive et le biais négligeable sera extrêmement difficile. Le paradigme le plus adapté à

la synchronisation de ces puces, globalement asynchrones et localement synchrones, est d'employer différents horloges. En l'absence d'une référence simple de synchronisation, les puces SoC deviennent des systèmes distribués. Les composants lancent des transferts de données de façon autonome, selon leurs besoins. Le modèle global de communication est alors entièrement distribué, avec peu ou pas de coordination globale [Benini02].

2.5.2 Réseau sur puce(*Network on Chip*)

La fiabilité d'interconnexion entre les composants dans un système sur puce est une contrainte principale. L'interconnexion physique représente toujours une limite de la performance et éventuellement un coût en consommation d'énergie. La conception de SoC complexes exige une approche modulaire basée sur des composants matériels et logiciels, et des approches d'interconnexion très fiables mais peu coûteuses. Dans la pratique, le but est la conception d'un micro-réseau reconfigurable. L'idée de micro-réseau ou *Network-on-chip* (NOC) est un nouveau paradigme de la conception de SoCs proposé par beaucoup de chercheurs [Benini02][Sgroi01][Kumar02] et il est prévisible que ce modèle soit un choix architectural important pour l'avenir des SoCs. Ainsi, dans [Virtanen03], les auteurs proposent une interface NoC pour un processeur de protocole.

Les architectures proposées pour les NoC représentent une plateforme générale et fixe qui peut être utilisée pour un grand nombre de SoC. Les modèles de référence des différentes en couches des réseaux informatiques sont utilisés par toutes les architectures proposées de NoC. Il est prévu que cette architecture de NoC facilite la réutilisation de divers éléments conceptuels matériels et logiciels du système, réduisant le temps de conception et la vérification. Cependant, la recherche sur les NoC en est toujours à ses débuts. Les NoC constituent une abstraction de la communication entre les composants et doivent répondre à des exigences de qualité de service tels que la fiabilité, la performance, les limites de consommation d'énergie et des délais de communication sur puce.

Il existe des différences entre les NoC et les réseaux informatiques : par exemple, les NoC montrent moins de non-déterminisme. Les réseaux locaux et à haute performance comme ceux développés pour les multiprocesseurs à grande échelle, peuvent avoir des conditions de fonctionnement et des contraintes semblables. Cependant, certaines caractéristiques distinctives, telles que des contraintes d'énergie, de temps de conception et de spécialisation sont uniques aux réseaux de SoC [Benini02].

La conception de réseaux informatiques a été traditionnellement découplée des applications spécifiques et est fortement influencée par la standardisation et la compatibilité. Dans les NoCs, ces contraintes sont moins restrictives parce que les fournisseurs conçoivent le réseau entier sur le silicium. Par conséquent, seule l'interface réseau des nœuds terminaux exige une standardisation. Les concepteurs peuvent,par exemple, concevoir l'architecture de réseau pour une classe d'applications.

Il existe différentes méthodes pour interconnecter les composants dans les NoC. Cette interconnexion doit permettre d'accéder aux ressources partagées. Dans les technologies courantes, le bus *backplane* est l'exemple le plus commun pour les structures avec ressources partagées. Ses avantages sont les surcoûts faibles, et le fait qu'il se compose de quelques bus actifs (les maîtres) et de beaucoup de bus passifs (les esclaves) qui répondent seulement aux demandes des maîtres.

Le réseau direct (ou point à point) utilise des liaisons directes entre les composants pour les interconnecter. Dans cette architecture, chaque nœud est connecté directement à un nombre limité de nœuds voisins. Cette architecture contient un bloc d'interface réseau, souvent appelé routeur, qui gère la communication et la relie directement aux routeurs

des nœuds voisins. Les réseaux d'interconnexion directe sont fréquemment utilisés pour implanter les systèmes à grande échelle parce que la bande passante de communication augmente également quand le nombre de nœuds augmente dans le système. Les réseaux indirects ou basés sur la commutation sont une alternative aux réseaux point à point pour avoir un réseau d'interconnexion redimentionnable. Dans ces réseaux, une liaison entre les nœuds doit passer par un ensemble de commutateurs. Les commutateurs fournissent seulement une connexion programmable entre leurs ports, installant un chemin de communication qui peut changer au cours du temps. Par exemple, le FPGA Xilinx Virtex II, avec la capacité d'implanter les DSP configurables contient des blocs logiques configurables (CLB : *Configurable Logic Block*), des RAM, des multiplieurs, des commutateurs et des tampons d'entrée-sortie. Chacun de ces éléments programmables est relié à une matrice de commutateurs, permettant plusieurs connexions à la matrice de routage général [Benini02].

La conception de NoC est basée sur l'abstraction, avec l'approche du modèle de couches OSI. L'application peut être déployée de plusieurs manières, sur des architectures homogènes ou hétérogènes. Ainsi, le trafic produit par l'application et par la couche présentation peut être de différentes formes. Les couches session et transport gèrent les interfaces entre le réseau et les éléments produisant du trafic. Le travail des couches réseau et liaison de données est de transmettre le message de la source à sa destination. Les deux points importantes ici sont : (i) la topologie et (ii) le protocole. La topologie concerne la structure du réseau de communication dans les systèmes sur puce. Le protocole indique comment les liens sont employés. De nombreuses combinaisons de topologies et de protocoles existent pour plusieurs modèles de trafic permettant une communication efficace. La couche physique porte les signaux électriques réels représentant le trafic.

2.6 Conclusion

Ce chapitre a brièvement introduit les différentes architectures de processeurs. Pour cela, trois grandes catégories d'architectures ont été présentées. Dans la première partie, les deux architectures et scalaires générales RISC et CISC ont été étudiées de manière comparative. Beaucoup de processeurs généraux utilisent une architecture hybride RISC-CISC. Cependant, pour une application particulière, l'une peut être préférable à l'autre. Dans la deuxième partie, nous avons présenté les méthodes architecturales parallèles. Certaines méthodes comme le MIMD et le parallélisme d'instructions sont plus utilisées et plus performantes dans les domaines d'application générales, alors que les autres, comme le SIMD et le parallélisme de données, sont intéressantes pour les applications spécifiques. Globalement, on peut trouver deux types de traitement dans un processeur spécialisé aux applications de réseau : les traitements généraux et le traitement du signal. Enfin, les caractéristiques principales de deux architectures de processeurs, les processeurs de réseau et les DSP, correspondants aux applications réseau, ont été introduites.

Deuxième partie

Étude architecturale d'un processeur de protocoles

Chapitre 3

Evaluation des architectures

3.1 Introduction

La conception d'une architecture de processeur spécialisée dépend des caractéristiques des applications auxquelles le processeur est dédié. Pour développer une architecture générale et adaptée au niveau du traitement des protocoles de réseau, il faut étudier les spécifications principales des protocoles. L'objectif est répertorier les principales tâches des protocoles. Ensuite, il s'agit d'identifier les tâches les plus communes et les plus critiques du point de vue temporel. Pour cela, nous avons étudié quatre protocoles dominants dans le domaine des réseaux. TCP/IP, PPP, ATM et les protocoles courants de la famille IEEE 802 ont été choisis afin d'en extraire les opérations communes ou ayant peu de différences. Les description générales de ces protocoles se trouvent dans le chapitre précédent. Ici, nous détaillons les opérations requises pour traiter ces protocoles. Une pile de couches utilisant les protocoles dominants est représentée dans la figure 3.1. Lorsque un train de bits arrive au terminal de réseau, il est traité par plusieurs protocoles avant d'être accessible à l'utilisateur.

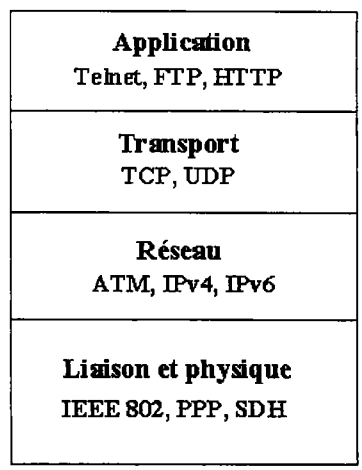


FIG. 3.1: La pile des protocoles dominants

Dans ce chapitre, nous étudions tout d'abord les opérations requises pour traiter ces protocoles dans les couches bases (réseau, liaison de données, MAC et physique), puis nous regroupons les tâches du traitement. Nous classons les traitements de protocoles dans ces couches en deux parties principales : les traitements orientés DSP (*Digital Signal Processing*) et les traitements non orientés DSP (ou généraux).

Dans la suite, nous évaluons les performances potentielles de différentes architectures de processeurs généralistes pour l'exécution de tâches générales. Sur la base d'une modélisation des algorithmes concernés par chaînes de Markov, un banc de simulation est proposé pour implanter la méthodologie d'évaluation. L'analyse des résultats obtenus par le banc de simulation nous permet de déterminer les architectures les plus appropriées par type d'algorithme.

3.2 Méthodologie

Nous proposons ici une méthodologie pour évaluer les différentes architectures. L'objectif est de vérifier la performance d'exécution des protocoles de coches 1,2 et 3 (physique, liaison de données et contrôle d'accès au medium (MAC) et réseau) sur différentes architectures de processeurs. Pour ce faire, un ensemble d'algorithmes représentatifs de ces protocoles a été choisi et exécuté sur quatre architectures génériques, afin d'en étudier les performances. La figure 3.2 montre le flot de la méthodologie proposée.

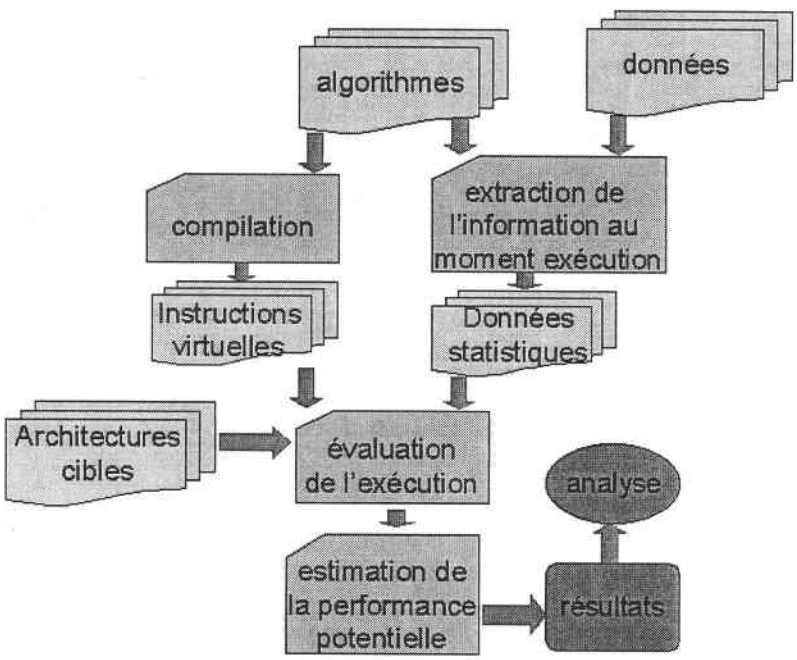


FIG. 3.2: Flot de la méthodologie proposée

Dans un premier temps, les algorithmes représentatifs (implantés en langage C) sont transformés en chaînes de Markov équivalentes. L'information statistique de la chaîne de Markov est obtenue à partir de l'évaluation de l'exécution sur n'importe quel processeur

conventionnel, car elle dépend uniquement des données d'exécution et non de l'architecture de processeur choisie. Dans le même temps, un compilateur produit du code virtuel indépendant de la plateforme. À cette fin, un jeu d'instructions virtuelles est considéré, non lié à un modèle architectural spécifique (RISC, CISC, etc.). Certaines mesures sont effectuées sur le code virtuel produit à chaque algorithme. Ces mesures sont transformées en mesures dynamiques (ou bien les mesures d'exécution) en utilisant d'une modélisation par chaînes de Markov. Un estimateur évalue ensuite le nombre de cycles nécessaires pour exécuter chaque instruction virtuelle sur les architectures cibles considérées.

L'interprétation d'une telle quantité d'information exige des méthodes adéquates. Ces méthodes ont généralement pour but de développer des modèles prévisionnels, de réduire la taille des données, de faire de la segmentation ou bien de classer des individus caractérisés. Dans le cas présent, les mesures obtenues sont étudiées en utilisant des méthodes d'analyse de données multidimensionnelle notamment pour mettre en évidence les liaisons existantes entre les mesures effectuées sur le code virtuel et un ensemble de critères. Cette analyse nous permet de déterminer les architecture les plus appropriées par type d'algorithme.

Il est à noter que la méthode proposée représente une estimation de la performance des différentes architectures. Même s'il est possible d'atteindre une précision importante pour les résultats, cela complique les calculs de façon non négligeable. Nous avons donc essayé d'atteindre une bonne approximation en évitant des calculs trop complexes. D'autre part, les architectures sur lesquelles seront exécutées les applications réelles sont évaluées à partir d'algorithmes de test (benchmarks). Par conséquent, ces algorithmes doivent être représentatifs des applications réelles.

Dans la section suivante, nous étudions les opérations requises pour traiter les protocoles de couches basses sélectionnés.

3.3 Etude des protocoles (extraction des tâches principales)

Dans cette section, sont étudiées les protocoles dominants des couches réseau, liaison de données, MAC et physique (IP, ATM, IEEE 802, et PPP) pour identifier les tâches nécessaires au traitement de ces protocoles. Nous nous concentrons sur les points communs entre ces tâches.

3.3.1 *Internet protocol (IP)*

Le protocole IP(*Internet Protocol*), est le protocole de la couche réseau et existe actuellement en deux versions IPv4 et IPv6. IP est situé au-dessous de la couche transport (protocoles TCP). L'unité de données dans IP est le paquet. Le service de remise de paquets est non fiable et un paquet peut être perdu sans qu'Internet ne le détecte. Un paquet IP est formé de deux parties, l'en-tête et les données. Les deux parties peuvent avoir une taille variable. En pratique, la taille des paquets est d'environ 1500 octets. La figure 3.3 représente la structure du paquet IPv4.

Le traitement principal dans le protocole IP s'effectue sur l'en-tête. Celui-ci est composé de d'une partie fixe de 20 octets et une partie optionnelle de taille variable. Pour identifier les opérations d'analyse de l'entête requises, nous décrivons la fonction de chaque partie de l'en-tête.

Le champ « version » indique la version du protocole IP. La longueur de l'en-tête

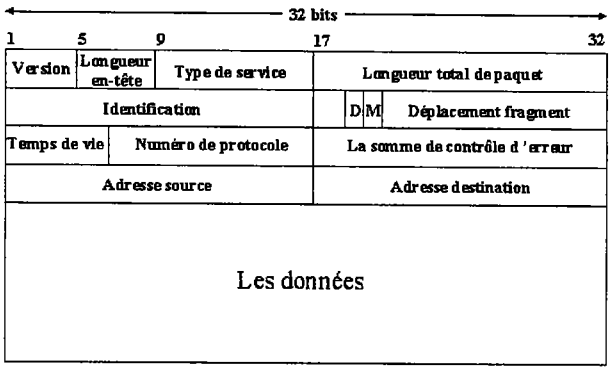


FIG. 3.3: Structure du paquet IP

permet de retrouver l'emplacement de début des données. Le champ « type de service » indique les spécifications requises pour la transmission de paquets. Le champ « longueur totale » indique la taille totale des paquets. Le champ « identification » identifie le fragment auquel le paquet appartient. Les deux bits de contrôle, « D » et « M » sont utilisés pour contrôler le processus de fragmentation. Si D est égale à « 1 », cela signifie que le paquet contient un datagramme non fragmenté. Le bit « M » indique qu'il existe encore des paquets appartenant à ce datagramme. Le « déplacement fragment » indique la position de ce paquet dans le datagramme. Le « temps de vie » spécifie (en secondes) le temps au-delà duquel paquet est détruit. Le « numéro de protocole » indique quel protocole dans la couche transport prend le datagramme après réassemblage. La « somme de contrôle » permet de déterminer si la transmission du paquet s'est effectuée correctement ou non. Cette valeur est calculée à chaque manipulation de l'en-tête du paquet. À la fin de l'en-tête, on trouve les adresses d'émetteur et de récepteur.

Les machines qui sont connectées via Internet sont identifiées par une adresse unique de 32 bits. Une adresse IP 32 bits est habituellement représentées par quatre numéros comme dans 130.232.125.56. Cette adresse est constituée de deux parties : un identificateur de réseau et un identificateur de machine dans ce réseau. L'espace d'adresses d'IP est divisé en quatre classes : A,B,C,D. Cette répartition est basée sur le nombre de bits spécifié pour chaque partie de l'adresse. Par exemple, dans la classe A, 7 bits sont utilisée pour la partie réseau et 24 bits pour la partie machine (ou hôte). En effet chaque réseau de classe A peut contenir jusqu'à 16777216 hôtes et il peut y avoir 128 réseaux de classe A.

Puisque IP est un protocole de couche réseau, il est nécessaire de transformer les adresses IP en adresses qui puissent être comprises et traitées par la couche liaison de données. Pour cette transformation (et son inverse, la transformation de l'adresse de couche liaison de données en adresses IP), la résolution d'adresse est nécessaire.

Dans Internet, deux protocoles sont employés dans ce but : ARP (*Address Resolution Protocol*) et RARP (*Reverse Address Resolution Protocol*). Une machine qui souhaite envoyer un message à une autre machine dont elle connaît l'adresse IP (c'est-à-dire l'adresse réseau), pose une question dans son réseau local demandant l'adresse " physique " (adresse de couche liaison correspondantes). Si la destination est dans le même LAN, il répond avec son adresse physique (par exemple l'adresse Ethernet). Autrement, le datagramme peut être envoyé, par exemple, à une adresse physique par défaut (par exemple le routeur LAN). La machine à l'adresse physique par défaut fait alors suivre au routeur de

réseau local de la machine destination [Tanenbaum03].

A Les tâches principales

En considérant les explications données sur le protocole IP, on peut distinguer quelques caractéristiques importantes dans le traitement de ce protocole :

1. Une des tâches importantes est l'analyse de l'en-tête de paquets. Pour cela, chaque champ du train de bits est comparé avec les valeurs possibles pour prendre une décision. Cette décision spécifie le chemin à suivre pour continuer le traitement du protocole.
2. Une mémoire rapide est nécessaire pour stocker les fragments de données entrants et éviter de perdre des fragments à cause d'une congestion éventuelle.
3. Pour traiter la somme de contrôle de l'en-tête, les opérations de complément à «1» et d'addition de mots de 16 bits sont nécessaires. Cette valeur devrait être modifiée après chaque traitement de l'en-tête.

3.3.2 Protocole ATM

Comme son nom l'indique, l'ATM est un protocole asynchrone. Cela signifie que des paquets ATM, appelés « cellules », sont envoyés seulement quand il y a de l'information à transmettre. Dans les protocoles synchrones, en revanche, les paquets sont envoyés tout le temps à intervalles réguliers. Ceci signifie que dans les transmissions synchrones les paquets sont envoyés même s'il n'y a rien à envoyer. Dans les réseaux ATM, la hiérarchie de connexion est à deux niveaux : les circuits virtuels et les conduits virtuels. Plusieurs circuits virtuels forment un conduit virtuel, tout comme plusieurs fils de cuivre isolés forment un câble.

Les cellules ATM peuvent être envoyées indépendamment ou L'en-tête d'une cellule ATM est constituée des 5 premiers octets et la charge utile de 48 restant. L'en-tête d'ATM contient les 5 premiers octets et la charge utile des cellules contient les 48 derniers octets. L'en-tête représentée dans la figure 3.4 contient l'information sur le circuit virtuel, VCI (*Virtuel Circuit Identifier*) et le conduit virtuel, VPI (*Virtuel Path Identifier*) à travers lesquels la cellule doit être transmise. Puisque l'information sue le conduit est un nombre entier de 8 bits et l'information sur le circuit un nombre entier de 16 bits, il est facile de déterminer qu'un hôte peut avoir jusqu'à 256 conduits virtuels entrants et sortants, chacun contenant 65 536 circuits virtuels. Le nombre de circuits virtuels disponibles est plus petit parce que quelques valeurs VCI sont réservées pour les opérations de service du réseau, telles que l'établissement d'une connexion.

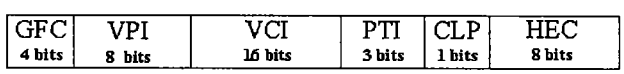


FIG. 3.4: En-tête d'une cellule ATM sur l'interface UNI (*User Network Interface)*

Le champ GFC (*Generic Flow Control*) permet de contrôler les flux de cellules entrant, de les multiplexer et de diminuer les périodes de congestion du réseau de l'utilisateur final. Selon le code GFC, différentes procédures sont définies pour le contrôle de flux. Ces procédures effectuent aussi le contrôle de la qualité de service dans le réseau de l'utilisateur final.

L'identificateur de type de charge utile, PTI (*Payload Type Identifier*) indique si la cellule est une cellule utilisateur ou une cellule de gestion de maintenance/ressource. En outre, les problèmes dans la transmission de cellules (congestion, etc) sont indiqués dans ce champ.

Le bit CLP (*Cell Loss Priority*) indique la priorité de la cellule. Les cellules avec une priorité de perte de cellules marqué à 1 sont ignorées avant les cellules avec CLP=0.

Le champ de contrôle d'erreur d'en-tête HEC (*Header Error Check*) est utilisé pour la délimitation de la cellule et le contrôle d'erreur dans l'en-tête de la cellule. Le HEC sert à détecter les erreurs et à les corriger. Le champ HEC est calculé à partir du polynôme constitué des bits de l'en-tête, à l'exception du champ HEC. Dans cette représentation polynômiale un train de bits de longueur n, soit « $u_n...u_2u_1$ » est associé au polynôme : $u(x) = u_n x^{n-1} + ... + u_2 x + u_1$. Ce polynôme est divisé par le polynôme générateur $x^8 + x^2 + x + 1$. Le HEC est produit en ajoutant 01010101 au reste de la division [Pujolle03].

Supposons qu'un message soit envoyé par une application à une application sur une autre machine via un réseau ATM. Le message transite de l'application à la couche d'adaptation AAL. Comme nous avons expliqué dans le chapitre 1, cette couche contient deux sous-couches CS et SAR. La sous-couche de convergence (CS) peut donner au message un en-tête et/ou une terminaison, selon les besoins de l'application et de la partie spécifique au service de la sous-couche de convergence. Ensuite le message, les en-têtes et les terminaisons supplémentaires sont découpés en unités de données de 48 octets. Ces unités sont passées à la sous-couche de segmentation et de réassemblage (SAR). La sous-couche SAR peut ajouter ses propres en-tête et/ou terminaison à chaque unité de données, puis elle les passe à la couche ATM. Il n'y a aucun contrôle d'erreurs ou de flux dans la couche ATM. AAL fournit des services aux programmes d'applications pour découper les données à l'émission et les réassembler à la réception, en vue de leur transport dans les cellules.

Lors de l'arrivée de la première unité de données, la couche ATM commence à établir un circuit virtuel avec l'hôte de réception. La couche ATM envoie une cellule de demande pour établir une voie de transfert. Les commutateurs de voie de transmission renvoient l'information de la voie virtuelle sélectionnée à l'unité qui l'a demandée. L'information de la voie virtuelle est alors incluse dans l'en-tête des cellules (VPI et VCI) de toutes les unités de données. Quand le circuit virtuel est établi, la couche ATM produit uniquement les cellules de 53 octets pour la couche physique, l'une après l'autre. La couche ATM ne fournit pas d'acquittement de réception des cellules à distance : ATM a été conçu pour une utilisation sur les réseaux à fibres optiques, qui sont fortement fiables.

Les cellules ATM peuvent être envoyées directement par un réseau ou par un porteur intermédiaire. La couche ATM fournit une séquence de cellules, et la sous-couche TC (*Transmission Convergence*) de la couche physique fournit une interface uniforme entre la sous-couche PMD (*Physical Medium Dependent*) et la couche ATM. La sous-couche PMD est chargée de mettre les bits dans un format correct pour le support physique. À l'extrémité de réception, l'exact opposé de ce qui a été décrit ici doit être fait.

La difficulté principale se rapporte à la synchronisation du train de bits entrant. Ce n'est pas un problème si les cellules ATM sont transportées par un autre protocole de la couche physique ayant a sa propre méthode de synchronisation. Cependant, si des cellules ATM sont envoyées directement sur le support physique, alors une méthode de synchronisation doit être utilisée. Pour cette tâche, le champ de contrôle d'erreur d'en-tête (HEC) des cellules ATM est employée. La couche TC maintient un registre à décalage de 40-bits

pour le train de bits entrant. La valeur de 40 bits est examinée pour déterminer si elle constitue une en-tête ATM valide (les 8 derniers bits sont cohérents avec les 32 premiers bits). Si les 8 derniers bits ne forment pas un HEC valide, alors le registre est décalé à gauche d'un bit, et le bit suivant dans le train entrant est inséré comme le bit le plus à droite dans le registre. Ceci est réalisé jusqu'à ce qu'un HEC valide soit trouvé.

Quand le contenu du registre (qui peut toujours être une séquence aléatoire), constitue une en-tête ATM valide, les 394 bits suivants (48 octets, charge utile possible des cellules) sont jetés et 40 nouveaux bits sont chargés dans le registre. Le processus est répété jusqu'à ce que des en-têtes valides consécutives soient trouvées.

La méthode précédente décrite fonctionne de la même manière pour la correction d'erreur : après avoir des en-têtes valides consécutives, l'opération normale est commencée. Dans l'opération normale, si des en-têtes consécutives inadmissibles sont produites, le système va de nouveau à la chasse, décalant le train de bits entrant d'abord pour trouver un HEC valide t puis pour trouver les en-têtes consécutives valides. Ce processus est représenté par une machine états à trois états dans [Tanenbaum03].

ATM s'annonce dans le futur, comme une technologie potentielle pour les communication par câble, appropriée pour les systèmes téléphoniques et pour les systèmes de transfert de données. Un des avantages d'ATM est la taille relativement petite et fixe des unités de transmission. Pour les besoins de transfert de voix et de video, il est beaucoup plus important d'avoir une voie de transmission à grande vitesse plutôt que d'avoir une voie de vitesse inférieure mais sans erreur. En outre, en raison de l'utilisation de la fibre optique pour des transmissions ATM, et malgré des mécanismes minimaux de correction et détection d'erreurs, le taux d'erreur est très bas.

A Les tâches principales

Les caractéristiques de ce protocole demandent les tâches de traitement spéciales comme celles décrites ci-après :

1. L'en-tête de chaque cellule ATM est analysée pour détecter et corriger les erreurs et pour extraire les informations de routage (VCI et VPI). Ces tâches exigent la reconnaissance du train de bits : un champ dans une en-tête entrante est comparé aux valeurs possibles en mémoire et une décision est prise. En outre, le calcul de l'HEC exige une unité de calcul de division polynômiale.

2. Selon le résultat de reconnaissance du train de bits, il faut déplacer rapidement la charge utile des cellules ATM dans une autre place de la mémoire ou dans une autre couche dans le protocole. Les techniques rapides de transfert de données sont nécessaires dans ce but.

3. En cours de réception d'un train de bits, le décalage rapide bit-par-bit est nécessaire pour le processus de synchronisation et pour trouver les frontières des cellules.

4. L'information venant des couches supérieures doit être découpée en blocs de données de 48 octets puis une en-tête de 5 octets doit être ajoutée à chaque bloc de données. Ceci peut être effectué en implantant des routines optimisées dans ce but : écriture de l'en-tête en début de cellule, lecture de 48 octets de données depuis la mémoire à la position du pointeur, mise à jour du pointeur pour positionner au début des prochains 48 octets, écriture des 48 octets à la droite de la cellule. À la réception, les cellules doivent être réassemblées pour constituer des plus grandes unités de données.

3.3.3 Protocoles IEEE 802

Les protocoles IEEE 802 définirent les normes LAN pour les couches physiques et liaison du modèle OSI. Ces normes divisent la couche Liaison en deux sous-couches : contrôle d'accès au support (MAC) et contrôle des liaisons logiques (LLC : *Logic Link Control*).

L'IEEE a défini les fonctionnalités de la sous-couche LLC dans la norme 802.2 et celles de la sous-couche MAC (*Medium Access Control*) et de la couche Physique dans les normes 802.x [Pujolle03].

Le protocole 802.3 définit des normes pour les réseaux en bus logique, qui utilise la méthode d'accès CSMA/CD notamment Ethernet. En outre, les versions 802.3u et 802.3z définissent les spécifications pour Fast Ethernet et Gigabit Ethernet, respectivement [Pujolle03].

Le protocole 802.4 définit des normes pour les réseaux en bus utilisant le passage de jeton. Il s'agit d'une disposition en bus qui utilise un message de diffusion générale. Chaque ordinateur reçoit toutes les données, mais seuls les ordinateurs concernés répondent au message de diffusion générale. Un jeton circulant sur le câble détermine qui a le droit d'émettre [Pujolle03].

Le protocole 802.5 définit des normes pour les réseaux en anneau utilisant le passage de jeton. Il s'agit d'un réseau en anneau logique qui transmet à 4 Mb/s ou 16 Mb/s. Malgré le nom d'anneau, ce réseau utilise un concentrateur et a l'apparence d'une étoile. Un jeton circulant sur le câble détermine qui a le droit d'émettre [Pujolle03].

Le protocole 802.6 définit les méthodes d'accès DQDB pour les WAN. Les normes 802.7 à 802.10 concernent les différentes services de LAN et WAN et à partir de 802.11 les normes concernent le domaine de réseaux sans fil.

La communication sans fil évoque de nouveaux problèmes dans le domaine des réseaux, tels que le bruit et les interférences accrus, le recouvrement de LAN et les besoins de sécurisation de la connexion. Les fonctionnalités de la sous-couche MAC sont à l'origine des tâches les plus critiques des réseaux LAN. La problématique du "sans-fil" cet aspect. Pour cela, dans la suite nous étudions les propriétés du protocole de la sous-couche MAC de 802.11.

Ce protocole est conçu pour fonctionner aux vitesses de quelques dizaines de Mbps. Le medium sans fil peut être l'infrarouge à une longueur d'onde de 850 à 950 nm, le FHSS (*Frequency Hopping Spread Spectrum*) dans la bande de 2,4 GHz, le DSSS (*Direct Sequence Spread Spectrum*) dans la bande 2,4 GHz et l'OFDM (*Orthogonal Frequency Division multiplexing*) dans la bande 5.2 GHz. Grâce au codage OFDM, des débits compris entre 6 et 54 Mbps sont atteints [Pujolle03].

La sous-couche MAC de 802.11 est très proche de celle de 802.3 dans sa conception : elle est conçue pour supporter de multiples utilisateurs sur un support partagé, en faisant l'émetteur écouter le support avant d'y accéder. Pour les LAN Ethernet 802.3, le protocole CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) régule l'accès des stations Ethernet au câble ; il détecte et gère également les collisions qui se produisent lorsque deux nœuds stations machines ou plus tentent de communiquer simultanément sur le LAN.

Dans un LAN sans fil, la détection des collisions est impossible parce que pour détecter une collision, une station doit être capable de transmettre et d'écouter en même temps. Or, dans les systèmes radio, il ne peut y avoir facilement de transmission et d'écoute

simultanées.

Pour prendre en compte cette différence, le standard 802.11 fait appel à un protocole légèrement modifié, baptisé CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*), ou à la fonction DCF (*Distributed Coordination Function*). Le protocole CSMA/CA tente d'éviter les collisions en imposant un accusé de réception des paquets (ACK), ce qui signifie que, pour chaque paquet de données arrivé, un paquet ACK est émis par la station de réception.

Ce protocole CSMA/CA fonctionne de la manière suivante : une station qui souhaite émettre écoute le canal de transmission et, si aucune activité n'est détectée, attend un temps aléatoire IFS (*Inter Frame Space*) avant d'émettre ; si le support est toujours libre après l'IFS, la station peut commencer à transmettre le paquet de données. Si le paquet est intact à la réception, la station réceptrice émet une trame ACK qui, une fois reçue par l'émetteur, met un terme au processus. Si la trame ACK n'est pas détectée par la station émettrice (parce que le paquet original ou le paquet ACK n'a pas été reçu intact), une collision est supposée avoir en lieu et le paquet de données est retransmis après attente d'un autre délai aléatoire.

Les fonctions nécessaires pour réaliser un accès sur une interface radio sont les suivantes [Pujolle03] :

- procédure d'allocation du support
- adressage des paquets
- formatage des trames
- contrôle d'erreur CRC
- fragmentation-réassemblage

Les stations qui fonctionnent sous la forme de LAN 802.11 forment un BSS (*Base Station Subsystem*). Chaque station et chaque BSS ont leurs propres adresses de réseau. Dans la norme 802.11, deux modes de fonctionnement existent : un fonctionnement avec contention ou le DCF (*Distributed Coordination Function*) et un fonctionnement sans contention ou le PCF (*Point Coordination Function*). Dans le fonctionnement avec contention, les stations essaient d'accéder au réseau en écoutant le support en CSMA/CA, comme expliqué ci-dessus. Le DCF est similaire aux réseaux traditionnels supportant le *best-effort*, ce qui signifie que le réseau fait de son mieux pour écouler le trafic. Le fonctionnement sans contention est facultatif et exige un contrôleur de point d'accès et est fondé sur l'interrogation à tour de rôle des stations, ou *polling*. Cette méthode d'accès garantit que chaque client obtient toujours une certaine quantité de bande passante [Pujolle03].

Si les deux sont en service simultanément, ils fonctionnent dans le réseau de sorte à ce qu'un mode soit toujours suivi de l'autre. Le temps assigné à chaque mode change dynamiquement selon les besoins des stations pour chaque type de trafic dans le réseau LAN. Les espaces inter-trames pour les stations fonctionnant dans différents modes sont définis de telle manière que les stations fonctionnant en mode sans contention disposent d'une période plus courte que les stations fonctionnant en mode avec contention. De cette façon, les stations qui fonctionnent en mode avec contention ne peuvent pas interrompre le fonctionnement sans contention.

A La trame 802.11

Chaque trame de la couche MAC de 802.11 comprend les parties suivantes (la Figure 3.5) :

- Une en-tête MAC qui contient les champs contrôle de la trame, durée, adresse et type de la séquence. La longueur de l'en-tête est de 30 octets.
- Un champ de longueur variable avec la charge utile réelle. La longueur de ce champ doit être comprise entre 0 et 2312 octets.
- Un champ de contrôle de trame (*FCS : Frame Check Sequence*) qui contient une somme contrôle CRC-32 de tous les champs de la trame. La longueur de la FCS est de 4 octets. Le calcul du FCS et les polynômes utilisés sont définis dans [IEEE802.11].

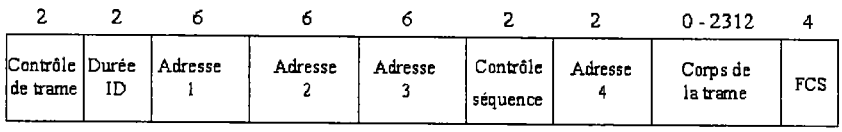


FIG. 3.5: La trame MAC de IEEE 802.11

Le champ de contrôle de la trame inclut des sous-champs pour la version du protocole, le type et le sous-type de la trame, le contrôle de la fragmentation et la gestion de la puissance. Il est prévu de passer la version de protocole de 0 à une autre valeur en cas de changement important dans les spécifications du protocole. Les sous-champs de type et de sous-type indiquent la fonction de la trame, par exemple, si c'est une trame de données, une trame de contrôle ou de gestion et quel est sa fonction principale (par exemple demande d'association/dissociation ou réponse).

Dans le mode DCF, le champ Durée/ID représente un type de trame dépendant de la durée de la trame(1..32767). Si le système est dans le mode PCF, alors ce champ contient la valeur fixe 32768. D'autres valeurs sont réservées pour une utilisation future ou utilisées pour porter l'information d'association dans les trames d'association.

Selon le type de trame, les champs d'adresses sur 48-bits peuvent présenter n'importe lequel des contenus suivants : un identificateur de BSS (l'adresse sur 48-bit identifiant le service de base réglé dans lequel la station fonctionne), une adresse de destination, une adresse de source, une adresse d'émetteur et une adresse de récepteur. Le format des adresses 48-bit est identique celui d'Ethernet.

Le champ de contrôle de la séquence possède deux sous-champs : le numéro du fragment (4 bits) et le numéro de séquence (12 bits). Si les données du service ou de la gestion doivent être réduites en fragments, le numéro de séquence contient une valeur spécifique d'identification pour l'unité de service ou de gestion. Le numéro de fragment est placé à zéro dans le premier et est incrémenté de un pour chaque fragment successif. Ce numéro reste constant dans toutes les retransmissions du fragment [IEEE802.11].

B Les tâches principales

Les caractéristiques de ce protocole demandent des tâches de traitement spécifiques, telles que :

1. Chaque trame de 802.11 contient la somme CRC-32 de la trame entière. Ainsi, pour chaque trame sortante, la somme doit être calculée et insérée dans la trame. Pour chaque trame entrante, la somme doit être recalculée afin de détecter des erreurs de transmission.

2. Pour pouvoir donner à chaque station d'un BSS la même chance d'envoyer et de recevoir des trames, les stations doivent être en *back-off* pendant une période de temps aléatoire lors de la détection du trafic sur le support de transmission. Il est très important que durant le *back-off* les périodes soient statistiquement aléatoires pour garantir l'accès à chacune des stations. Pour ces tâches, il faut générer des nombres aléatoires.
3. Pour produire les différents espaces inter-trames dont on a besoin pour les différents modes de fonctionnements (DCF et PCF) et également pour contrôler la durée de back-off, des temporisateurs temps réel sont nécessaires.
4. Toutes les en-têtes des trames sont inspectées pour connaître le type de trame, la somme de contrôle CRC, la fragmentation et l'information d'adressage. Ces tâches exigent la reconnaissance de *bitstream*. Comme les en-têtes de trames et les trames changent de taille selon leurs types, différents traitements de l'en-tête sont nécessaires.

3.3.4 PPP *(Point-to-point protocol)*

Le protocole PPP fournit une méthode standard pour le transport de trames multi-protocoles sur les liens point-à-point. PPP est composé de trois composants principaux :

- Une méthode pour l'encapsulation des datagrammes multi-protocoles, qui fournit le multiplexage des différents protocoles de la couche réseau simultanément sur la même liaison.
- Un protocole de contrôle de liaison (*LCP : Link Control Protocol*) qui établit, configure, et examine la connexion de liaison de données. Afin d'être suffisamment souple est adapté être portable à une large variété d'environnements, PPP fournit un protocole de contrôle de liaison. LCP est employé pour convenir automatiquement des options de format d'encapsulation et des limites variables sur les tailles des paquets, de détecter les erreurs de configuration et de fermer la liaison. Tout comme les autres protocoles utilisant PPP, le LCP transmet ses données en les encapsulant dans les trames PPP. Un paquet et un seul est encapsulé dans le champ « Information » de chaque trame PPP dont le champ « Protocole » a pour valeur 0xC021 (le code de LCP). On distingue trois types de paquets susceptibles d'être émis par LCP : les paquets de configuration, utilisés pour établir et configurer la liaison, les paquets de terminaison (*Terminate Packets*), qui servent à fermer la liaison et les paquets de maintenance (*Maintenance Packets*) qui permettent la gestion de la liaison.
- Une famille de protocoles de contrôle de réseau (*NCP : Network Control Protocol*) établit et configure les différents protocoles de la couche réseau. Les liaisons point-à-point tendent à aggraver beaucoup de problèmes avec la famille des protocoles de réseau. Par exemple, la tâche de gestion des adresses IP, qui est problématique dans les LAN, est particulièrement difficile pour les liaisons point-à-point avec commutation de circuit.

Afin d'établir des communications sur une liaison point-à-point, chaque bout de la liaison PPP doit d'abord envoyer des paquets LCP pour configurer et examiner la liaison de données. Après l'établissement de la liaison, le terminal peut être authentifié. Puis, PPP doit envoyer des paquets NCP pour choisir et configurer un ou plusieurs protocoles choisis de la couche du réseau. Une fois que chacun des protocoles de la couche réseau a été configuré, des trames de chaque protocole peuvent être envoyés sur la liaison. La liaison restera configurée pour la communication jusqu'à ce que des paquets LCP ou de NCP ferment explicitement la liaison, ou jusqu'à ce que certains événements externes se produisent.

Il y a des temporisateurs spéciaux employés par l'automate du protocole. Par exemple, le temporisateur de relancement est employé pour chronométrer des transmissions de paquets de *Configure-Request* et *Terminate-Request*. L'expiration du temporisateur de relancement provoque un événement d'arrêt, et la retransmission du paquet correspondant Configure-Request ou de *Terminate-Request*. Il existe aussi d'autres compteurs, par exemple Max-Terminate, qui indiquent le nombre de paquets *Terminate-Request* envoyés sans recevoir de *Terminate-Ack* avant d'assumer que le terminal ne peut répondre.

A La trame PPP

La trame PPP est une trame générique HDLC modifiée. Une trame PPP a la forme montrée dans la figure 3.6.

Drapeau 1	Adresse 1	Contrôle 1	Protocole 1-2	Les données et bourrage	Détection d'erreurs 2-4 octets	Drapeau 1

FIG. 3.6: La trame standard PPP

Les trames sont délimitées à l'aide du drapeau 01111110, et le flot de données est examiné octet par octet à la recherche de ce drapeau. La transparence utilise le caractère d'échappement (*Control Escape*), dont la valeur est 01111101. Après le calcul et l'ajout de la FCS (*Frame Cheque Sequence*), l'émetteur examine l'ensemble de la trame entre les deux drapeaux. Le caractère d'échappement, le drapeau et les caractères cochés dans l'ACCM (*Asynchronous Control Character Map*) sont remplacés par un ensemble de deux octets : le caractère d'échappement suivi du caractère original auquel on a appliqué un ou-exclusif avec 0x20. À la réception, chaque caractère d'échappement est retiré, et on applique un ou-exclusif avec 0x20 au caractère suivant.

Le champ adresse contient l'adresse de diffusion à tous les récepteurs, 11111111 (0xFF). L'intérêt est que si la liaison point-à-point est relié un réseau à toutes les machines en aval recevront les messages qui transitent par la liaison point-à-point. Dans le champ contrôle, la séquence standard 00000011 (0x03) indique que la trame est une trame d'information non numérotée (*UI : Un-numbered Information*). Il est possible d'utiliser un mode numéroté pour plus de fiabilité. Le champ protocole ne vient pas de HDLC mais est propre à PPP. Codé sur 1 ou 2 octets, il indique quel protocole transmet des données dans le champ Information. C'est là le principe de l'encapsulation, qui permet à un protocole de la couche réseau de transférer des données par l'intermédiaire du champ Information d'une trame PPP standard. Le champ d'information, de longueur variable, contient les données à transmettre, et en particulier toutes les données envoyées par les protocoles de la couche réseau. Afin de pouvoir mettre en œuvre le contrôle d'erreur, la longueur d'une trame est limitée par le MRU (*Maximum Receive Unit*), dont la valeur par défaut est de 1500 octets. Le champ Bourrage contient des bits non significatifs qui complètent toute trame de longueur utile inférieure au MRU. C'est au protocole auquel sont destinées les données de faire la différence entre les champs Information et Bourrage. Le code correcteur d'erreur place dans le champ détection d'erreurs de la redondance afin de détecter si une erreur s'est produite. Le calcul est effectué sur l'ensemble des champs Adresse, Contrôle, Protocole, Information et Bourrage, avant l'ajout des bits ou octets de transparence et des bits start ou stop. Le système utilisé est un contrôle polynomial de polynôme générateur $1+x^5+x^{12}+x^{16}$ pour un codage se fait sur 16 bits (valeur par

défaut). Il est également possible de négocier une FCS (*Frame Check Sequence*) de 32 bits ; cela réduit le taux d'erreur résiduelle, mais diminue le débit car les trames sont plus longues. Ce champ est transmis du bit de poids faible au bit de poids fort. À la réception, on supprime les bits ou octets de transparence et les bits start ou stop avant de vérifier l'absence d'erreurs [Sun99].

B Les tâches principales

Les caractéristiques de ce protocole demandent les tâches de traitement spécifiques telles que :

1. Une fonction de calcul du CRC et de vérification des calculs est nécessaire pour contrôler les erreurs dans les trames entrantes et sortantes.
2. L'implantation d'un tel protocole exige une unité qui mette constamment à jour une horloge et plusieurs systèmes de compteurs. Les compteurs peuvent être remis à zéro, forcés ensemble à une certaine valeur, incrémentés et décrémentés directement ou à certains moments.
3. La principale caractéristique de PPP et à la fois son principal avantage, est la possibilité qu'il offre, grâce au principe de l'encapsulation, de transporter simultanément les données de plusieurs protocoles de la couche réseau.
4. Une fonction de la reconnaissance de train de bits est nécessaire dans les données entrantes pour effectuer certaines actions basées sur le résultat de l'assortiment. Cette fonction est celle qui analyse les en-têtes dans les unités de données des protocoles correspondants.

3.4 Algorithmes

Dans cette section, nous présentons les algorithmes qui ont été choisis comme *benchmarks*. Pour évaluer les performances des différentes architectures de processeurs, il existe plusieurs groupes de *benchmarks* [Wolf00][Memik01]. Cependant, il n'en existe pas pour les processeurs dédiés et les protocoles que nous désirons étudier. Pour cette raison, nous avons étudié divers protocoles des couches liaison de données, MAC (*Medium Access Control*) et réseau, puis nous avons élu douze algorithmes génériques issus de ces protocoles. Dans la suite de ce chapitre, nous présentons brièvement ces algorithmes représentatifs.

3.4.1 Couche liaison de données

Cette couche découpe généralement le train de bits en trames et calcule une somme de contrôle pour chaque trame. Puisque la couche physique accepte et transmet simplement un flot de bits sans en connaître la signification ni la structure, c'est à la couche liaison de données de créer et de reconnaître les frontières des trames. Cela peut être réalisé en utilisant plusieurs méthodes. Cette couche effectue également un contrôle de flux pour régulariser le volume de données échangées entre la source et la destination [Tanenbaum03]. Beaucoup de normes et de recommendations sont disponibles pour cette couche. HDLC (*High level Data Link control*) de l'OSI a été la première vraie norme. Il est encore largement utilisée aujourd'hui. L'UIT-T a repris le mode équilibré de la procédure HDLC. Ce protocole est appelé LAP-B (*Link Access Protocol Balanced*). Cette norme a été complétée par un LAP-D associé au canal de RNIS. Il existe le protocole PPP qui est utilisé pour l'accès au réseau Internet ou pour la liaison entre deux routeurs. Nous avons étudié les

protocoles dominants de cette couche et extrait les *benchmarks* basés sur les algorithmes suivants :

- Utilisation des *flags* de début et de fin de trame et de bits de transparence pour délimiter les trames :

  Le début et la fin de la trame sont reconnaissables par la présence d'une suite d'éléments binaires, qui doit être unique. Dans la technique dite des « bits de transparence » (*bit stuffing*), chaque trame commence et finit par une séquence particulière de bits (01111110) appelée *flag*. Lorsque la couche liaison de données de la source détecte cinq « 1 » consécutifs dans les données à transmettre, elle ajoute à leur suite un bit « 0 » avant d'envoyer le train de bits sur la ligne. Quand le récepteur reçoit cinq bits « 1 » consécutifs suivis d'un « 0 », il enlève automatiquement ce dernier. Cette technique est utilisée dans les protocoles orientés bit comme le protocole LAP-B. Les *benchmarks* numéros 1 et 2 sont choisis pour ces opérations.

- Utiliser des caractères de début et de fin de trame et de caractère de transparence pour délimiter des trames :

  Dans cette méthode, les séquences particulières en forme caractère (adapté pour les protocoles orienté caractère) sont placées en début et en fin de trame. Lorsque la station destination perd la synchronisation, il lui suffit de rechercher les séquences particulières pour retrouver la délimitation des trames. Si les séquences particulières (les caractères) apparaissent dans les données, la couche liaison des données de la source répète les séquences particulières dans les données. La couche liaison de données de la destination enlève les séquences de caractère de transparence. Les *benchmarks* numéros 2 et 3 sont sélectionnés pour ces opérations.

- Décalage d'une partie de la trame de quelques bits :

  Parfois, il est nécessaire de décaler une partie de la trame, par exemple dans les méthodes de synchronisation pour retrouver les frontières des cellules du réseau ATM, comme nous avons expliqué dans le chapitre précédent. Il est clair que l'on doit décaler bit par bit. Le benchmark numéro 5 représente cette opération.

- Encapsulation de cellules ATM dans une trame SDH (*Synchronous Digital Hierarchy*) :

  Une tâche importante de la couche liaison de données dans certains protocoles est l'encapsulation de cellules ATM dans les trames. L'objectif d'ATM n'est pas de standardiser un format particulier pour la transmission des cellules. Les cellules peuvent être encapsulées dans divers types de conteneurs sur divers types de supports physiques, comme par exemple dans une trame SDH. Il existe des normes précisant comment encapsuler les cellules dans les trames propres à ces systèmes. Le benchmark numéro 6 correspond à cette opération. Nous avons utilisé la norme ATM Forum 0151 pour développer ce *benchmark*.

- Tramage et détramage dans le protocole PPP :

  PPP apparaît à la couche liaison de données et est responsable du transfert de données via des connexions sérielles. PPP encapsule un datagramme dans une trame

PPP et fait appel à la couche physique pour livrer cette trame. Cela signifie que la couche PPP par exemple, reçoit un datagramme IP, des couches supérieures, ajoute sa propre information de contrôle et construit une trame PPP. La structure d'une trame PPP est similaire à celles des trames définies par la norme HDLC. Les trames PPP sont délimitées par des flags de début et de fin de trames. Le bourrage d'octets est une opération qui garanti la transparence des données sur les chemins des communications où peuvent circuler des caractères spéciaux tels que le *flag* de trame ou la valeur d'échappement utilisée par la procédure de bourrage d'octets elle-même. Dans cette procédure, toutes les occurrences de caractères spéciaux marquées dans l'ACCM *(Asynchrone Control Character Map)* sont remplacées par une séquence qui est produite par un algorithme spécifique. La procédure inverse est réalisée au niveau du récepteur pour le détramage. Les détails sur PPP sont dans [Simpson094][Simpson194]. Les benchmarks numéros 7 et 8 sont sélectionnés pour ces opérations.

- Calcul de la somme de contrôle de l'en-tête :

  Dans certains protocoles comme IP, on trouve une zone contenant une valeur codée sur, par exemple, 16 bits qui permet de contrôler l'intégrité de l'en-tête afin de déterminer si celle-ci n'a pas été altérée pendant la transmission. La somme de contrôle d'en-tête est formée par le complément à 1 de la somme des mots de 16 bits de l'en-tête. Cette valeur devrait être modifiée après chaque traitement de l'en-tête. Le benchmark numéro 9 représente cette opération.

- Les opérations de reconnaissance du train de bits :

  Dans ces opérations, un champ du train de bits entrant est comparé avec les valeurs possibles en mémoire, pour prendre une décision. Cette decision détermine ensuite le chemin du traitement du protocole. Les opérations de recherche sont également intégrées dans ce groupe. Elles sont énormément utilisées pour la reconnaissance des trames et le filtrage des paquets.

3.4.2 Sous-couche MAC (*Medium Access Control*)

Les protocoles utilisés pour déterminer qui sera le prochain élu d'un canal de communication à accès multiples sont regroupés dans une sous-couche interne à la couche liaison de données appelée contrôle d'accès au canal ou MAC. Cette sous-couche joue un rôle très important dans les réseaux LAN, et plus particulièrement dans ceux dont le fonctionnement repose sur le principe de l'accès multiple [Tanenbaum03].

Les protocoles MAC peuvent être divisés en trois categories : assignation fixée (TDMA, FDMA...), accès aléatoire (ALOHA, CSMA/CD, CSMA/CA...) et protocoles à la demande (passage de jeton, polling...). Aujourd'hui, grâce au développement rapide des réseaux locaux sans fil (WLAN), beaucoup de nouveaux protocoles MAC dédiés à ce domaine sont développés. Nous avons choisi les algorithmes suivants parmi l'ensemble des protocoles MAC :

- Fragmentation :

  Les protocoles LAN typiques utilisent des paquets de plusieurs centaines d'octets.

Dans un environnement LAN sans fil, il existe plusieurs raisons pour lesquelles il est préférable d'employer de plus petits paquets. Premièrement, en raison du taux plus élevé d'erreurs sur les bits d'un lien radio, la probabilité de perdre paquet augmente avec la taille de celui-ci. Deuxièmement, en cas de corruption (soit en raison d'une collision, soit à cause du bruit), il faut le retransmettre. Par conséquent, il y a d'autant moins de surcharge (*overhead*) que le paquet petit. Un mécanisme simple de fragmentation-réassemblage a donc été introduit dans la couche MAC de la norme IEEE 802.11. Ce mécanisme est un algorithme simple de type « envoyer-et-attendre » : la station de transmission n'est pas autorisée à transmettre un nouveau fragment jusqu'à ce qu'elle reçoive un accusé-réception (ACK) pour le fragment précédent, ou qu'elle décide que le fragment a été retransmis un nombre trop important de fois [Tanenbaum03].

- Algorithme de *backoff* exponentiel :

  Le *backoff* est une méthode bien connue pour résoudre les conflits entre différentes stations voulant accéder au canal. Cette méthode impose à chaque station de choisir un nombre aléatoire N entre 0 et un nombre déterminé N_{max}. Une station voulant tenter d'accéder au canal devra attendre pendant un nombre de *slots* correspondant N, tout en vérifiant si une autre station n'a pas déjà accédé au canal. Le terme *backoff* exponentiel signifie que chaque fois que la station choisit un *slot* et subit une collision, le nombre maximum N_{max} augmentera exponentiellement lors du choix du nombre aléatoire N suivant.
  Le standard IEEE 802.11 définit un algorithme de *backoff* exponentiel qui doit être exécuté dans les cas suivants :
  - quand la station écoute le canal avant la première transmission d'un paquet et que le canal est occupé ;
  - après chaque retransmission ;
  - après une transmission réussie.

3.4.3 Algorithmes représentatifs

À partir des algorithmes présentés dans la section précédente, douze benchmarks ou programmes de test ont été choisis. Ils sont présentés dans le tableau 3.1.

3.5 Génération du code virtuel

La méthodologie présentée dans la figure 3.2 exige la génération d'un code indépendant de la plateforme. Un compilateur produisant du code virtuel pour un processeur virtuel a donc été développé pour les algorithmes écrits en langage C. Ce code virtuel sera reciblé plus tard sur les différentes architectures.

L'utilisation d'un compilateur classique aurait été possible. Cependant le code produit pour un processeur réel quelconque est fortement contraint par les caractéristiques de son architecture. Il est particulièrement difficile d'éliminer ces contraintes lorsque l'on rent convertir le code vers une architecture très différente. Employer un modèle libéré de cette contrainte (une architecture virtuelle de processeur) permet d'obtenir un code pouvant prendre diverses architectures pour cibles.

En effet, un processeur virtuel idéal ne devrait souffrir d'aucune limitation sur le jeu d'instructions ou sur le matériel (ressources disponibles en nombre infini). Cependant, il serait tout à fait inconcevable de réaliser un tel modèle trop complexe. En pratique, une

benchmark	algorithme
1	détramage avec bit de transparence
2	tramage avec bit de transparence
3	tramage avec caractère de transparence
4	détramage avec caractère de transparence
5	décalage d'une partie de la trame de quelques bits
6	encapsulation des cellules ATM
7	tramage PPP
8	détramage PPP
9	calcul la somme de contrôle
10	assortiment du train de bits et recherche dans un tableau
11	fragmentation
12	algorithme back-off (LAN sans fil)

TAB. 3.1: Les benchmarks et les algorithmes

approximation ayant les caractéristiques suivantes a été considérée :

- le nombre de registres a été limité à 128. Ce nombre devrait être suffisant pour limiter les transferts de données entre le processeur et la mémoire, qui sont la plupart du temps inefficaces. Une largeur de données fixe est considérée et cette largeur détermine la dimension des registres dans le processeur virtuel (et également dans toutes les architectures réelles) ;

- le format d'instruction est composé de quatre champs :
  INS P1, P2, P3
  P1, *P2* et *P3* dénotent des paramètres facultatifs et représentent les opérandes : des opérandes de mémoire ou des registres, voire des données (seulement comme source). L'opération liée au champ INS peut appartenir à l'une des catégories suivantes : operation arithmétique et logique, transfert de données ou contrôle ;

- le temps d'accès mémoire est supposé être assez court de sorte qu'aucune mémoire cache ne soit exigée. De même, la mémoire est assez grande pour contenir tout le programme et les données.

Dans ce processeur virtuel, nous avons défini le jeu d'instruction comme suit :

- rdm : *read memory*
- wrm : *write memory*
- sub : *subtraction immediate*
- add : *addition*
- and : *logic and*
- xor : *logic exclusive or*
- or : *logic or*
- shf : *shift operation*
- div : *division*
- mul : *multiplication*
- cmp : *compare*
- bun : *brunch(conditional)*
- jmp : *jump*
- call : *call procedure*

L'autre raison accessoire qui nous a mené à développer notre propre compilateur est que pour calculer la durée des séquences de nos algorithmes, nous avons besoin de manipuler certains éléments internes du compilateur. Hors cette opération n'est pas aisée dans un compilateur classique du fait de sa complexité. C'est pourquoi, nous avons développé un compilateur simple et minimal pour le langage C ANSI.

Comme indiqué préalablement, le temps requis pour exécuter un algorithme sur une architecture réelle peut être calculé à partir d'une chaîne de Markov si la durée de chaque séquence dans l'algorithme est connue. À cette fin, un reciblage du code virtuel sur l'architecture réelle doit être fait pour chaque séquence de l'algorithme.

Le reciblage du code injecte les contraintes du jeu d'instructions dues à la structure architecturale. Par exemple, un nombre restreint de registres limite la quantité de données qui peut être conservée dans le processeur et par conséquent exige plus d'échanges de données entre les registres et la mémoire. Un nombre restreint de modes d'adressage peut exiger plus d'instructions pour réaliser la fonction d'une instruction virtuelle. Il est à noter que la génération effective du code réel n'est pas exigée car, seul le celui du nombre de cycles d'instructions est nécessaire.

3.6 Modélisation des algorithmes par chaîne de Markov

La première étape de cette méthode est d'obtenir des modèles équivalents des algorithmes, pouvant être facilement manipulés. Des graphes orientés sont généralement employés à cette fin. L'exécution des algorithmes peut être émulée avec des chaînes de Markov homogènes à temps discret [Ross89], qui permettent de calculer les probabilités d'exécution des séquences dans un programme. Les chaînes de Markov homogènes à temps discret ont un nombre fini d'états (nœuds du graphe) s_i. En effet, la probabilité de transition $p_{t,t+1}(s_i,s_j)$ de l'état s_i à l'instant t, à l'état s_j à l'instant $t+1$ dépend seulement des états s_i et s_j, et non de l'instant t.

Une chaîne de Markov X_n (une suite de variables aléatoires) est un processus à temps discret. Elle est définie par son vecteur de probabilités d'états $P_{et}[n]$ dont les éléments sont :

$$P_{et,i}[n] = pr\{x_n = a_i\} \qquad i = 1,2,... \tag{3.1}$$

où x_n et a_i indiquent une variable aléatoire à l'instant n et un état de la chaîne respectivement. et des probabilités de transition :

$$P_{ij}[n_1,n_2] = pr\{x_{n2} = a_j | x_{n1} = a_i\} \tag{3.2}$$

qui constituent la matrice de transition. Nous avons :

$$\begin{aligned} \sum_j P_{ij}[n_1,n_2] &= 1 \\ \sum_i P_{et,i}[k]P_{ij}[k,n] &= P_{et,j}[n] \end{aligned} \tag{3.3}$$

Si le processus X_n est homogène, la probabilité de la transition dans l'équation (3.2) dépend juste de $m = n_2 - n_1$. Donc :

$$P_{ij}[m] = pr\{x_{n+m} = a_j | x_n = a_i\} \tag{3.4}$$

Pour une chaîne de Markov le nombre d'états fini, l'équation de Chapman-Kalmogoroff peut être écrite sous la forme d'un vecteur [Ross89] :

$$P[n+k] = P[n]P[k] \tag{3.5}$$

Donc :

$$\begin{aligned} P[n] &= P^n \\ P &= P[1] \end{aligned} \tag{3.6}$$

Si nous écrivons l'équation (3.3) sous forme d'un vecteur, en utilisant l'équation (3.5), nous avons [Ross89] :

$$P_{et}[n] = ... = P_{et}[n-k]P^k = ... = P_{et}[0]P^n \tag{3.7}$$

Si $P_{et}[n] = P_{et}$, le processus homogène X_n est aussi stationnaire et son vecteur d'état est la solution du système :

$$\begin{aligned} P_{et}P &= P_{et} \\ \sum_i P_{et,i} &= 1 \end{aligned} \tag{3.8}$$

Dans notre utilisation des chaînes de Markov, un algorithme est composé de plusieurs séquences. Une séquence consiste en un ensemble maximal d'instructions consécutives sans aucune instruction de branchement. Une séquence est représentée par un nœud du graphe dans la chaîne de Markov. Considérons l'exemple suivant (représenté dans la figure 3.7) :

```
if (condition) then
  {statements_in_if}
statements_after_if

seq1-> condition evaluation
seq2-> statements_in_if
seq3-> statements_after_if
```

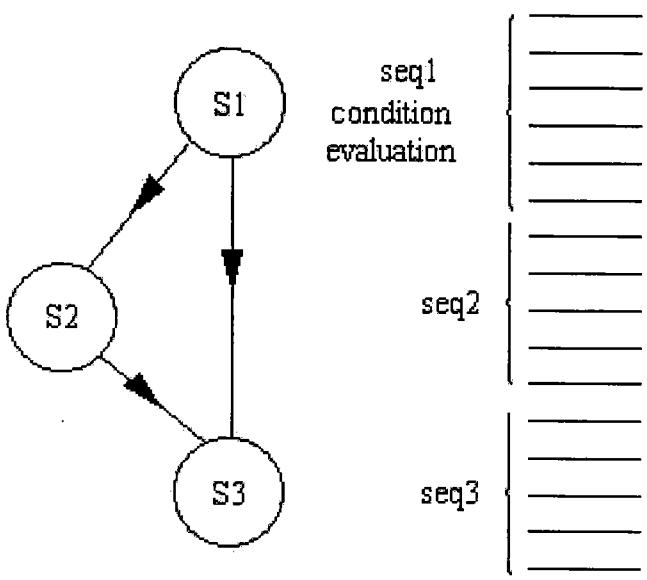


FIG. 3.7: exemple de l'équivalence en algorithme/graph

À la fin de seq1, deux séquences différentes peuvent être suivies. En fait, les probabilités du branchement dépendent des données évaluées dans la condition. Des probabilités moyennes peuvent être obtenues à partir de l'évaluation de l'exécution de l'algorithme sur un processeur conventionnel quelconque, en utilisant des données représentatives.

Dans une chaîne de Markov, l'état j est accessible s'il existe une probabilité P_{ij} différente de zéro. Un état est absorbant s'il est accessible et fermé (il ne peut le quitter). Une

chaîne de Markov est absorbante si elle contient au minimum un état absorbant. Dans notre utilisation des chaînes de Markov, on veut calculer la durée de l'algorithme comme le passage de l'état début à l'état fin de l'algorithme : on a donc considéré des chaîne absorbantes avec un état fin absorbant.

On définit le vecteur D par ses éléments représentant le temps (le nombre de cycles) requis pour le passage de l'état i à l'état absorbant, et le vecteur T par ses éléments representant le temps du séjour dans chaque état. On peut calculer le temps de passage de l'état i à l'état n par l'équation suivante :

$$D_i = T_i + \sum_{j=1}^{n} P_{ij} D_j \qquad i = 1,2,...n \tag{3.9}$$

ou, sous forme matricielle :

$$D = T + PD \tag{3.10}$$

On peut calculer le vecteur D dont le premier élément (D_1) est la durée de l'algorithme. En d'autres termes, le temps requis pour exécuter l'algorithme peut être calculé à partir de la chaîne de Markov si la durée de chaque nœud (les éléments du vecteur T), c'est-à-dire chaque séquence dans l'algorithme, est connue. La durée d'une séquence dépend des instructions qui la composent et donc aussi de l'architecture réelle sur laquelle s'exécute la séquence.

D'autre part, on peut calculer le nombre moyen de cycles passés dans chaque état avant d'arriver à l'état absorbant supplémentaire par l'équation suivante :

$$N = (I - Q)^{-1} \tag{3.11}$$

Q est obtenu à partir de P en éliminant la ligne et la colonne correspondant à l'état absorbant et N est une matrice où N_{ij} est le nombre de passages par l'état j en partant de l'état i et avant d'arriver à l'état absorbant. La première ligne de cette matrice représente le nombre de cycles passés dans les états de la chaîne en partant de l'état début.

3.7 Architectures à évaluer

Dans cette section, nous introduisons différents modèles architecturaux de processeurs généralistes. Les performance de ces architectures, pour l'exécution des algorithmes représentatifs, ont été évaluées sur notre banc de simulation.

3.7.1 CISC

Les caractéristiques principales des architectures CISC sont un grand nombre d'instructions et la complexité élevée de certaines d'entre elles. Cette architecture utilise divers formats d'instructions ainsi que plusieurs modes d'adressage. De plus, peu de registres généraux étant disponibles, les opérandes de données doivent souvent être lues et stockées directement en mémoire [Patterson96].

Les instructions à cycles multiples permettent à la période d'exécution d'une instruction d'être adaptée à sa complexité [Patterson96]. Le contrôle microprogrammé est généralement utilisé dans les CISC traditionnels, mais le contrôle câblé est parfois également

employé.

Pour vérifier la performance de ce type d'architecture, nous avons utilisé un modèle basé sur la famille des Pentium. Les processeurs de la famille Pentium (I, II, III, MMX, Céléron, et autres puces x86), sont construits sur le principe du jeu d'instructions complexe (CISC). Les modèles récents possèdent une architecture CISC-superscalaire mais les modèles antérieurs sont plus proches de l'architecture CISC standard.

Pour cette raison, nous avons utilisé la liste du nombre de cycles d'horloge nécessaires pour exécuter les instructions de la première version du Pentium, extraite des documentations d'Intel. Nous avons considéré les instructions plus fréquents dans nos algorithmes pour calculer le nombre de cycles d'horloge des séquences ou des algorithmes. La liste suivante représente les instructions considérées :

- `mov`
- `lea` (load effective addresse)
- `push`
- `sub`
- `add`
- `and`
- `xor`
- `or`
- `shift`
- `cmp`
- `jmp`
- `call`
- `jmp` (conditional : `bun`)
- `pop`
- `inc/dec`
- `mul/imul`
- `div/idiv`

Pour chaque algorithme le code assembleur a été généré par le compilateur gcc et le nombre d'instructions ci-dessus a été compté dans ce code.

3.7.2 RISC

Le but fondamental de la conception architecturale RISC est de maximiser la vitesse d'une architecture en exécutant la plupart des fonctions de manière logicielle, à l'exception de celles dont l'implantation matérielle apporte un gain net à la performance globale. Les instructions complexes et les différents modes d'adressage des CISC justifient l'utilisation d'un microcode et les instructions mais cycles multiples compliquent la conception des processeurs. Par contre, la conception RISC permet d'éliminer le microcode parce qu'elle conduit à un jeu d'instructions réduit et à des modes d'adressage simples. La conception du jeu d'instructions est faite de sorte à ce que la plupart des instructions soit exécutée en un seul cycle à chaque étage du pipeline [Patterson96].

Le grand nombre de registres, le jeu d'instructions réduit, et le nombre limité de modes d'adressage des RISC rendent plus facile l'optimisation du code par le compilateur et permettent d'avoir un pipeline d'efficacité maximale.

Pour vérifier la performance de ce type d'architecture, nous avons utilisé un modèle basé sur la famille MIPS. Ce modèle possède 32 registres généraux de 32 bits appelés $R0, R1, ...R31$. De plus, il existe un ensemble de registres à virgule flottante. Seize registres de 32 bits sont utilisés pour sauvegarder provisoirement les données, car cette architecture est de type registre-registre et ne peut accéder à la mémoire que par les instructions de chargement et de rangement (*load-store*). Le reste des registres est utilisé pour les autres fonctions comme enregistrement des arguments et des résultat des procédures ou la gestion du pointeur de pile. Les opérations sont effectuées sur des entiers 32 bits et des flottants 32 bits. Les octets et les demi-mots sont chargés dans des registres avec une extension à 32 bits. Les seuls modes d'adressage des données sont les modes immédiat et déplacement. La mémoire est adressable à l'aide d'une adresse de 32 bits.

La figure 3.8 montre les formats des instructions. Il en existe trois groupes : 1, 2 et 3. Dans le groupe 1, on trouve cinq instructions : `rdm` (*read memory*), `wrm` (*write memory*), `mov`, et deux instructions de branchement conditionnel (`bez` et `bnz`). Les instructions ALU (*Arithmetic and Logic Unit* : unité arithmétique et logique) appartiennent au groupe 2. Toutes les instructions ALU sont des instructions de type registre-registre. Ces opérations comprennent les opérations arithmétiques et logiques simples ainsi que des instructions de comparaison entre deux registres. Si le test est vrai ces instructions placent un 1 dans le registre destination, sinon, elles placent la valeur 0. Le groupe 3 comprend les instructions `jmp`, `ret` et `call`. La liste suivante présente les instructions :

- `rdm` : *read memory*
- `wrm` : *write memory*
- `mov` : *move immediate*
- `bez` : *brunch if zero*
- `bnz` : *brunch if not zero*
- operations arithmétiques et logiques
- `slt` : *set if less than*
- `sgt` : *set if great than*
- `seq` : *set if equal*
- `sne` : *set if not equal*
- `sle` : *set if less or equal*
- `sge` : *set if great or equal*
- `jmp` : *jump*
- `call` : *call procedure*
- `ret` : *return procedure*

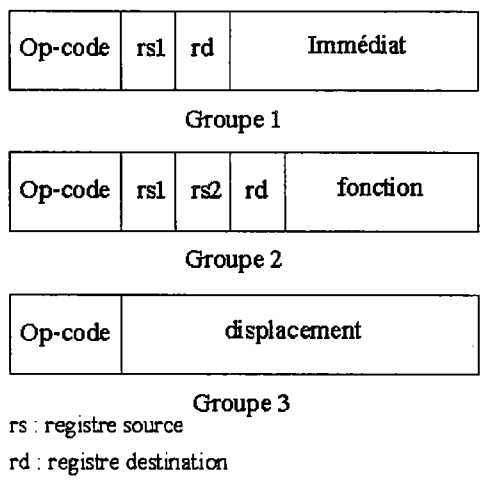


FIG. 3.8: Les formats d'instructions du modèle RISC

3.7.3 Superscalaire

Dans les architectures superscalaires, les instructions sont décomposées intérieurement en micro-opérations qui correspondent directement aux fonctions matérielles. On observe la séquence normale des instructions par une fenêtre d'instruction [Gloria99].

À l'intérieur de cette fenêtre, l'ordonnancement est fait dynamiquement par le matériel. La taille de la fenêtre a un impact direct sur la complexité de l'ordonnancement et sur le parallélisme pouvant être réalisé. L'unité de contrôle d'un processeur superscalaire est souvent très compliquée, mais sa structure peut être rendue compatible au niveau binaire avec celle d'un processeur scalaire [Lenell94]. Ainsi, un compilateur standard peut être employé. Dans un processeur superscalaire typique, le matériel peut lancer de 1 à 8 instructions par cycle d'horloge. Normalement, ces instructions doivent être indépendantes et satisfaire certaines contraintes, telles qu'un accès mémoire au plus réalisé par cycle.

Le modèle que nous avons utilisé dans notre évaluation des architectures superscalaires est basé sur une méthode utilisée dans certain processeurs tels le Power PC 620 et le MIPS R10000. Cette méthode combine l'exécution spéculative avec l'ordonnancement dynamique basé sur l'algorithme de Tomasulo [Patterson96]. L'idée de base de l'implantation de l'exécution spéculative est de permettre aux instructions de s'exécuter de manière non ordonnée mais de les forcer à se terminer dans l'ordre. En effet, les processus de fin d'exécution et de la garantie de terminaison de l'instruction sont séparés, puisque les instructions peuvent terminer leur exécution bien avant d'être validées [Patterson96]. Nous avons simulé l'exécution des instructions en 5 étapes, dans un pipeline à cinq étages uniquement pour les opérations entières (cf. figure 3.9).

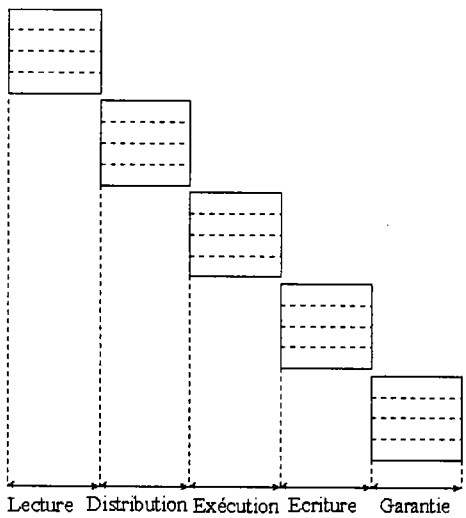


FIG. 3.9: Pipeline du modèle superscalaire

1. **Lecture :**
   Dans cette étape, quatre instructions sont lues simultanément en un cycle d'horloge de la mémoire. Celle-ci est supposée ne pas constituer une frein pour les accès au programme.
2. **Distribution :**
   Dans cette étape, les instructions sont lancées simultanément en un cycle d'horloge dans les stations de réservation et le tampon de réordonnancement. Les stations de réservation mémorisent les opérandes des instructions en attente de démarrage via l'unité de distribution. L'idée de base est qu'une station de réservation lit et mémorise une opérande aussitôt qu'elle est disponible, éliminant le besoin d'obtenir

l'opérande depuis un registre [Patterson96]. À la distribution d'une instruction, les registres des opérandes en attente sont renommés avec le nom des stations de réservation. C'est le processus de renommage de registres. En utilisant des stations de réservation, on élimine les aléas EAE (Écriture Après Écriture) et EAL (Écriture Après Lecture). Le tampon de réordonnancement contient les instructions et leurs résultats entre le moment où les opérations associées aux instructions se terminent et le moment où les instructions sont validées. Nous avons consideré quatre unités d'exécution et une station de réservation pour chacune d'elles. Chaque station de réservation peut mémoriser quatre instructions.

3. **Exécution :**
   Cette étape recherche les aléas LAE (Lecture Après Écriture). Lorsque les opérandes sont disponibles, les opérations correspondantes sont exécutées par les unités d'exécution. Si une ou plusieurs opérandes ne sont pas disponibles, on attend que ceux-ci soient calculés. En considérant, pour le types d'algorithmes que nous avons sélectionné, que la majorité des instructions sont de type chargement-rangement, nous avons décidé d'intégrer à l'unité d'exécution deux unités LD-ST (*Load/Store*), une ALU et une CTU (*Control Transfer Unit*).
4. **Écriture des résultats :**
   Lorsque les résultats sont disponibles, ils sont écrits dans le tampon de réordonnancement ainsi que dans toutes les stations de réservation attendant ces résultats.
5. **Garantie (*Commit*) :**
   Dans cette étape, quand le résultat de l'instruction à la tête du tampon est disponible, le résultat est rangé dans le registre ou en mémoire (si l'opération est un rangement), et les instructions dans le tampon se décalent vers la tête du tampon. En effet, le tampon de réordonnancement fonctionne comme une file circulaire.

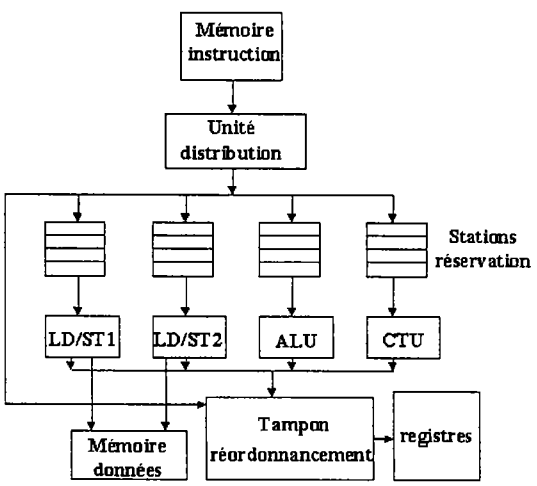


FIG. 3.10: Le modèle superscalaire

3.7.4 VLIW

Le VLIW est une méthode efficace pour augmenter la performance au delà des possibilités des architectures standards RISC. On profite du parallélisme spatial en utilisant des

unités fonctionnelles multiples pour exécuter plusieurs opérations simultanément. Une instruction VLIW est se composé d'instructions constitutives plus simples, groupées ensemble par le compilateur, qui doit avoir la pleine connaissance de la micro-architecture et des contraintes des ressources pour regrouper des instructions indépendantes, exécutables en parallèle. Le compilateur emploie des techniques d'optimisation telles que le pipeline logiciel, le déroulage de boucle et la prédiction de branchement [Patterson96]. Du point de vue de l'amélioration d'exécution, les structures superscalaire et VLIW se ressemblent, mais dans un VLIW la complexité de contrôle est transférée du matériel au logiciel. En effet, l'unité de contrôle du processeur VLIW n'a pas à détecter de dépendances entre instructions au moment de l'exécution [Abnous95].

Dans notre modèle, une instruction VLIW peut inclure deux opérations LD-ST, une opération ALU et une opération CTU. Pour maintenir les unités fonctionnelles occupées, il doit exister suffisamment de travail dans une séquence de code linéaire pour remplir tous les champs d'opération disponibles. Dans notre modèle, un modèle simulé de VLIW recherche les opérations parallèles en déroulant les boucles et en ordonnançant le code à travers les blocs d'une fenêtre d' instructions. Il est évident que des instructions NOP sont parfois insérées dans une instruction VLIW à cause des limites du parallélisme d'instructions.

3.8 Analyse des résultats

Beaucoup d'information sont fournies par le reciblage des algorithmes sur différentes architectures, ainsi que par l'évaluation par des chaînes de Markov. Pour chaque architecture, la durée de la séquence et la contribution globale de la séquence dans le temps d'exécution global est disponible. De plus, la dépendance fonctionnelle entre les séquences peut être facilement obtenue.

L'interprétation d'une telle quantité d'information exige des méthodes adéquates telles que l'analyse de données multidimensionnelle. À cette fin, nous avons utilisé le logiciel SPSS, qui est un paquetage statistique. Le but global est d'évaluer pour l'adéquation des différentes implantations architecturales avec les contraintes temporelles des algorithmes.

Pour vérifier la performance de diverses architectures, nous avons procédé en étapes. Dans la première, nous comparons les architectures (CISC-RISC) et (superscalaire-VLIW) séparément. Dans la deuxième, nous évaluons la possibilité d'utiliser une architecture multiprocesseurs constituée de différentes architectures RISC, CISC, superscalaire et VLIW.

Notre critère pour l'évaluation est le rapport entre le nombre de cycles d'horloge pour exécuter les benchmarks sur les architectures désirées et le nombre d'instructions virtuelles requises pour le processeur virtuel. Dans notre approche, ni l'organisation de la mémoire, ni les transfert entre celle-ci et le processeur n'ont été considérés, car nécessitant une vue plus détaillé du compilateur, ce qui complique excessivement la démarche sans bénéfice majeur.

La Figure 3.11 montre le rapport entre le nombre de cycles d'horloge et le nombre d'instructions virtuelles pour les deux architectures CISC et RISC. Les résultats des simulations montrent que le nombre de cycles d'horloge requis pour exécuter les benchmarks sur l'architecture CISC est en moyenne deux fois plus élevé que sur un RISC. Le cycle d'instruction est généralement plus long dans les processeurs basés sur le modèle RISC, mais la considérable différence estimée de performance entre RISC et CISC indique une nette préférence du RISC. De plus, la philosophie CISC consiste à compacter

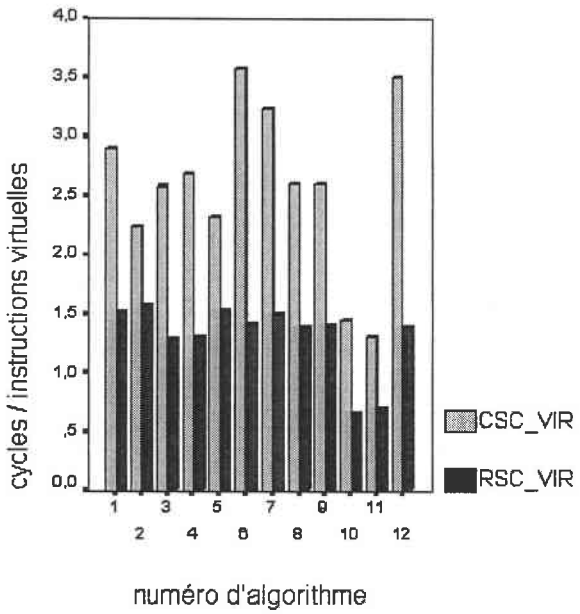


FIG. 3.11: Rapport du nombre de cycles dans les architectures CISC et RISC par le nombre d'instructions virtuelles

les opérations simples dans des instructions complexes, alors que le traitement de protocoles n'exige pas d'instruction complexe. Une étude semblable [Elkateeb00] montre qu'une architecture RISC est bien performante comme pour le processeur de traitement de protocoles dans un réseau ATM.

Nous avons également simulé l'exécution des benchmarks sur des architectures ILP (*Instruction Level Parallelism*), c'est-à-dire superscalaire et VLIW avec quatre unités fonctionnelles. Tout d'abord, une étude quantitative a été réalisée sur les instructions virtuelles dans les benchmarks. Comme les résultats l'illustrent dans la figure 3.12, le nombre d'instructions de référence à la mémoire (LD-ST) est deux fois plus grand que le nombre des instructions d'ALU et des instructions de branchement (Jump et Jump conditionnel). En ce basant sur ces résultats, deux unités LD-ST, une unité ALU et une unité CTU (*Control Transfer Unit*) ont été employés comme étant le choix le plus approprié.

Les résultats de simulation de l'architecture superscalaire sont montrés dans la figure 3.13. Comme la figure l'illustre, avec 4 unités fonctionnelles dans un modèle superscalaire, le facteur d'accélération moyen n'est pas que de 2 (nous avons obtenu des résultats semblables pour VLIW comme cela est montrée dans la Figure 3.14).

Cette faible accélération n'indique pas une bonne performance, particulièrement en considération de la complexité du réordonnancement (*scheduling*) dans l'architecture superscalaire. Ces résultats indiquent qu'il y a beaucoup de dépendances d'instructions dans les benchmarks. Ce problème est dû à beaucoup de tâches de reconnaissance du train de bits et aux instructions de référence à la mémoire dans nos applications, cela augmente les dépendances d'instructions entre de petits blocs de base (les séquences) dans le code. Dans ces tâches, un champ dans une en-tête entrante est comparé aux valeurs possibles en mémoire et une décision est prise selon la valeur. En effet, dans les algorithmes de traitement des protocoles, les instructions exécutées sur un paquet dépendent du contenu du paquet, ce qui ajoute des dépendances entre les instructions.

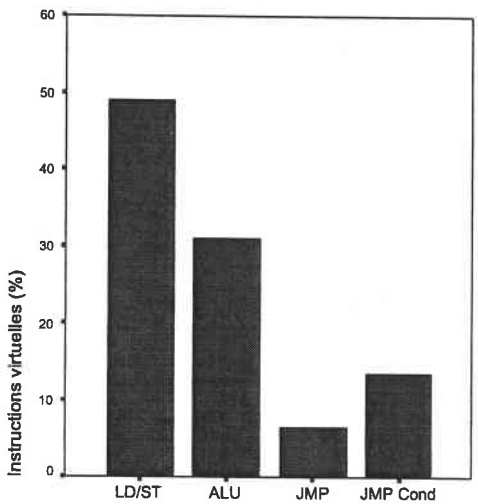


FIG. 3.12: Comptage des instructions virtuelles

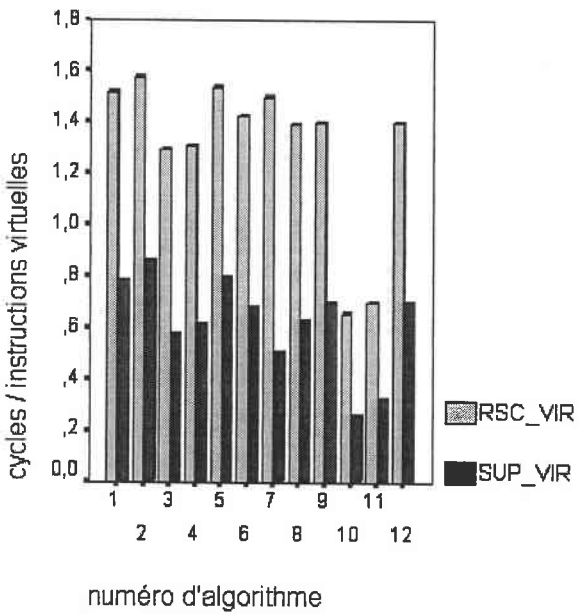


FIG. 3.13: Résultats des mesures sur l'architecture supercalaire

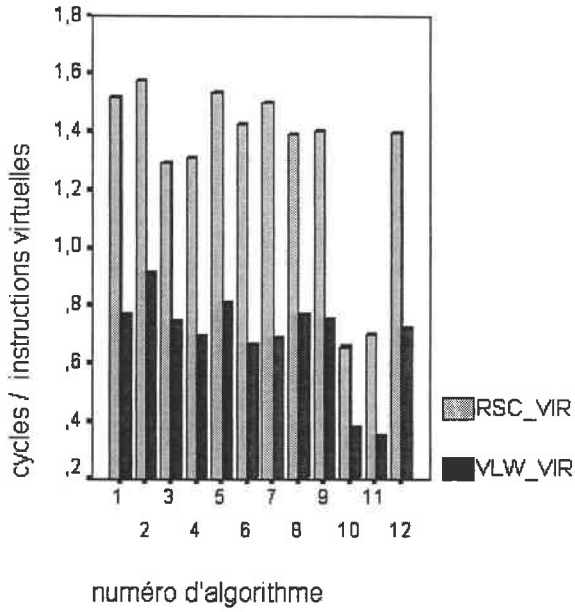


FIG. 3.14: Résultats des mesures sur l'architecture VLIW

Dans les architectures ILP, la taille de la fenêtre de réordonnancement a un impact direct sur la complexité du réordonnancement et sur le parallélisme qui peut être atteint. Cet impact a été également étudié. Les résultats indiquent que, d'une part, il n'existe pas suffisamment de comportement communs dans les benchmarks, et que, d'autre part, compte tenu de l'augmentation de la complexité due au réordonnancement, l'amélioration de performance est insuffisante en raison de la petite dimension des blocs de base (séquences). Par exemple, pour les benchmark 5 et 10, les résultats ont été montrés dans les figures 3.15 et 3.16 pour les architectures superscalaire et VLIW, respectivement.

En considérant les résultats obtenus, l'architecture RISC présente une bonne performance par à rapport à l'architecture CISC. D'autre part, les architectures ILP , ne sont pas des bonnes candidates, à cause du nombre élevé de dépendances entre les instructions dans nos applications. Par conséquent, globalement l'architecture RISC s'avère être un candidat adéquat pour effectuer les traitements non-orientés DSP.

À partir de ce point, nous voulons évaluer l'intérêt d'exécution des algorithmes sur une architecture de type multiprocesseurs. Dans cette approche, les applications sont divisées en unités plus petites (les grains) et ces grains sont exécutés sur différentes unités de traitement qui peuvent être basées sur des architectures différentes. Par exemple, une architecture scalaire (RISC-CISC) ou superscalaire. À cette fin, on cherche une bonne affinité entre les grains et les architectures. En d'autres termes, on doit vérifier la capacité du regrouper des grains par notre méthodologie, en fonction des architectures. Par exemple, il est évident qu'il faut utiliser des architectures parallèles (superscalaires ou VLIW) pour exécuter des grains à haute capacité de paralléliser. Pour cela, nous avons considéré deux niveaux de granularité pour diviser les applications en grains : les algorithmes et les séquences. Dans cette étude, nous avons choisi les benchmarks représentatif des algorithmes de nos applications. Une séquences consistent en un ensemble maximal d'instructions consécutives avec une instruction de branchement à la fin de la séquence, ce qu'on appelle bloc de base du code et qui constitue le plus petit niveau de granularité.

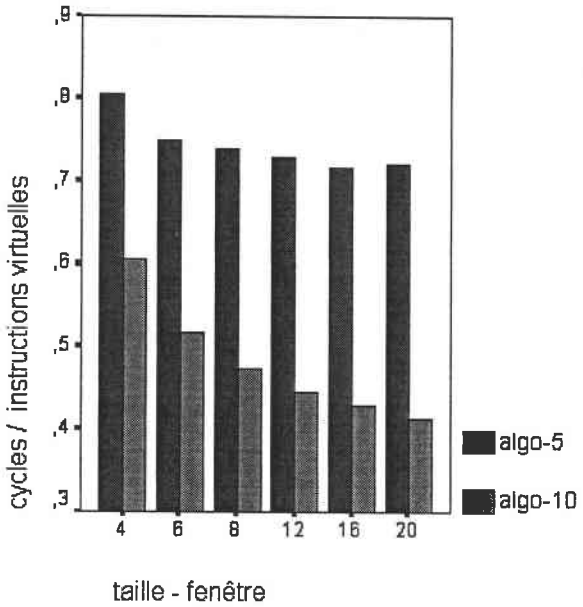


FIG. 3.15: Impact de la taille de la fenêtre de réordonnancement dans l'architecture superscalaire

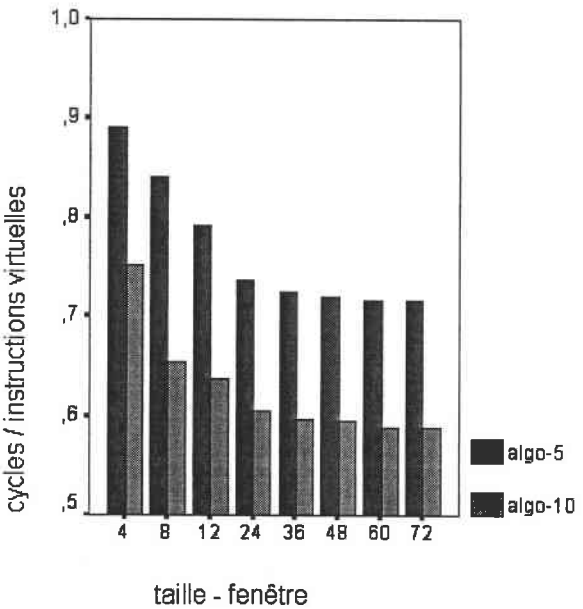


FIG. 3.16: L'impact de la taille de la fenêtre de réordonnancement dans l'architecture VLIW

Pour vérifier l'affinité entre les grains et les architectures, nous avons utilisé une méthodologie basée sur l'analyse multi-dimensionnelle de données. Pour cela, nous avons réalisé une base de données qui regroupe les individus selon les variables représentatives (critères) concernant la performance des différentes architectures. Dans notre base de données, il y a quatre variables représentatives qui mesurent pour chacune des quatre architectures étudiées, le rapport entre le nombre de cycle d'horloge requis pour exécuter un grains et le nombre d'instructions virtuelles qu'il contient. Les variables explicatives sont constituées des diverses mesures effectuées sur les instructions virtuelles. Nous décrivons ces variables ci-dessous :

- ALU-MSR : pourcentage d'instructions arithmétiques et logiques.
- RD-MSR : pourcentage d'instructions de lecture de la mémoire.
- WR-MSR : pourcentage d'instructions d'écriture à la mémoire.
- ALU-DIST : distance moyenne entre les instructions ALU.
- RDRW-MSR : distance moyenne entre les instructions de la mémoire.
- DEPCY : nombre de dépendances de données entre les instructions virtuelles par à rapport au nombre d'instructions virtuelles.
- DCT-ADR : pourcentage d'instructions qui utilisent le mode d'adressage direct.
- INDCT-ADR : pourcentage d'instructions qui utilisent le mode d'adressage indirect.
- N-REG : nombre maximum de registres utilisés.

Dans cette base de données, les individus sont les séquences ou les algorithmes. Pour classer les individus, nous avons utilisé la méthode AFD (analyse factorielle discriminante). L'analyse factorielle discriminante permet de mettre en évidence les liaisons existantes entre une observation (individu) et un ensemble des caractères explicatifs, ainsi que la possibilité d'établir une règle décisionnelle.

En premier lieu, une étude a été effectuée en utilisant comme niveau de granularité les algorithmes. Tout d'abord, nous avons classer les algorithmes en trois catégories basées sur le critère CISC/RISC, c'est-à-dire, le rapport entre le nombre de cycles requis pour les exécuter sur une architecture CISC par rapport à cette valeur dans une architecture RISC. Dans cette étape, les mesures d'exécutions de différents algorithmes sur plusieurs jeux de données représentatifs sont utilisées comme individus. Les résultats de l'analyse AFD sont montrés sur la figure 3.17.

Comme la figure 3.17 le montre, les algorithmes sont bien discriminés par les deux fonctions 1 et 2. La fonction 1 (qui comprend 95.9% de variance expliquée [1]) a bien discriminée les groupes 1 et 2 du groupe 3. Selon les coefficients discriminant les variables et la matrice de corrélation entre les fonctions et variables, ALU-MSR, WR-MSR et RD-MSR ont la plus grande capacité de discrimination. Les résultats montrent qu'un critère discriminant peut être trouvé concernant les instructions de référence à la mémoire et celles utilisant ALU, pour classer les algorithmes en fonction de leur affinité avec les architectures RISC et CISC.

Le deuxième critère utilisé est le rapport RISC/SUP. Ce rapport représente la capacité des architectures superscalaires ou des architectures ILP (*Instruction Level Parallelism*) matérielles à exécuter les algorithmes. Comme la figure 3.18 le montre, les algorithmes sont bien discriminés par les deux fonctions et surtout la fonction 1. Cette fonction est fortement déterminée par la variable DEPCY. En effet, cette variable, indiquant la quantité de dépendances de données entre les instructions virtuelles, est un bon critère pour trouver

[1] une quantité statistique qui représente le pourcentage de l'information initialement contenu dans les variables explicatives qui est conservé dans la fonction.

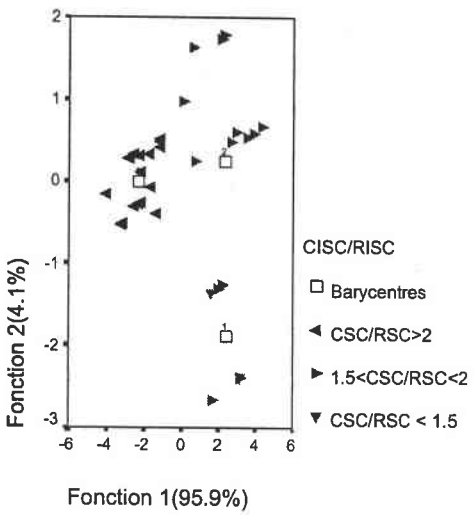


FIG. 3.17: Résultats AFD avec le critère CISC/RISC au niveau de l'algorithme

l'affinité entre les algorithmes et les architectures ILP.

Nous avons aussi utilisé le critère VLIW/SUP. Le résultat de l'AFD (cf. 3.19 et 3.20) montre que la fonction 1 a bien discriminé les deux groupes mais, selon la matrice de corrélations, il n'existe pas une bonne corrélation entre la fonction et les variables. Cela signifie que les mesures effectuées sur le code virtuel ne peuvent pas discriminer de manière assez claire les affinités entre les algorithmes et les architectures VLIW et superscalaire. Ce résultat est logique parce que bien qu'utilisant des approches différentes (l'une logicielle, l'autre matérielle), les architectures VLIW et superscalaire suivent la même philosophie architecturale.

La deuxième analyse a été effectuée en utilisant comme niveau de la granularité les séquences. Plus de 450 séquences de différents algorithmes sont utilisées comme individus. Le critère CISC/RISC est le premier qui a été utilisé. Les résultats sont présentés dans la figure 3.21. Bien qu'assez bonne, la capacité de discrimination des fonctions est dégradée par rapport aux résultats du niveau algorithmes. Cette dégradation est due à un grand écart-type pour les variables avec la plus grande corrélation à deux fonctions discriminantes. Dans ce cas, ALU-MSR , WR-MSR,RD-MSR et RDWR-DIST ont la plus grande capacité de discrimination selon la matrice de corrélation entre les variables et les fonctions.

Nous avons utilisé le rapport RISC/SUP comme deuxième critère. Comme il a été mentionné, ce rapport présente la capacité d'exécution de séquences en parallèle au niveau instruction ou ILP. La figure 3.21 montre les résultats.

Selon ces résultats, les fonctions 1 et 2 peuvent assez bien discriminer les séquences. Les corrélations maximales entre ces fonctions et les variables correspondent aux variables : RD-MSR, WR-MSR et DEPCY. Comme la figure le montre, la fonction 1 avec 86.3% de variance expliquée a bien discriminé les séquences de RISC/SUP<1.5 et les séquences de RISC/SUP>1.5.

VLIW/SUP est l'autre critère examiné. En effet, ce critère permet une comparaison entre le modèle ILP par compilateur (VLIW) et ILP par matériel (superscalaire). Les résultats sont illustrés dans la figure 3.23.

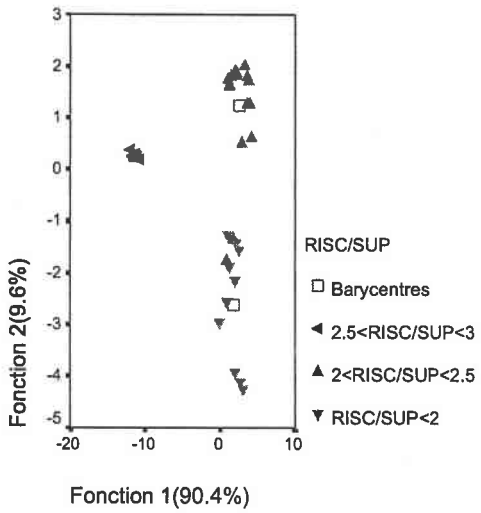


FIG. 3.18: Résultats de l'AFD avec le critère RISC/SUP au niveau algorithme

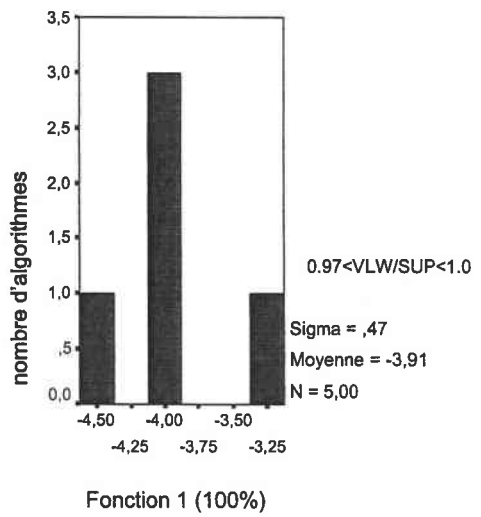


FIG. 3.19: Résultats de l'AFD avec le critère VLIW/SUP au niveau algorithme

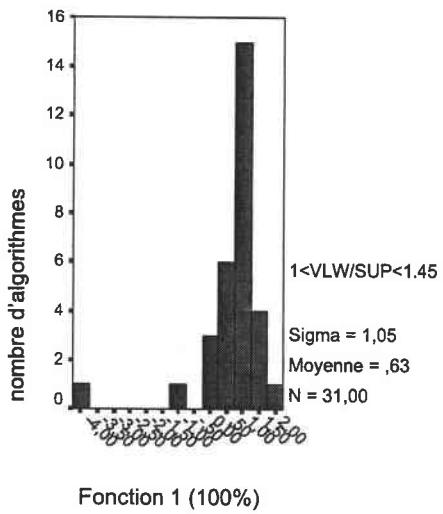


FIG. 3.20: Résultat de l'AFD avec le critère VLIW/SUP au niveau algorithme

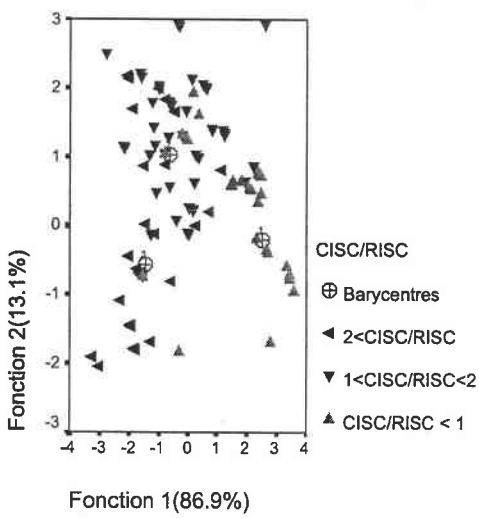


FIG. 3.21: Résultat de l'AFD avec le critère CISC/RISC au niveau séquence

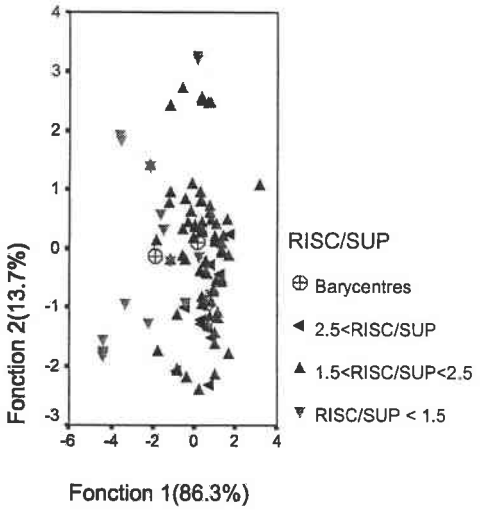


FIG. 3.22: Résultat de l'AFD avec le critère RISC/SUP au niveau séquence

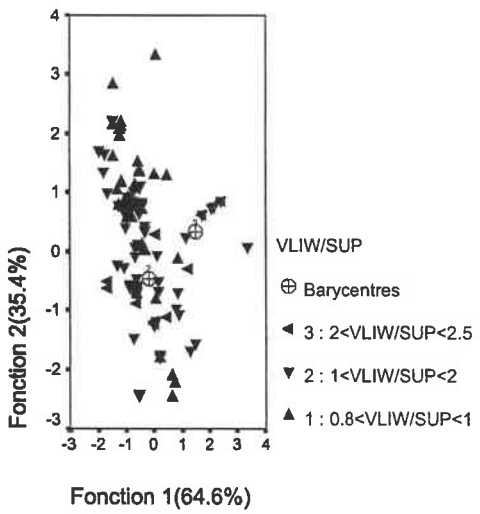


FIG. 3.23: Résultat de l'AFD avec le critère VLIW/SUP au niveau séquence

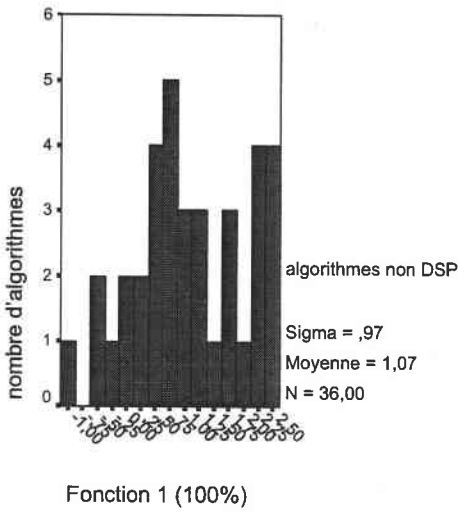


FIG. 3.24: Résultat de l'AFD avec le critère Non-DSP au niveau algorithme

Comme la figure le montre, il n'existe pas de bonne discrimination et les variables sélectionnées ne constituent pas des choix adéquats. Les plus grandes corrélations absolues existent entre la fonction 1 et les variables ALU-MSR, DEPCY, WR-MSR et RD-MSR.

Globalement, les résultats des mesures, au niveau des séquences montrent que la sélection de séquences comme le niveau de granularité n'est pas une bonne sélection. Cette situation est tout à fait logique et raisonnable parce que dans la plupart des algorithmes considérés, il y a beaucoup d'instructions de décision telles que les sauts conditionnels. Cette fréquence de ce type d'instruction diminue la taille de séquence et les caractéristiques des algorithmes n'apparaissent pas bien dans des petits blocs d'instructions virtuelles.

Nous avons également étudié, le pouvoir de discrimination des variables pour distinguer les traitements orientés DSP des traitements non-orientés DSP. Pour cela, trois tâches typiques (benchmark) des noyaux de traitements DSP ont été examinées : la convolution, le produit scalaire de vecteurs et le calcul d'un bloc réel du filtrage FIR. Les figures 3.24 et 3.25 montrent les résultats au niveau algorithmes. Comme les résultats l'indiquent, il existe une bonne discrimination entre les traitements non-DSP et les traitements DSP. La fonction discriminante obtient les plus grandes corrélations avec les variables INDCT-ADR, WR-MSR et N-REG.

Les résultats au niveau des séquences sont illustrés dans les figures 3.26 et 3.27. Ici, la discrimination est plus faible qu'au niveau algorithme.

Dans cette étude, les résultats globaux indiquent qu'en utilisant des caractères adéquats du code virtuel, nous pouvons avoir une bonne vision prévisionnelle des comportements de différentes architectures dans l'exécution des applications. Dans l'étape de conception de l'architecture multiprocesseurs, on peut, à l'aide de ces résultats, déterminer le nombre approprié d'unités de traitement et choisir leur architecture. Cette méthode peut également être utilisée comme méthode de répartition des algorithmes entre les différentes architectures.

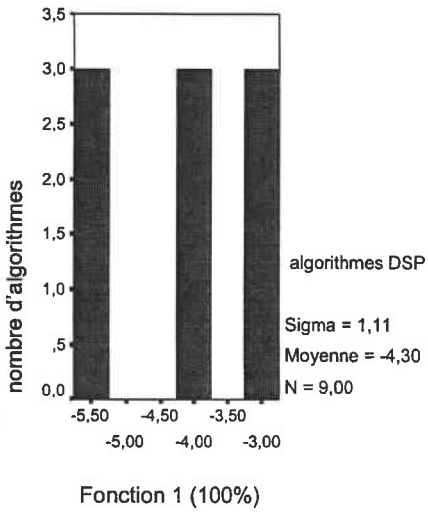


FIG. 3.25: Résultat de l'AFD avec le critère DSP au niveau algorithme

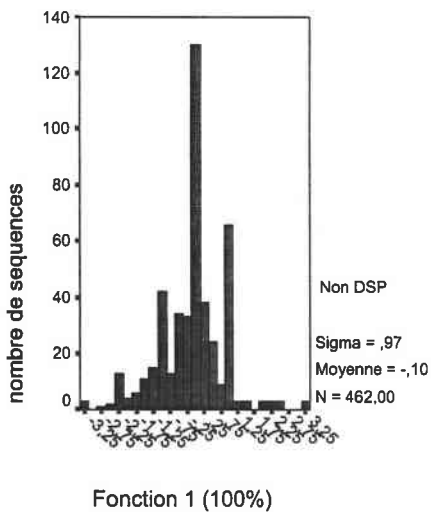


FIG. 3.26: Résultat de l'AFD avec le critère Non-DSP au niveau de séquence

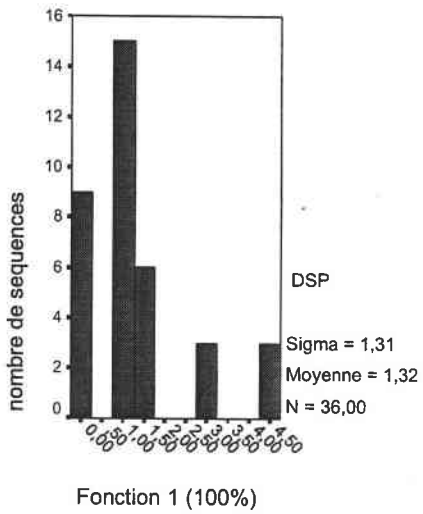


FIG. 3.27: Résultat de l'AFD avec le critère DSP au niveau de séquence

3.9 Conclusion

Dans ce chapitre, nous avons proposé une méthodologie pour vérifier la performance de diverses architectures pour implanter les protocoles de réseaux haut débit. Tout d'abord, nous avons étudié les opérations requises pour traiter ces protocoles, et puis nous avons regroupé les tâches du traitement. Par la suite, nous avons proposé une méthodologie pour évaluer les performances des architectures pour exécuter ces tâches. Dans cette méthodologie, un compilateur produisant du code virtuel pour un processeur virtuel a donc été développé pour les algorithmes écrits en langage C. Ce code virtuel est ensuite reciblé sur différentes architectures. Nous avons utilisé la chaîne de Markov pour convertir les mesures statiques en mesures dynamiques.

À partir de ces mesures et à l'aide de méthodes d'analyse multi-dimensionnelle de données, on peut avoir une bonne vision pour choisir les architectures adaptées. En d'autres termes, cette méthode offre une aide à la conception d'architectures ayant un niveau élevé de parallélisme grâce à l'exécution simultanée de tâches et en considérant d'affinité entre les algorithmes et les architectures basées sur un modèle multiprocesseurs. Les architectures CISC, RISC, Superscalaire et VLIW ont été évaluées par la méthode proposée. Les résultats d'analyse des mesures montrent que les deux architectures RISC et DSP sont les bonnes candidates comme cœurs de processeur pour exécuter les algorithmes représentatifs généraux et DSP respectivement.

Les résultats indiquent que cette méthode peut également être utilisée comme une méthode de répartition des algorithmes entre les différentes architectures dans un modèle multiprocesseurs hétérogène. De cette façon, une bonne sélection de variables représentatives extraites du code virtuel permet de choisir l'architecture cible la plus appropriée pour obtenir une bonne performance. Les résultats d'analyse des mesures indiquent que la prise en compte de la granularité au niveau des algorithmes donne des résultats plus intéressants qu'au niveau des séquences.

Chapitre 4

Modèle proposé

4.1 Introduction

Dans ce chapitre, nous étudions tout d'abord les différentes possibilités de conception d'une architecture multi-processeurs pour le traitement des protocoles, puis nous proposons un modèle architectural. Ce modèle permet l'utilisation performante de la bande passante disponible sur les réseaux haut débit. Le but est de réaliser une architecture de traitement des protocoles standards de ce type, définis par le modèle OSI, pour les terminaux d'utilisateurs, dans les réseaux de débit supérieur à un Gbits/s. Pour évaluer ce modèle, nous l'avons simulé en utilisant un réseau de files d'attentes (*Queuing Network*). Enfin, nous présentons les résultats de la simulation.

4.2 Parallélisme dans le traitement de protocoles

Les architectures de réseaux sont typiquement structurées sous la forme de couches hiérarchiques de protocoles, comme cela a été présenté dans le chapitre 1. Les exemples comprennent le modèle OSI, le modèle UIT-T et le modèle Internet. Le traitement parallèle de protocoles peut être réalisé entre différents utilisateurs (connectés et non-connectés) et entre les voies de réception et de transmission. Les modèles de résaux permettent également l'exécution pipeline entre différentes couches de protocoles ou dans une couche individuelle [Zitterbrat91]. Cette section traite chacune de ces formes de parallélisme.

4.2.1 Pipeline entre les couches de protocoles

Les couches de protocoles définies dans les architectures de réseaux sont indépendantes. Elles effectuent les fonctions de couches hautes, indépendamment de couches basses. Les interfaces de service entre ces couches de protocoles sont typiquement utilisées de façon asynchrone. Des messages sont utilisés pour la communication et la synchronisation parmi les différentes couches. Selon le flot de ces unités de données, chaque niveau du protocole peut les combiner ou les partager avant de les transmettre au niveau suivant. Chaque protocole (ou partie de protocole) effectue certaines opérations sur les unités de données, telles que le décodage de la zone adresse, le calcul et la vérification du nombre de séquences, et la génération/consommation des acquittements. On peut observer une telle forme de pipeline dans la plupart des équipements de réseau existants. Par exemple, la couche physique est habituellement implantée dans une puce spécialisée et le

protocole de liaison de données est habituellement exécuté par un processeur généraliste. Parallèlement, le processeur hôte traite les protocoles des couches plus élevées.

Si différentes parties de protocole dans les couches d'une architecture de réseau sont exécutées en pipeline, la vitesse du pipeline résultante sera limitée par la couche la plus lente. Par conséquent, afin de maximiser l'accélération d'une exécution, une analyse exacte est nécessaire pour identifier les conditions de traitement des différentes couches et pour équilibrer la charge sur les ressources de traitement disponibles.

4.2.2 Parallélisme et pipeline dans les couches de protocoles

Le comportement de la plupart des protocoles est décrit sous forme de machines à états finies (FSM : *Finite State Machine*) ou de tables de transition d'états. Pour chaque coté de la connexion utilisée par une couche de protocole, au moins une machine à états existe des deux côtés de la connexion. Par conséquent, une telle machine à états peut être implantée comme un processus cyclique qui s'exécute plus ou moins indépendamment des machines à états pour d'autres connexions. Naturellement, ces FSM peuvent ne pas être nécessairement implantées sous forme de processus individuels. Elles peuvent également être implantées par un processus unique qui entretient toutes les connexions en service, en lisant et en mettant à jour des vecteurs d'état correspondant aux différentes connexions. Il est clair que lorsque l'accès aux ressources partagées est requis dans une couche (par exemple, temporisateurs ou buffers), une forme de synchronisation est nécessaire.

Pour certains types de protocoles, les machines à états pour les différentes connexions peuvent s'exécuter en parallèle, ce qui donne l'augmentation du débit sortie globale. Ainsi, on peut implanter les machines d'état de la réception et de l'émission transmission en parallèle.

Il existe d'autres types de parallélismes entre les différentes parties d'un protocole. Par exemple, une des opérations très courantes dans la plupart des protocoles est le calcul du CRC. Cette opération peut s'exécuter en parallèle avec les opérations de l'inspection de l'en-tête d'un paquet. Sur une échelle plus fine encore, on peut diviser les étapes de traitement au sein d'une même couche de protocole en différents processus ; cela peut s'exécuter en mode pipeline et en parallèle pour la réception et la transmission. L'interprétation des demandes de service de la couche immédiatement supérieure, la génération de PDUs (*Protocol Data Unit*), et le passage des demandes de service à la couche immédiatement inférieure constituent de tels exemples.

Dans tous les cas mentionnées, la synchronisation, l'équilibrage de charge et le partage des ressources sont très importants. Nous discutons de ces sujets dans la section suivante.

4.2.3 Communication et synchronisation

Malgré les nombreuses formes de parallélisme et les différentes possibilités de pipeline présentées dans les sections précédentes, il n'existe pas toujours de manière assez simple d'obtenir une implantation performante de type pipeline et/ou parallèle dans une architecture des couches de réseaux. Chaque architecture avec sa structure individuelle doit être étudiée pour identifier les parties plus adaptées d'être placé dans différents processus en parallèle. Un facteur important dans cette analyse est la relation entre les exigences de traitement et les exigences de communication et de synchronisation des parties individuelles. Par exemple, l'exécution de diverses étapes d'un protocole dans un pipeline à grain fin sur plusieurs processeurs n'est valable que les surcoûts de l'ordonnancement et de la synchronisation sont très faibles en comparaison du travail effectué dans les diverses

étages du pipeline.

Un autre facteur important est la manière dont l'architecture est spécifiée. Si, par exemple, on considère les spécifications d'un protocole de couche où seul une FSM simple contrôle la transmission et la réception des données dans la connexion, il est plus difficile de réaliser l'exécution parallèle de ce protocole que dans le cas de FSM séparées. De même si un protocole utilise un flux de données séparé pour retourner des paramètres tels que les acquittements, la quantité de synchronisation requise entre les machines à états de la réception et la transmission sur le nœud de réception peut être minimale. Si cette information est envoyée dans les paquets de données (piggybacked), les machines à états de réception et transmission peuvent avoir besoin de synchroniser ou de partager l'information de spécification du protocole pour chaque trame reçue. Notons également que dans la conception d'une architecture parallèle pour le traitement des protocoles de couches, la façon dont les différents composants communiquent devrait également être étudiée.

Dans le modèle de référence OSI, les protocoles de couches adjacentes échangent des information par les primitives de service. Dans une implantation, ces primitives de service sont souvent tracées sur des messages ou des appels de procédures. Les primitives contiennent habituellement des paramètres et des valeurs de retour. Certaines primitives de service (par exemple, DATA.request, DATA.indication) peuvent également contenir des unités de données. La duplication des unités de données entre protocoles de couches, comme cela se fait souvent, devrait être évitée dans toute implantation haute performance parce qu'elle limite le débit sortie considérablement. Afin d'éviter de copier des données de (protocole) dans une implantation multi-processeur, tous les processeurs de protocole et le processeur hôte doivent avoir un accès partagé à une copie unique des unités de données de protocole [Kaiserwerth93].

4.3 Modèle proposé

Naturellement la dépendance logique entre les différentes parties d'un traitement de protocoles implique de respecter un certains ordre de traitement de parties. Comme nous avons dit dans la section précédente, il y a deux approches principales pour accélérer le traitement et augmenter le débit sortant : le pipeline et le parallélisme. Nous avons choisi la méthode pipeline, en essayant de compenser les points faiblesses de cette méthode par d'autres techniques. La figure 4.1 présente le modèle architectural proposé. C'est un modèle général de type multiprocesseurs qui utilise différentes unités de traitement. Cependant, l'idée principale est d'utiliser des unités de traitement esclaves (SPU : *Slave Processing Unit*) spécialisées, (au lieu d'unités identiques et générales) et une unité maître (MU : *Master Unit*) qui dirige les opérations des SPU. Les SPU peuvent être soit des petits cœurs de processeurs spécialisés, soit des unités matérielles dédiées. Un réseau d'interconnexion relie les SPU et le MU. Le chemin de données principal est composé d'une chaîne de traitement pipeline du traitement qui peut avoir une configuration linéaire, ou non linéaire. Les paquets de données entrent dans la chaîne du pipeline au niveau du premier étage mais ne le quittent pas nécessairement à l'étage final.

Les flux de données entre le MU et les SPU (moins critiques du point de vue du débit) pour transférer les différents paramètres des tâches des SPU. Le réseau d'interconnexion offre également la possibilité de la distribution du code aux SPU par le MU. Pour obtenir un débit plus élevé, on peut remplacer la chaîne du pipeline de SPU par une matrice de SPU, ce qui constitue un modèle étendu de notre proposition pour le traitement parallèle de différents paquets.

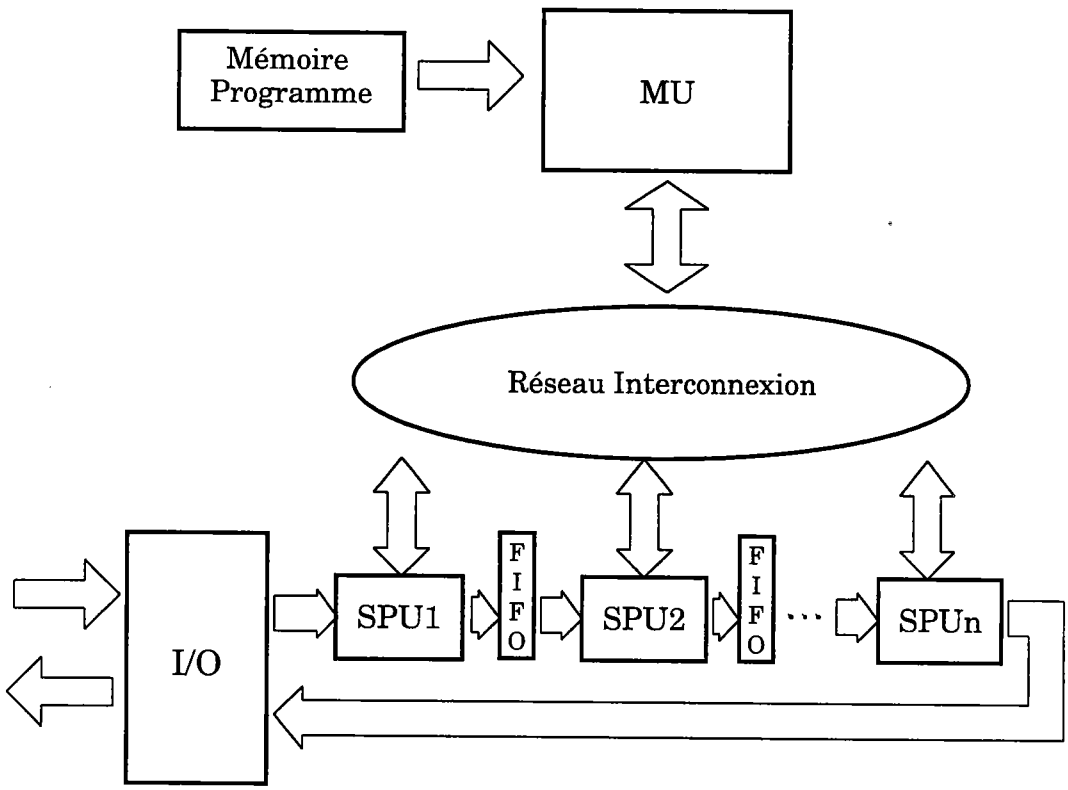


FIG. 4.1: Modèle proposé

4.3.1 Chemin de données principal

Le chemin de données principal est composé d'une chaîne de SPUs reliés en série. Cette architecture pipeline est bien adaptée aux applications réseaux telles que les protocoles des couches. En effet, le modèle fonctionnel typique de telles applications est une chaîne d'étapes relativement indépendantes, les données circulant d'une fonction la suivante dans la chaîne. Dans notre modèle, nous pouvons assigner une ou plusieurs de ces fonctions à un SPU unique. La performance du système peut être affectée de manière significative par les architectures choisies pour les SPU.

Pour une bonne répartition de la charge entre les SPU, le modèle utilise des unités du tampon intermédiaire. Ces tampons participent à la synchronisation des données lors des transferts entre les étages successifs du pipeline. Ils doivent être accessibles à deux SPU simultanément. Par conséquent, nous utilisons un modèle de FIFO (*First-In-First-Out*) à double accès.

Selon le compromis souhaité pour le système entre vitesse et coût, le modèle peut utiliser un pipeline linéaire ou non linéaire. Dans le pipeline linéaire, il n'y a que des chemins de direct vers l'avant (*feedforward*) entre les SPU, et le nombre d'étages physiques doit être égal au nombre d'étapes logiques. Par exemple, si un protocole est composé de cinq fonctions successives, nous pouvons le mettre en application sur un pipeline linéaire de cinq SPU, comme illustré sur la figure 4.2.

Supposons que les fonctions puissent être regroupées dans un plus petit nombre de

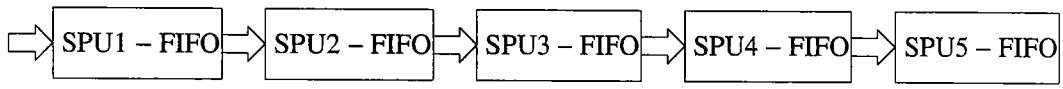
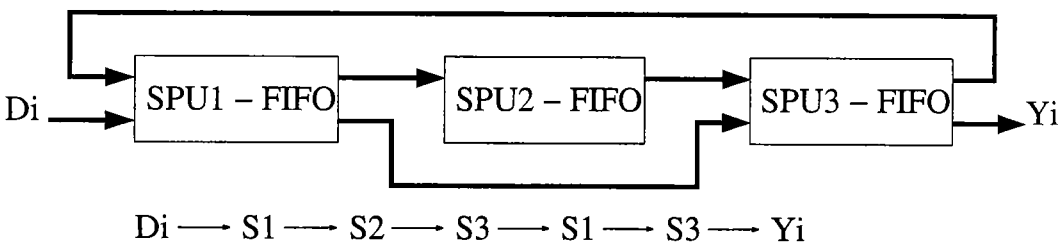


FIG. 4.2: Pipeline linéaire

catégories de fonctions. Par exemple, supposons que les fonctions 1 et 4 et les fonctions 3 et 5 soient semblables, au niveau de certains caractères : trois SPU peuvent alors être employés au lieu de cinq. Cependant, la réutilisation de certains SPU pour exécuter les cinq fonctions exigera une utilisation non linéaire du pipeline. De cette façon, on peut regrouper les fonctions et assigner chaque catégorie à l'architecture appropriée. Dans ce cas, les chemins direct (*feedforward*) et de retour (*feedback*) sont requis pour relier les SPU. Dans le pipeline, le nombre d'étapes logique est alors supérieur au nombre d'étages physiques. Certains SPU doivent maintenant traiter alternativement plusieurs fonctions. En outre, une latence additionnelle est générée dans le pipeline pour un fonctionnement correct. La figure 4.3 montre un mode de fonctionnement permettant d'éviter les conflits, pour l'exemple présenté.



	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15	t16
SPU1	D1	D2		D1	D2	D3	D4		D3	D4	D5	D6		D5	D6	
SPU2		D1	D2				D3	D4				D5	D6			
SPU3			D1	D2	D1	D2		D3	D4	D3	D4		D5	D6	D5	D6

FIG. 4.3: Pipeline non linéaire

Dans ce cas, le débit de données en sortie est réduit. En effet, quand un SPU est en train de traiter un bloc de données provenant d'un autre SPU (par exemple, quand le SPU1 reçoit le bloc de données D1 du SPU3), il ne peut pas traiter un nouveau bloc de données (D3 dans l'exemple). Dans cet exemple, le débit en sortie n'est plus que 2/5 de celui de l'exemple précédent.

4.3.2 MU

Le concepteur peut employer un pipeline linéaire ou non linéaire. Dans ce dernier cas, le choix des fonctions et de leur distribution efficace sur les SPU ainsi que la commutation correcte des données vers les SPU sont à la charge du MU. De plus, dans tous les cas, le MU est responsable de l'envoie des paramètres de fonctionnement des fonctions à chaque SPU. Pour effectuer ces tâches, le MU est composé de trois parties principales (Figure 4.4).

Le premier est un contrôleur de commutateur (switch handler) qui commande l'unité

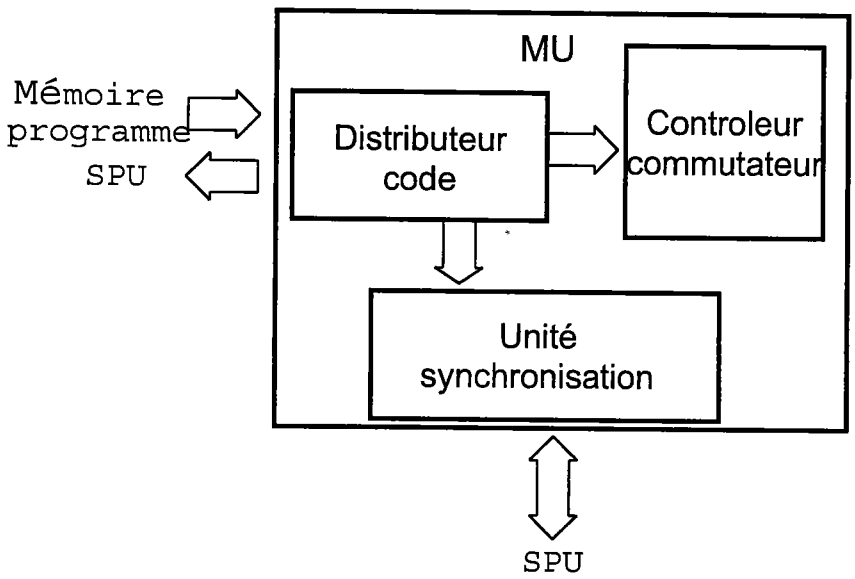


FIG. 4.4: Schéma général du MU

de commutation du réseau d'interconnexion. La deuxième partie est une unité de décodage et de distribution qui reçoit les programmes et les paramètres de la mémoire principale et les distribue parmi les SPU.

La dernière partie est une unité de synchronisation qui s'occupe de l'équilibrage de charge et de la synchronisation des SPU. En raison de la dépendance entre la commande de commutateur et la répartition de charge pour la distribution du code parmi les SPU, l'unité du décodage et de distribution est chargée de fournir l'information requise aux deux autres unités.

Tout ce qui est dit précédemment, concernant les tâches du MU au niveau de la commutation des unités de données, est basé sur une hypothèse de centralisation pour permettre le contrôle des SPU par le MU. Cette hypothèse deviendra trop chère au niveau du volume de données échangées entre les SPU et le MU, quand le nombre de SPU augmente. Nous parlons de l'autre hypothèse alternative de décentralisation dans la section suivante.

4.3.3 Réseau d'interconnexion

Dans notre modèle, il existe deux blocs du réseau de connexion dans le réseau d'interconnexion. Le premier bloc contient quatre bus (2 bus adresse et 2 bus données). Le second est un réseau de commutation. Les bus permettent le transfert des données (les paramètres) et du code entre le MPU et les SPU. Le réseau de commutation indique les chemins de circulation des paquets entre les SPU dans la structure pipeline. Nous avons proposé deux hypothèses de fonctionnement concernant le contrôle de la circulation des données entre les SPU : l'approche centralisée, synchrone et statique et l'approche décentralisée, asynchrone et dynamique.

Dans la première approche, à la fin de chaque période dévolue au traitement des tâches, le réseau de commutation détermine quel SPU traitera le paquet de données. Ce commuta-

teur reçoit les signaux de commande nécessaires du MU. De cette manière les communications entre SPU peuvent être prévues au moment de la compilation, et la détermination dynamique des destinations des données pendant l'exécution n'est pas nécessaire . Par conséquent, il est possible de déterminer les communications à l'avenir dans un système statique. Un schéma de niveau fonctionnel de ce commutateur dans le cas d'un réseau d'interconnexion pour 4 SPU est montré dans la figure 4.5.

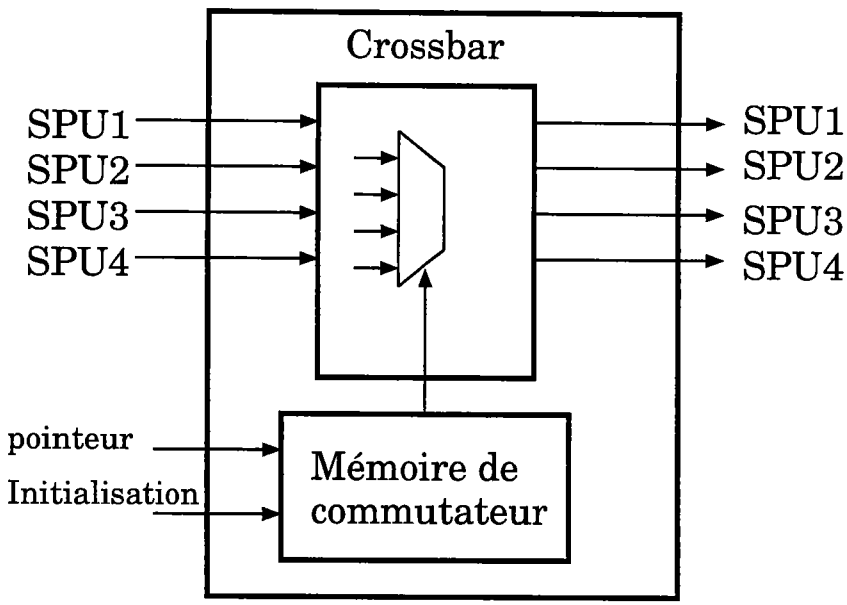


FIG. 4.5: Commutateur du réseau d'interconnexion

Dans cette architecture, une mémoire de commutation contient les différents arrangements du multiplexeur et un pointeur adresse les choix du multiplexeur aux instants voulus. En mode pipeline linéaire, le pointeur choisit la même séquence dans chaque temps d'exécution de tâche de manière répétitive. La détermination de la durée du temps d'exécution de tâche et l'initialisation de la mémoire du commutateur sont effectuées par l'unité contrôleur de commutation du MU, pendant la compilation. En raison du pipeline synchrone, le temps d'exécution de tâche dans notre modèle est le temps d'exécution de tâche le plus long des différentes tâches s'exécutant aux différents SPU.

Le commutateur et son mode de fonctionnement sont parmi les parties les plus critiques à mettre au point, lors de la conception. Lorsque des systèmes sur puce (*on-chip*) de grande taille sont conçus, la communication globale est le facteur limitant la performance globale du système. C'est pour cela que nous proposons une deuxième solution, décentralisée et dynamique. Nous expliquons tout d'abord la décentralisation et puis le dynamisme.

La décentralisation signifie qu'un contrôle décentralisé peut diminuer le coût global du système. Dans ce cas, le contrôle de la circulation des données se fait par les SPU. Après le traitement d'un paquet, chaque SPU doit déterminer la destination du paquet selon un schéma de partition globale du programme qui est préparé par le MU. En conséquence, sans require à la commande du commutateur par le MU, un réseau de type commutation de paquets peut acheminer chaque paquet entre les SPU jusqu'à la fin du traitement du paquet. Dans ce cas, il faut naturellement joindre une petite partie adresse aux paquets.

Dans une architecture réseau, les paquets traversent des couches différentes, et dans chaque couche il y a plusieurs fonctions. Généralement, les paquets ne passent pas tous par toutes les fonctions et donc la longueur du traitement est variable en fonction de chaque paquet. Par conséquent, si nous utilisons un système dynamique de cheminement des paquets entre les SPU, l'utilisation des ressources sera plus performante moyennant cependant un surcoût additionnel.

La sélection entre les deux méthodes (centralisée et décentralisée) dépend du nombre de fonctions, de la granularitée des fonctions et de la diversité des SPU dans la conception du système.

4.3.4 Architecture du SPU proposée

Les SPU peuvent être, soit de petits cœurs de processeurs spécialisés soit des unités matérielles dédiées. La figure 4.6 montre l'architecture générale d'un SPU de type petit cœur de processeur spécialisé. Elle est composé de cinq parties principaux : le buffer d'entrée, la table d'instruction (IT : *Instruction Table*), l'unité de traitement, les bus d'instructions, de données et d'adresses, et la mémoire de données (DM).

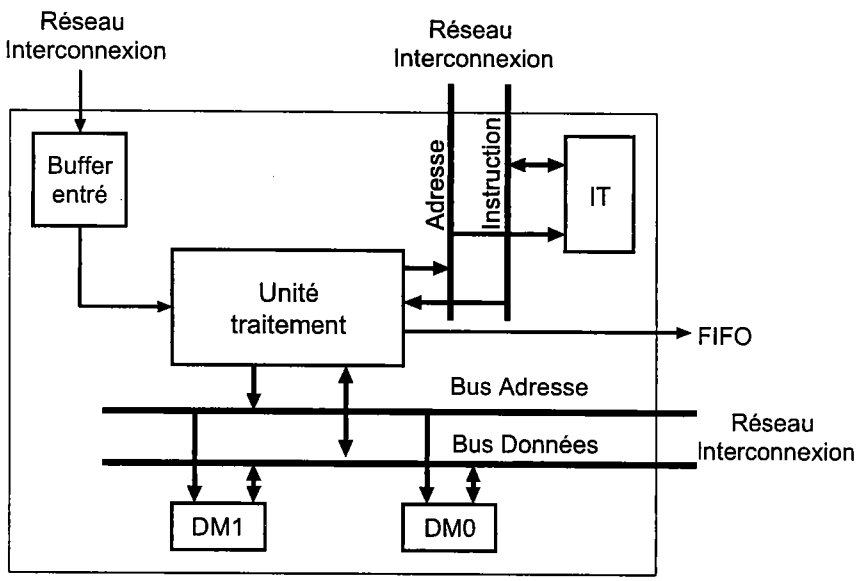


FIG. 4.6: Architecture de SPU

Le *buffer* d'entrée est utilisé pour le stockage des données entrantes. Ce *buffer* capture à chaque cycle d'horloge les nouvelles données sur le port d'entrée. La table d'instruction enregistre les instructions reçues du MU. L'unité de traitement est responsable des tâches de traitement des données. Elle utilise deux mémoires, l'une pour le stockage temporel des données et l'autre pour le stockage des paramètres des tâches. Ces paramètres, par exemple, sont employés pour le traitement des en-têtes des paquets. Ils dépendent des différents protocoles.

4.4 Simulation du modèle

Comme nous l'avons expliqué dans la section précédente, il existe deux hypothèses de fonctionnement concernant la synchronisation et la communication des SPU et du MU.

En considérant les avantages de la méthode décentralisée, nous avons basé notre modèle sur cette hypothèse et nous l'avons étudié par la simulation. En théorie, il serait souhaitable de mettre en uvre un modèle qui tienne compte des caractéristiques des algorithmes traités et des données. Dans la pratique, pour simplifier le problème, nous avons utilisé un modèle de simulation limité au caractéristiques statistiques des données.

4.4.1 Méthode de simulation

Nous avons simulé le fonctionnement du modèle par un réseau de files d'attentes (*Queuing Network*). Ce réseau contient des composants connectés les uns aux autres et soigneusement ajustée pour prendre en compte les caractéristiques propres d'application de bout en bout. Ces ajustements aident à optimiser le débit tout en préservant la stabilité globale du système. Chacune des ressources représente une file dans laquelle sont placées les demandes en attente d'utilisation de cette ressource.

Lorsque un paquet de données arrive pour être traité dans un SPU, il est placé dans une file d'attente. Puis, lorsque c'est son tour, il est traité par le SPU. Ensuite il est envoyé vers un commutateur de paquets pour être acheminé vers un autre SPU. Dans notre réseau, chaque nœud est composé de trois parties : une file d'attente qui représente l'attente de chaque paquet pour prendre le service ; un serveur avec une fonction de distribution de probabilité appropriée qui représente le traitement du paquet par le SPU ; un serveur avec une fonction de distribution de la probabilité appropriée qui représente le temps de buffering et de commutation du paquet. La figure 4.7 montre les composants de chaque nœud.

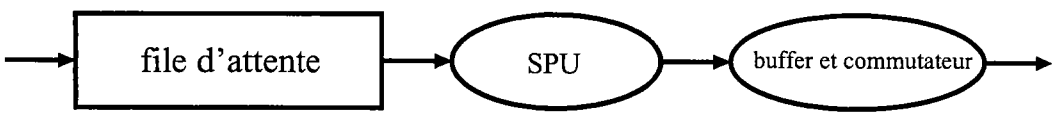


FIG. 4.7: Combinaison de chaque nœud dans le réseau de files d'attente

La configuration du réseau de files d'attente n'est pas fixe dans notre simulation et une matrice de probabilités de transition détermine quel chemin traverse chaque paquet pour atteindre la fin du traitement. En effet, cette matrice simule le passage des paquets dans les diverses fonctions.

Nous avons utilisé la méthode de simulation par événements discrets (*Discrete Event Simulation*). La simulation par événements discrets est pour construire des modèles permettant d'observer le fonctionnement temporel (ou dynamique) d'un système lequel est soumis à une sucession d'événements qui le modifient. Ces simulations ont vocation à appliquer des principes simples à des systèmes de grande taille.

La simulation discrète se divise en deux grandes catégories : synchrone ou découpage temporel (*time-slicing*) et asynchrone ou par événements (*event-sequencing*). Dans le mode synchrone, on simule à chaque fois le passage d'une unité de temps sur tout le système. Dans le mode asynchrone, on calcule l'arrivée du prochain événement et on ne simule qu'événement par événement, ce qui permet souvent des simulations rapides, bien

Applications	requis de la qualité	caractéristiques
Paquet téléphonique	le retarde bas, le perte moyen, jitter bas	CBR[1] ou VBR [2]
Email	pas du perte	le meilleur effort
web pages	throughput haut, le perte bas	self similar
vidéo uncompréssé	le retarde bas, le perte moyen	CBR
vidéo compréssé	le retarde bas, le perte bas	VBR
Telnet, FTP	le retarde bas, pas du perte	self similar
la commande du systèmes en temps réel	pas du perte, pas du retarde	rt-VBR [3]
les base de données distribué	pas du perte, le retarde bas	VBR

TAB. 4.1: Caractéristiques des applications

qu'un peu plus complexes à programmer [Ross89]. Nous avons utilisé le mode *event-sequencing*.

4.4.2 Modèles de trafic entrant

Les applications ont leurs contraintes de qualité de service (QoS) propres, comme la perte de paquet tolérable, le retard, etc. Des paramètres, comme le taux maximal de paquets, la longueur des paquets, la valeur limite du retard et de la taille des salves (*bursts*) de paquets, sont employés pour caractériser le trafic des applications.

Les applications peuvent être globalement classées en deux types : temps réel ou non. Les applications non temps réel sont sensibles à la perte de paquets mais flexibles vis à vis des retards. Le trafic en temps réel peut être caractérisé par les limites de retard et *jitter* (la variation du retard pour différents paquets causée par le stockage dans différents routeurs). Il est en général plus tolérant aux pertes de paquet. C'est le cas, par exemple, du trafic audio/video.

La modélisation du trafic implique la mesure et l'analyse du réseau ainsi que des applications déployées dessus. Tracer le trafic d'application aux modèles exige la connaissance de base de l'application elle-même. Il existe deux groupes de méthodes principales de caractérisation : les méthodes basées sur les mesures et les modèles statistiques, et les méthodes basées sur le modèle déterministe, mais il y a toujours des chevauchements entre les deux. Dans la méthode basée sur les mesures, on évalue le flot individuel ou collectif du trafic et on essaye de l'adapter au modèle analytique. La deuxième méthode utilise le pire cas du comportement du traffic pour trouver ses caractéristiques du trafic. Le tableau 4.1 présente les caractéristiques des différentes applications [Kode01].

Le modèle de Poisson a été traditionnellement employé pour modéliser le trafic. Il utilise la distribution exponentielle. Le processus de Poisson est un processus sans mémoire, les valeurs des variables aléatoires aux différents temps sont non-corrélées.

Le trafic de données entrant a généralement modélisé par la distribution de Poisson,

[1] Constant Bit Rate
[2] Variable Bit Rate
[3] real time Variable Bit Rate

mais des études récentes ont réfuté ce modèle pour les applications réseau. Le modèle de Poisson échoue à capturer correctement les caractéristiques pour des flux de trafic tels que, HTTP (WWW), telnet et ftp. Les analyses statistiques de données WWW ont montré à être auto-similaire (*self-similar*) [Leland94][AhleHagh04]. Le trafic sur réseau Ethernet est également auto-similaire [Kode01].

Un flux du trafic est auto-similaire lorsqu'il présente une ressemblance structurelle pendant une durée assez longue. Il n'y a aucune longueur normale pour une salve (*burst*) de paquets dans le trafic auto-similaire, ils apparaissent à plusieurs échelles de temps. Le trafic auto-similaire montre la dépendance à une échelle de temps longue ce qui signifie que les valeurs à l'instant sont typiquement corrélées avec des valeurs à tous les moments futurs. En d'autres termes, l'éclatement du trafic ne diminue pas avec l'échelle du temps à laquelle le trafic est observé. Le degré auto-similitude peut être mesuré par le paramètre Hurst ou H : $0.5 < H < 1.0$ et quand $H \rightarrow 1$, le degré auto-similitude augmente [Leland94][Willinger97].

4.4.3 Résultats

Pour évaluer le modèle proposé et bien connaître ses capacités, nous l'avons simulé en utilisant la méthode simulation par événements discrets. Le modèle de simulation est composé de quatre SPU et les paquets de données circulent entre les SPU.

Le plan de circulation est spécifié par une matrice de transition. Elle consiste en une matrice de probabilités dont les éléments m_{ij} représentent la probabilité de transition des paquets de $Fonction_i$ à $Fonction_j$. Ces deux fonctions peuvent être dans un même SPU ou dans deux SPU différents. En effet, cette matrice simule le passage de paquets entre des fonctions différentes d'un protocole. Elle est la représentation mathématique d'une graphe orientée qui modélise la circulation de paquets entre différentes fonctions. Pour chaque simulation, un programme générateur de graphes aléatoires produit cette matrice, ce qui est nécessaire pour évaluer le modèle dans les différents cas de partitionnement des protocoles en fonctions. Dans le cas général, une seule condition est requise pour la matrice de transition :

$$\sum_j m_{ij} = 1 \tag{4.1}$$

Pour simuler le trafic de données entrant, nous avons choisi un trafic auto-similaire, parce que ce type de trafic est prédominant dans le secteur des réseaux haut débit. Nous avons expliqué les propriétés de ce type de trafic dans la section précédente. Dans [Leland94][Willinger97], il a été démontré que le trafic auto-similaire peut être généré par le multiplexage de plusieurs sources de paquets qui produisent des paquets, avec la fonction de distribution de Pareto, dans deux périodes ON et OFF. Les périodes ON correspondent aux trains de paquets côte à côte et les périodes OFF sont les périodes de silence entre les trains de paquets.

Pour générer le trafic auto-similaire, les séquences des tailles de périodes ON se produisent avec la distribution Pareto et la taille minimum de 1 correspond au paquet unique. Les périodes OFF se produisent de la même manière. La distribution de Pareto a la fonction de probabilité suivante :

$$P(x) = \frac{\alpha\beta^{\alpha}}{x^{\alpha+1}} \qquad x \geq \beta \tag{4.2}$$

où α est le paramètre de forme pour le trafic auto-similaire. Il doit être compris entre 1 et 2. La valeur de β est la valeur minimum de x et correspond à la taille minimum du train de paquet, 1 dans notre cas. La valeur moyenne de cette distribution est obtenue par :

$$E(x) = \frac{\alpha\beta}{\alpha - 1} \tag{4.3}$$

En utilisant de l'équation ci-dessous, la distribution Pareto s'obtient à partir de la distribution uniforme :

$$X_{Pareto} = \frac{\beta}{U^{1/\alpha}} \tag{4.4}$$

où U a une distribution uniforme entre 0 et 1.

La valeur de β, ou la valeur minimum de la variable aléatoire x, en distribution Pareto, est égale à 1 et correspond à la taille minimum du train de paquets [Kramer97].

La simulation a été effectuée pour 3 niveaux de charges différentes : charge lourde, charge moyenne et charge légère correspondant à l'arrivée de paquets 95%, 75% et 55% du temps, respectivement. Quatre tailles différentes de paquets ont été vérifiées : paquets de 200, 500, 750 et 1250 octets. La simulation a été effectuée pour une vitesse de ligne de 4 Gb/s. Un trafic auto-similaire est produit par différentes sources avec la distribution de Pareto des intervalles ON-OFF. Nous avons utilisé une distribution normale pour la variation des tailles de paquets. Le temps de *buffering* et de commutation des paquets sont affectés par ces variations. Pour les paquets les plus long, le temps critique est le temps de commutation et de *buffering* alors que, pour les paquets plus petits, le temps du traitement est plus critique.

Les résultats de la simulation pour une charge de 95% sont montrés dans la figure 4.8.

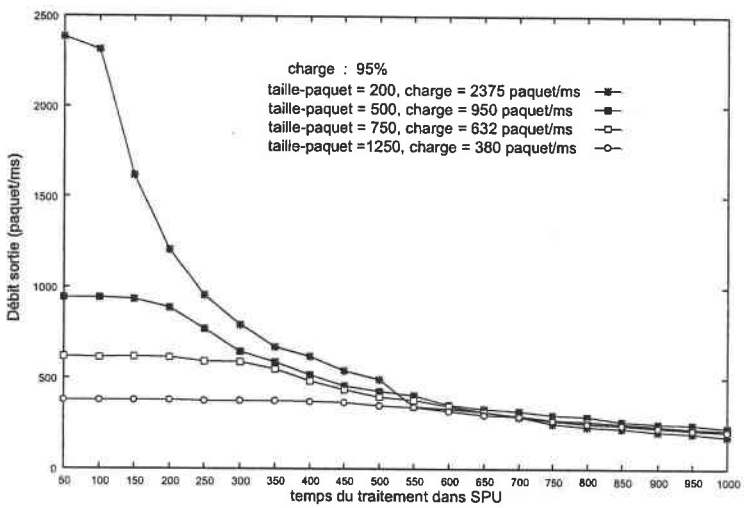


FIG. 4.8: Débit sortie pour une charge de 95%

Ce schéma présente la variation du débit en sortie (*throughput*) en fonction du temps

de traitement des paquets dans les SPU. Dans cette figure, les quatre graphes correspondent à quatre tailles de paquets différentes. Les résultats permettent quelques remarques importantes :

- Le modèle proposé peut répondre aux charges lourdes quelle que soit la taille des paquets, mais ce type de charge requiert des SPU plus rapides et des *buffers* plus grands pour éviter la saturation du système.
- La variation de débit en sortie (*Throughput*) augmente lorsque le temps du traitement dans SPUs diminue.
- On constate que la variation du débit en sortie par rapport au temps de traitement dans les SPU est plus grande pour les petits paquets que pour les paquets de grande taille. Cela signifie que, pour les paquets de petites tailles, le temps de traitement dans les SPU est prédominant par rapport au temps de buffering et de commutation. La taille des paquets affecte le débit sortie du modèle parce que plus de temps est nécessaire pour sauvegarder et commuter des paquets de grandes tailles. D'autre part, pour une vitesse de ligne fixe, le débit des paquets diminue quand la taille des paquets augmente.

Les résultats obtenus dans les figures 4.9 et 4.10 montrent que ce modèle est capable de bien répondre aux charges de 75% et 55% avec des SPU moins rapides. Toutefois, les remarques précédentes sont valables. Pour les charges plus faibles, la saturation de la sortie est due à des temps de traitement plus grands. Par exemple, pour une taille de paquets de 200 octets, lorsque la charge diminue de 42% passage de (95% à 55%) le système a besoin de SPU 3 fois moins rapides.

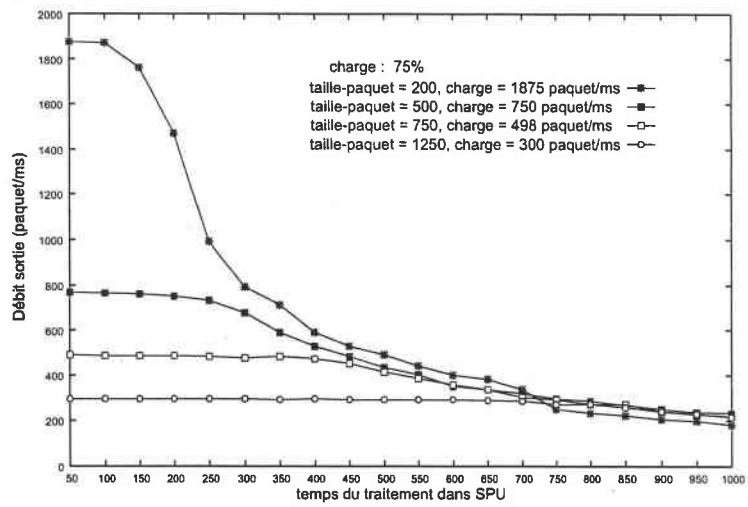


FIG. 4.9: Débit sortie pour une charge de 75%

Si on définit la latence comme le temps requis pour sortir le premier paquet après le démarrage du système, les figures 4.11, 4.12 et 4.13 représentent la latence du modèle pour les trois niveaux de charge. La distance entre les courbes indique que les paquets les plus grands sont ceux qui provoquent la latence plus grande. Lorsque le temps de traitement dans les SPU augmente, les courbes correspondant à deux tailles différentes de paquets se rapprochent, ce qui signifie que le temps de traitement dans les SPU est prédominant pour les paquets de petite taille.

Comme les figures le montrent, il est évident que pour une charge plus élevée, le temps

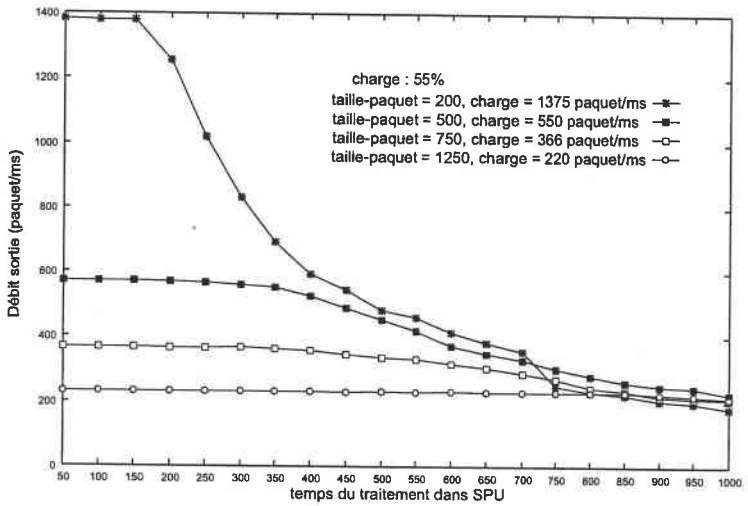


FIG. 4.10: Débit sortie pour une charge de 55%

de traitement est rôle prédominant.

La répartition de l'application en fonctions différentes et leur allocations aux différents SPU sont des tâches critiques lors de la conception d'une architecture basée sur le modèle proposé. Pour vérifier ce point, le modèle a été simulé pour plusieurs matrices de transitions. Les résultats sont comparés dans la figure 4.14.

Il existe une grande différence du comportement, au niveau du débit en sortie selon la nature plus ou moins désordonnées de la matrice de transition et allocation. Dans un premier cas, le modèle a été simulé pour une matrice assez désordonnées et 10 fonctions.

Dans un deuxième cas, plus de fonctions ont été considérées avec une matrice moins désordonnées et des conditions de fonctionnement particulières. Chaque SPU envoie les paquets traités au SPU suivant ou précédent et les paquets circulent parmi les SPU jusqu'à la fin du traitement. Contrairement cas précédent où l'on utilise une matrice de transition totalement générale, on considère ici certains conditions qui simplifient la matrice. Un paquet traité par la fonction F_i, sera traité par les fonctions F_{i+1}, F_{i+2} ou F_{i-1} avec les probabilités 0.6, 0.2 et 0.2. Dans ce cas, le même débit en sortie, peut être atteint avec des SPU quatre fois moins rapides.

Selon les résultats de la simulation, le modèle répond bien dans ce cas, même avec une matrice plus grande. Mais comme la figure 4.15 le montre, la latence est très sensible aux dimensions de la matrice de transition.

Nous avons également vérifié la réponse du modèle aux différentes vitesses de ligne. D'abord, pour un trafic auto-similaire où les trains de paquets présentent des silences entre eux, puis pour un trafic de type *stream* avec de longs trains de petits paquets sans silences. Les résultats sont présentés dans les figures 4.16 et 4.17. Dans le cas du trafic auto-similaire, quand la vitesse de ligne est multipliée par deux (passage de 1 à 2 Gb/s), les SPU doivent être deux fois plus rapides. Ce résultat est considérable, car ce modèle utilise 4 unités fonctionnelles pour réaliser 10 fonctions, alors qu'un modèle basé sur un pipeline linéaire requiert 10 unités fonctionnelles. Cette performance est dégradée avec le trafic de type *stream* sans silence. Dans ce cas, des SPU quatre fois plus rapides sont requis pour une augmentation de deux fois de la vitesse de ligne.

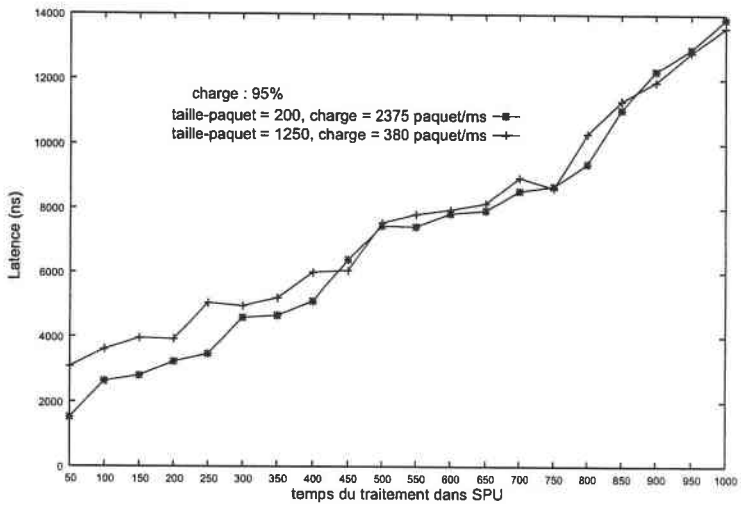


FIG. 4.11: Latence pour une charge de 95%

4.5 Conclusion

Dans ce chapitre, nous avons proposé un modèle architectural pour un processeur de traitement de protocoles de réseaux. L'idée principale est d'utilisation un modèle multi-processeurs basé sur l'emploi de processeurs spécifiques simplifiés. L'architecture proposée, en considérant les caractéristiques du trafic de réseau permet d'avoir une bonne performance.

En effet, ce modèle est basé sur deux notions principales : la première est que, si on partitionne les protocoles en fonctions indépendantes, les paquets de données ne seront évidement pas tous traités par toutes les fonctions. Par conséquent, les paquets peuvent être traités uniquement par les fonctions nécessaires de manière dynamique. Cependant cette approche augmente les coûts du système à cause du stockage et de la commutation de paquets entre SPU, mais elle permet une utilisation performante des ressources disponibles. La deuxième notion est celle de l'utilisation du temps de silence entre les trains de paquets. Ce temps permet d'utiliser une architecture pipeline non-linéaire qui présente une certaine latence pour faire circuler les données parmi les étages du pipeline, mais avec moins d'exigence au niveau du matériel.

Les résultats de la simulation montrent que le modèle peut être performant, surtout pour un trafic auto-similaire avec une charge moyenne. Ainsi, ces résultats indiquent que pour un trafic temps réel de type *stream* avec une charge lourde il faudra chercher des modèles parallèles adaptés. Les résultats de la simulation indiquent également que le partitionnement des protocoles en différentes fonctions (leurs granularités et les dépendances entre elles), et l'allocation de celles-ci aux SPU sont très importantes pour atteindre une bonne performance.

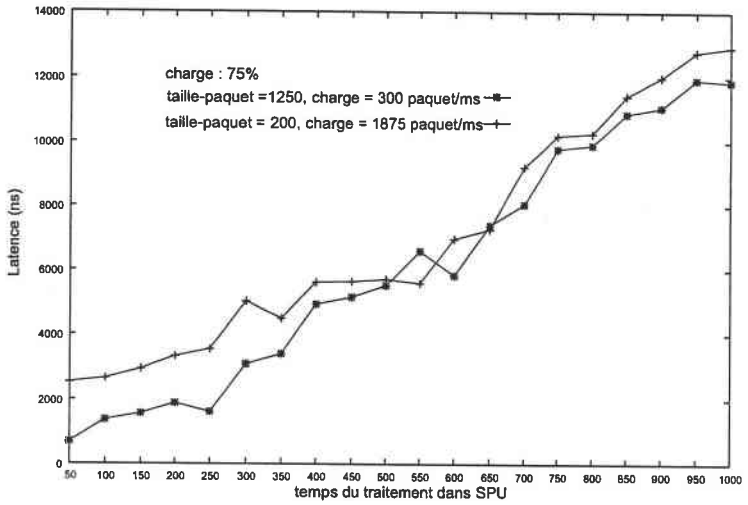


FIG. 4.12: Latence pour une charge de 75%

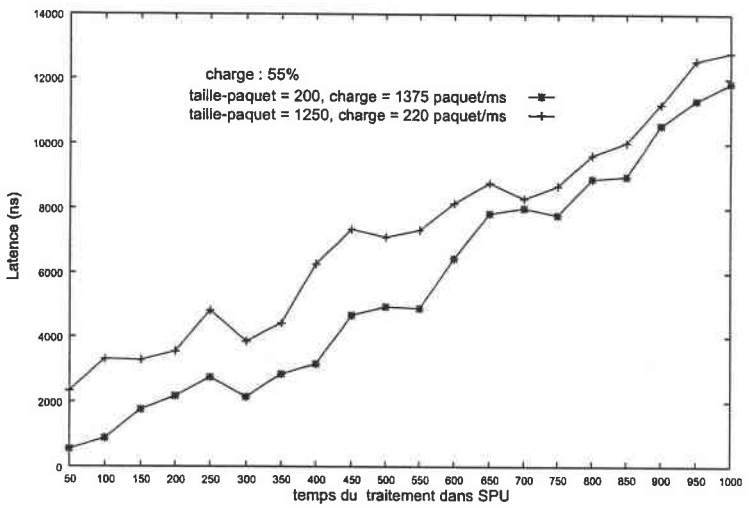


FIG. 4.13: Latence pour une charge de 55%

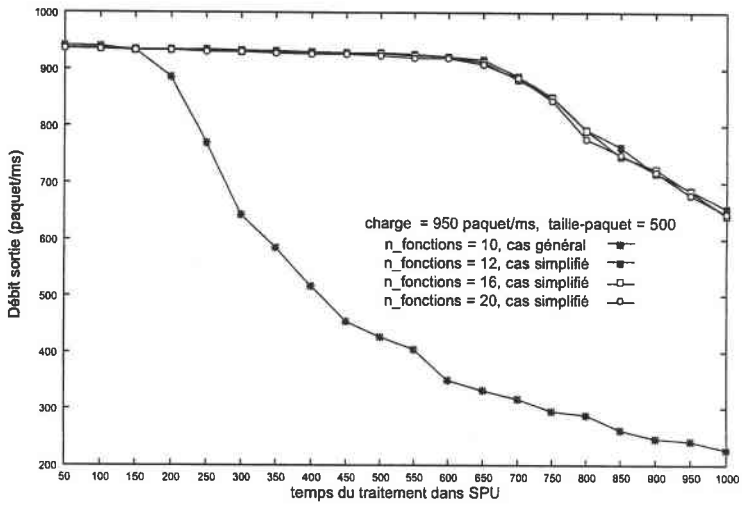


FIG. 4.14: Comparaison entre différentes manières de répartition

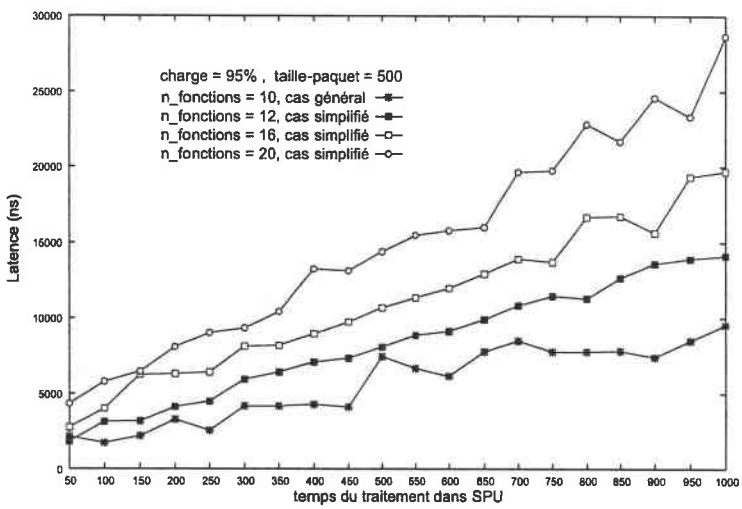


FIG. 4.15: Comparaison entre différent maniéres de répartition

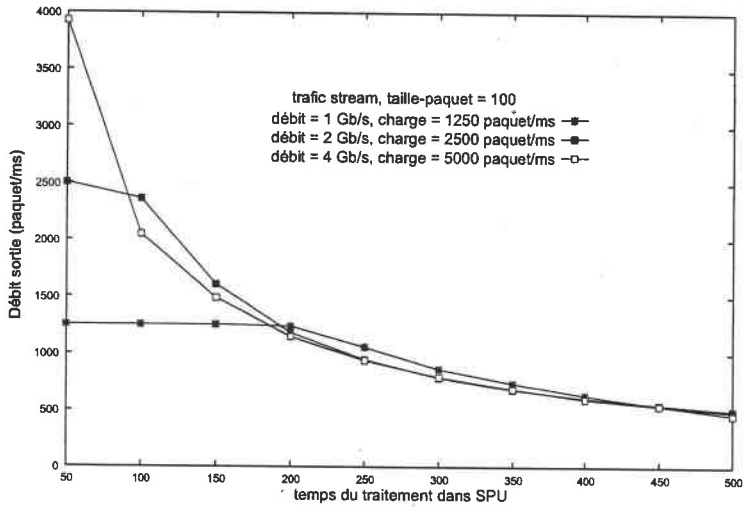


FIG. 4.16: Comparaison entre différentes vitesses dans un trafic stream

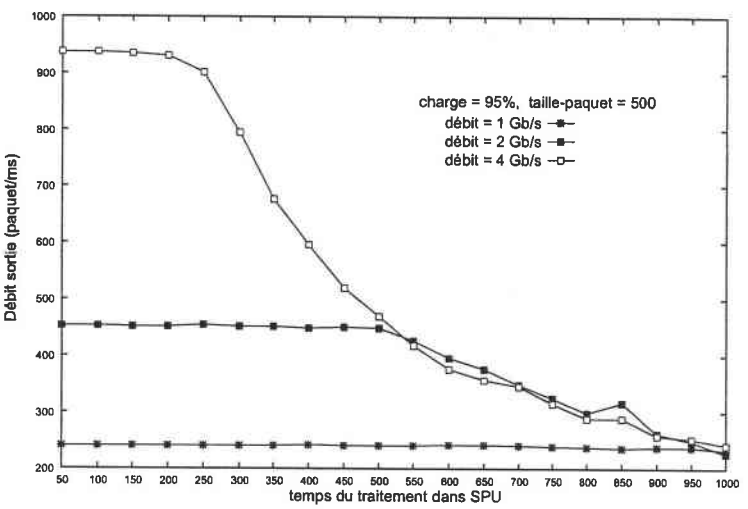


FIG. 4.17: Comparaison entre différentes vitesses dans un trafic auto-similaire

Chapitre 5

Implantation du SPU

5.1 Introduction

Nous avons étudié les possibilités d'un certain nombre d'architectures générales et répandues comme candidates pour les SPU pour l'exécution de certaines tâches générales des protocoles. Les architectures candidates sont, d'une part les architectures typique *Load-Store* ou RISC/CISC, d'autre part, deux architectures ILP (Instruction Level Parallelism) : superscalaire et VLIW.

Dans le chapitre 3, une étude a été effectuée pour extraire les tâches principales et communes des protocoles dominants d'aujourd'hui. Pour cela, nous avons étudié les protocoles TCP/IP, ATM, PPP, et l'IEEE 802.11. Dans cette vérification, nous nous sommes concentrés sur les protocoles des couches basses (réseau, liaison de données, MAC, et physique). Suite tâches ou algorithmes des couches inférieures des réseaux, nous les avons classées en deux catégories principales : les tâches orientées DSP, et les tâches non orientées DSP (les tâches générales).

Enfin, nous avons choisi une architecture de type RISC parce que, comme nous l'avons expliqué dans le chapitre précédent, pour nos applications sa performance est meilleure que celle des autres architectures. En outre, la simplicité de conception (matérielle) d'un RISC peut nous aider à réduire la complexité d'implantation des unités de traitement SPU qui fonctionnant comme de petits cœurs de processeurs spécialisés (figure 4.6). D'autre part, nous pouvons employer l'architecture DSP-RISC pour les tâches orientées DSP, ce qui permet de réduire l'hétérogénéité de l'architecture globale.

Dans ce chapitre nous expliquons la conception et l'implantation de ce type de SPU sur FPGA.

5.2 SPU-RISC

5.2.1 Chemin de données

Un processeur RISC entier de 32 bits a été conçu comme SPU de traitement des tâches générales. Le plan global a été pris de l'architecture DLX proposée par PATTERSON et HENNESSY dans [Patterson96]. L'architecture utilise un pipeline avec cinq étages. Les étages du pipeline sont : Lecture, Décodage, Exécution, Mémoire et Écriture. La figure 5.1 présente le schéma global du chemin de données de l'architecture.

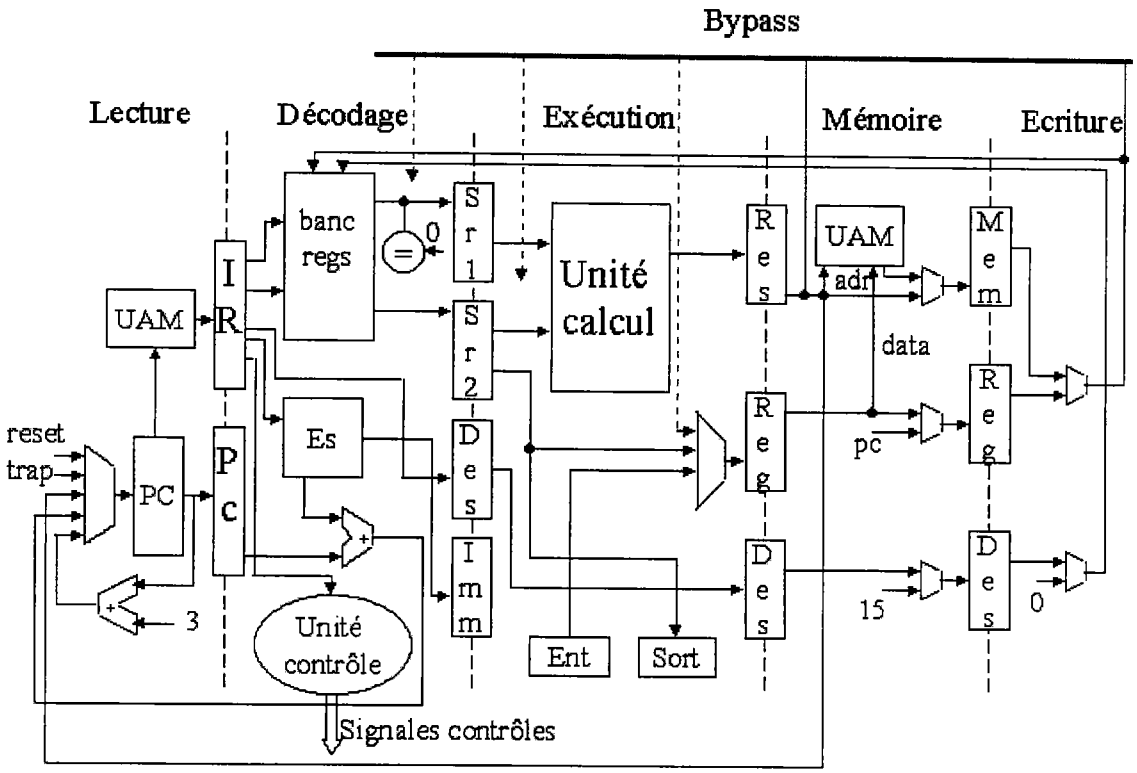


FIG. 5.1: Chemin de données SPU-RISC

Le chemin de données est composé d'un ensemble d'unités logiques interconnectées par les registres du pipeline et les multiplexeurs. Les commandes des multiplexeurs proviennent de l'unité de contrôle de l'étage décodage. De plus, les registres du pipeline mémorisent temporairement les valeurs entre les étages du pipeline.

Dans l'étage de lecture les instructions sont lues de la mémoire programme par l'UAM (unité d'accès mémoire). Cette unité est une interface qui permet de connecter le cœur à des mémoires avec différentes organisations. Dans cette architecture la mémoire de programme est séparée de la mémoire de données. Le registre PC (*Program Counter*) indique l'adresse de l'instruction à lire.

L'étage de décodage s'occupe du décodage et de l'interprétation des instructions. L'unité de contrôle génère les signaux de commande des étages suivants. Les instructions de branchement sont évaluées dans cet étage pour diminuer le retard dû à ce type d'instruction dans le pipeline. De plus, les données sont extraites soit du banc de registres pour certaines instructions, soit elles sont mises en forme par l'unité Es (extension du signe).

Les opérations arithmétiques et logiques sont exécutées dans l'étage d'exécution par l'unité de calcul. Le calcul des adresses de données, pour les opérations de lecture et d'écriture en mémoire, est effectué dans cet étage.

L'étage de mémoire a deux tâches principales : l'accès à la mémoire de données et les opérations dans l'instruction JUMP concernant l'adresse de l'instruction suivante.

Enfin, dans l'étage d'écriture, les opérations concernant la mise à jour du banc de registres ou d'écriture des résultats dans les registres, soit à partir de la mémoire, soit à partir de l'unité calcul, sont effectuées.

5.2.2 Formats des instructions

Cette architecture utilise un format d'instruction de 24 bits. Deux formats d'instructions utilisés sont montrés dans la figure 5.2. D'après notre étude quantitative sur les applications et la volonté de simplifier la complexité du matériel, 24 instructions ont été choisies (le tableau 5.1). Toutes les instructions, sauf MUL, nécessitent un cycle d'horloge pour s'exécuter dans l'étage d'exécution.

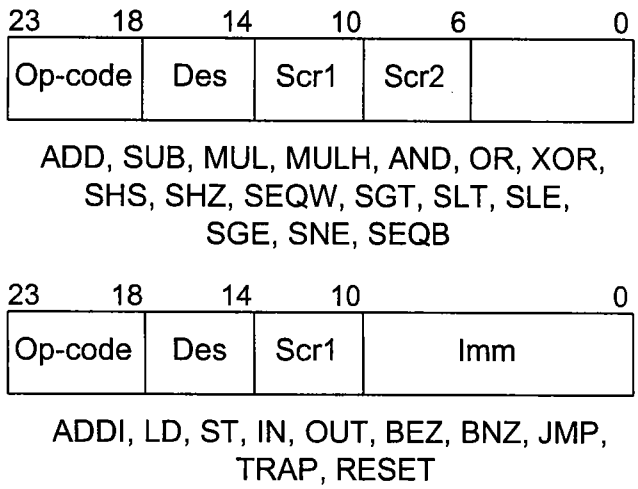


FIG. 5.2: Formats d'instructions

Les instructions peuvent employer quatre modes d'adressage. Dans le premier format d'instruction, l'adressage registre est utilisé lorsque les champs *Scr1*, *Scr2* et *Des* désignent les registres contenant les données. Dans le deuxième format, si *Scr1* égale 0 (indiquant le registre R0, dont le contenu est toujours égal à 0) et *imm* contient l'opérande, alors le mode adressage immédiat est réalisé. Dans l'instruction JUMP, l'adresse du saut est représentée par la somme du contenu du registre désigné par *Scr1* et de *imm*. Par conséquent, il s'agit du mode d'adressage direct. Dans les instructions LD et ST, le contenu du registre désigné par le champ *Scr1* représente l'adresse de l'opérande : il s'agit du mode d'adressage indirect. Le tableau 5.1 présente le jeu d'instructions.

Dans les instructions de décalage, la valeur de la deuxième opérande indique le nombre de décalages. Quel que soit le nombre de bits, le décalage se fait en un cycle d'horloge. Ainsi, le signe '-' ou '+' de la deuxième opérande indique un décalage à droite ou à gauche, respectivement. En cas d'interruption, matérielle ou logicielle, une instruction TRAP déroute l'exécution des instructions vers une adresse spéciale.

Instruction	Opération
ADD	addition
ADDI	addition immédiate
MUL	multiplication and return LSW
MULH	multiplication and return MSW
SHZ	shift in zero
SHS	shift in sign
SUB	subtract
AND	bitwise AND
OR	bitwise OR
XOR	bitwise XOR
LD	load from data memory
ST	store in data memory
OUT	put data on OUT port
IN	get data from IN port
BEZ	branch if zero
BNZ	branch if not zero
JMP	jump
SLT	set if less than
SGT	set if great than
SLE	set if less or equal
SGE	set if great or equal
SEQW	set if word equal
SEQB	set if byte equal
SNE	set if not equal
TRAP	jump to interrupt vector
RST	reset

TAB. 5.1: Instructions SPU-RISC

5.2.3 Banc de registres

Le banc de registres contient 16 registres de 32 bits. La figure 5.3 présente le schéma global du banc de registres. Lorsque les numéros de registres sont présenté aux entrées *rs1* et *rs2*, les contenus des registres correspondants seront disponibles aux sorties *val1* et *val2*, le comportement comparable à celui d'un circuit combinatoire. Par contre, la valeur à écrire (*val_écr*) sera enregistrée dans le registre indiqué par *rd* lors du prochain front montant d'horloge.

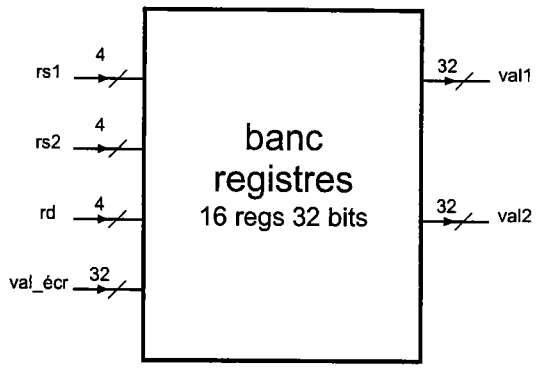


FIG. 5.3: Banc de registres

5.2.4 Unité de calcul

L'unité de calcul peut effectuer les trois catégories d'opérations suivantes :

- les opérations arithmétiques et logiques (`ADD, SUB, OR, AND, XOR, SHZ, SHS`)
- les opérations de comparaison (`SGT, SLT, SEQW, SEQB, SLE, SGE, SNE`)
- l'opération de multiplication

Comme la figure 5.4 le montre, cette unité est composé de trois parties : l'unité arithmétique et logique, l'unité de comparaison et l'unité de multiplication.

Toutes les opérations prennent un cycle d'horloge pour s'exécuter, sauf la multiplication qui prends 33 cycles d'horloge. L'algorithme de Booth a été utilisé pour effectuer la multiplication. On pouvait utiliser d'autres architectures pour faire la multiplication en un cycle d'horloge, mais cela aurait en pour conséquences un accroissement de la surface d'implantation et une diminution de la fréquence maximale de l'horloge, ce qui ne se justifierait que si le nombre de multiplications dans les algorithmes était très élevé.

Les deux lignes d'entrées *In1* et *In2* représentent les opérandes des opérations. Les signaux de contrôles *op*, *Cmp* et *Mult* indiquent les types d'opérations qui seront effectuées sur les opérandes. La ligne d'entrée *Mul-res* indique, dans le cas de la multiplication, quelle partie du résultat de la multiplication (la moitié de poids faible ou la moitié de poids fort) sera sélectionneé en sortie.

5.2.5 Gestion du pipeline et aléas de données

Dans le pipeline, surviennent des situations, appelées aléas, qui empêchent l'exécution au cycle d'horloge prévu de l'instruction suivante dans une séquences d'instructions. Les aléas peuvent obliger de suspendre le fonctionnement du pipeline. Pour éviter les suspen-

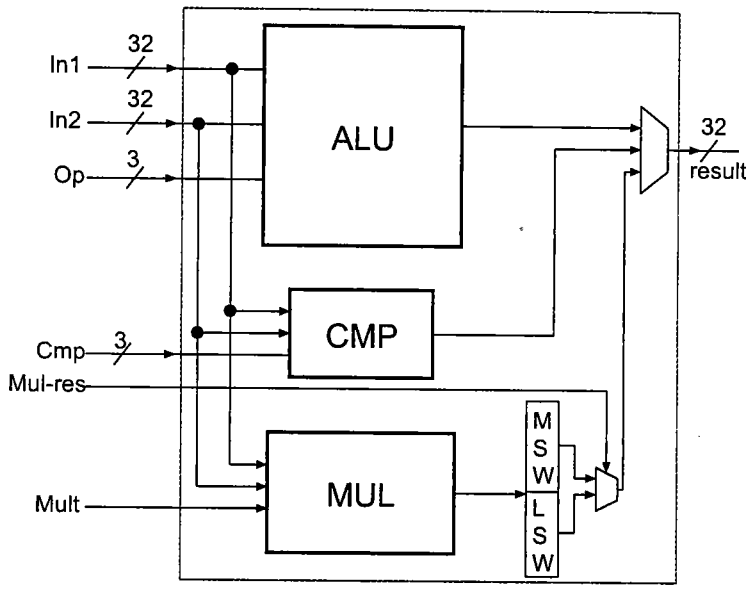


FIG. 5.4: Unité de calcul

sions inutiles, nous avons utilisé la technique de dérivation (*Bypass*) dans l'architecture du pipeline. Par cette méthode, un résultat est passé directement à l'unité fonctionnelle qui en a besoin. Il subsiste toutefois des aléas pour lesquels la suspension est obligatoire.

Comme la figure 5.1 le montre, il y a trois chemins de dérivation (Bypass) dans notre architecture et la logique de dérivation vérifie le remplissage des conditions de dérivation. Dans deux cas, l'architecture utilise obligatoirement la suspension du pipeline :

- La suspension du pipeline dans l'étage de décodage, quand l'évaluation des instructions de branchement exige des informations qui ne sont encore disponible. Dans ce cas, une instruction NOP est ajoutée après l'instruction de branchement.
- La suspension du pipeline dans l'étage d'exécution, quand l'instruction suivante, une instruction `LD`, exige le résultat de `LD`.

5.2.6 Implantation sur FPGA

L'architecture SPU-RISC a été décrite en VHDL au niveau RTL et validé sur FPGA. Nous avons utilisé Synplify-Pro pour synthétiser ce processeur. Une implantation sur FPGA de type virtex2 XC2V250 de Xilinx a été effectuée. Selon les outils de conception, le SPU-RISC fonctionne correctement avec une fréquence d'horloge d'environ 75 MHz et exige 1787 LUT pour être implanté. Les résultats de la simulation présentent un CPI moyen (Cycle d'horloge Par Instruction) de 1.2 dans l'exécution de nos benchmarks. En d'autres termes, il peut atteindre jusqu'à 62.5 MIPS (*Million Instructions Per Second*). Ces résultats ont été obtenus en acceptant les suspensions pour les deux cas mentionnés dans la section précédente. Nous avons essayé de diminuer le nombre de suspensions en ajoutant des chemins de dérivation, mais cette approche pose des problèmes de chemin critique en abaissant qui baisse fortement la fréquence maximale d'horloge. Par exemple, en ajoutant un chemin de dérivation entre l'étage d'exécution et de décodage, pour annuler la suspension due à l'instruction «branch», la fréquence baisse jusqu'à 50 MHz. Le tableau 5.2 montre plus de détails concernant l'implantation du SPU-RISC.

Type FPGA	Ressources logiques utilisées	La fréquence maximum
ALTERA FLEX EPF10K70	3151 (LUTs) (84%)	12.9 MHz
Xilinx Virtex2 XC2V250	1787 (LUTs) (58%)	76.7 MHz
ALTERA MERCURY EP1M120	2469 (ATOMs) (51%)	81.2 MHz

TAB. 5.2: L'implantation de SPU-RISC

5.3 SPU-DSP

5.3.1 Chemin de données

Un processeur DSP-RISC de 16 bits a été développé comme SPU de traitement des tâches orientés DSP. Bien qu'une architecture DSP puisse être plus performante pour prendre en charge les tâches de traitement du signal, le choix effectué permet de diminuer la complexité du système global en réduisant l'hétérogénéité architecturale et en simplifiant la réalisation des compilateurs.

Comme la figure 5.5 le montre, il existe une grande ressemblance entre le chemin de données de ce SPU et celui du SPU-RISC. En effet, par l'ajoute de quelques blocs indispensables, une architecture RISC a été convertie en une architecture ayant les capacités générales d'un processeur DSP. Au niveau de la structure du pipeline, il n'existe pas de grande différence entre les deux architectures. Deux registres pipelines, l'accumulateur (Ac) et l'adresse (Adr), sont ajoutés entre l'étage de décodage et celui d'exécution. De même, sont ajouté entre les étages exécution, mémoire et écriture les registres contenant le résultat de l'opération MAC (*Multiply-ACcumulate*).

La longueur des instructions, dans ce SPU, est de 32 bits et la longueur du champ immédiat (dans les instructions immédiates) est de 16 bits. Pour cette raison, dans l'étage de décodage il n'existe pas d'unité Es (extension du signe).

Dans l'étage d'exécution, en cas d'instruction MAC, le résultat est transféré aux étages suivants du pipeline par deux registres intermédiaires *Res (16 bits)* et *MAC (24 bits)*. Les opérations arithmétiques, logiques, MAC et décalage de 32 bits sur le contenu du registre Acc (accumulateur), sont exécutées dans l'étage d'exécution, par l'unité de calcul.

Le banc de registres et l'unité de calcul sont assez différents entre les deux architectures. Ces deux blocs sont décrits dans les sections suivantes, après l'explication du format des instructions.

5.3.2 Format des instructions

L'architecture SPU-DSP utilise un format d'instructions de 32 bits. Les instructions sont regroupées en deux formats principaux, comme la figure 5.6 le montre. Par rapport à l'architecture SPU-RISC, le nombre d'instructions de comparaison a été réduit, mais certaines instructions spécialisées DSP, comme `MAC` (Multiply-ACcumulate), ou des instructions avec des modes adressages spécialisés, sont ajoutées. Toutes les instructions, sauf `MUL` et `MAC`, prennent un cycle d'horloge pour s'exécuter dans l'étage d'exécution.

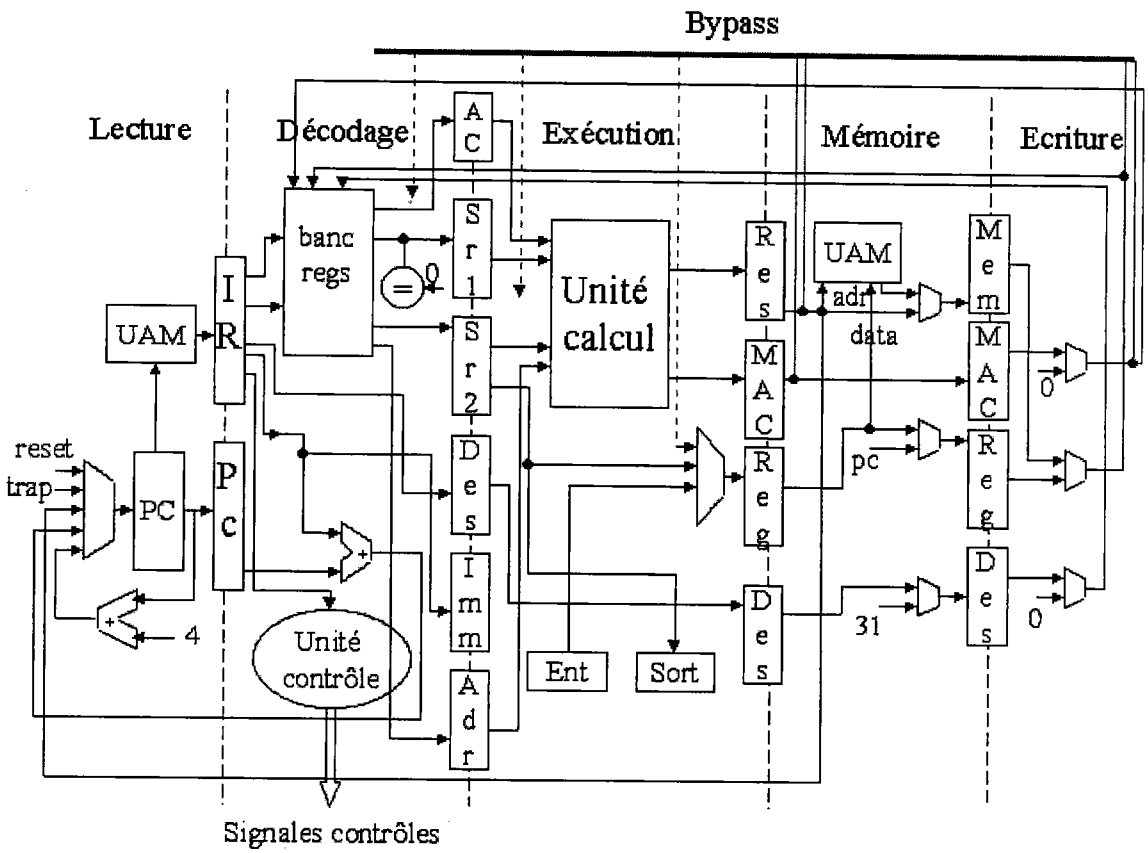


FIG. 5.5: Chemin de données SPU-DSP

L'instruction MAC prend deux cycles d'horloges pour s'exécuter en mode pipeline.

En plus des modes d'adressage de l'architecture RISC pure décrits dans les sections précédentes, cette architecture utilise trois modes spécifiques :

1. l'auto-incrémentation et décrémentation de pointeurs
2. l'adressage modulo
3. l'adressage bit-inverse

Les deux premiers sont utilisés dans l'exécution d'algorithmes, tels que le filtrage FIR. Le troisième est un type d'adressage particulier utilisé dans l'algorithme de « transformée de Fourier rapide (FFT) » qui manipule les échantillons dans un ordre différent de l'ordre séquentiel. Le tableau 5.3 montre le jeu d'instructions.

En plus des instructions de décalage sur les registres généraux, l'instruction SHA peut décaler le registre accumulateur jusqu'à 32 bits en un cycle d'horloge. Ainsi, les opérations de comparaison, le changement du PC (*program counter*) et l'incrémentation du compteur sont effectués en un cycle d'horloge par l'instruction LOOP. En cas d'interruption matérielle, ou logicielle, une instruction TRAP déroute l'exécution des instructions vers une adresse spéciale d'interruption.

Instruction	Opération
ADD	addition
ADDI	addition immediate
MUL	multiplication and return LSW
MULH	multiplication and return MSW
SHZ	shift in zero
SHS	shift in sign
SUB	subtract
AND	bitwise AND
OR	bitwise OR
XOR	bitwise XOR
LD	load from data memory
ST	store in data memory
OUT	put data on OUT port
IN	get data from IN port
BEZ	branch if zero
BNZ	branch if not zero
JMP	jump
SLT	set if less than
SGT	set if great than
SEQ	set if word equal
MAC	multiply and accumulate
SHA	shift accumulate register to 32 bits
LOOP	loop
LDID	load with increment or decrement address
LDBR	load with bit-reversed addressing
LDMD	load with modulo addressing
STID	store with increment or decrement address
TRAP	jump to interrupt vector
RST	reset

TAB. 5.3: Instructions du SPU-DSP

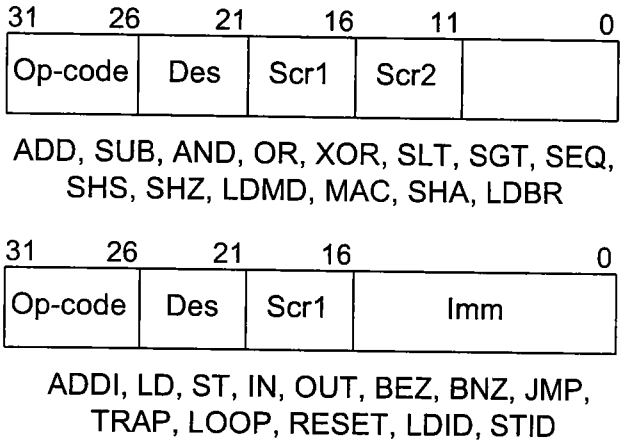


FIG. 5.6: Format d'instruction

5.3.3 Banc de registres

Le banc de registres contient 32 registres de 16 bits. Par rapport à l'architecture SPU-RISC, le nombre de registres a été augmenté à cause de l'existence d'opérations plus longues dans les algorithmes DSP. Les registres R28, R29 et R30 fonctionnent à la fois comme accumulateur de 40 bits et comme registres généraux. Ainsi, le registre R27 dans le mode adressage modulo contient l'adresse de début de table.

La figure 5.7 représente le schéma global du banc de registres. Lorsque les numéros de registres sont présentés aux entrées *rs1* et *rs2*, les contenus des registres correspondants seront disponibles aux sorties *val1* et *val2*, le comportement étant comparable à celui d'nu circuit combinatoire. Par contre, la valeur à écrire (*val_écr*) sera enregistrée dans le registre indiqué par *rd* lors de prochain front montant d'horloge. Ainsi, la valeur à écrire dans l'accumulateur sera sauvegardée dans les registres R28, R29, et R30, au premier front montant du cycle d'horloge à condition que l'entré *mac* soit au niveau logique «1». Un niveau logique « 1 » à l'entrée adr_mod signifie que les bits 0 à 20 de l'entrée *Acc_écr* contiennent la valeur du pointeur et le nombre de registres contenant cette valeur dans les instructions avec adressage auto-incrémenté ou auto-décrémenté.

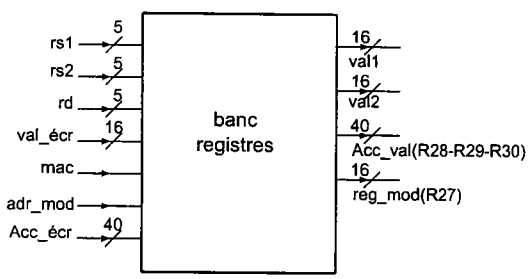


FIG. 5.7: Banc de registres

5.3.4 Unité de calcul

Comme la figure 5.8 le montre, cette unité est composée de cinq parties principales : l'unité arithmétique et logique, l'unité de comparaison, l'unité MAC, l'unité shift-32 et l'unité de génération d'adresses : AGU (*Address Generation Unit*).

Les opérations arithmétique et logique et la comparaison sont effectuées par les unités ALU et CMP. L'unité MAC s'occupe des opérations de multiplication et MAC. Les deux lignes d'entrée *In1* et *In2* représentent les deux opérandes des opérations. Les signaux de contrôle *Op*, *Cmp*, *Mac* et *Shift* indiquent les types d'opérations qui seront effectuées sur les opérandes.

La sortie *result* représente le résultat de toutes les opérations et les seize bits de poids le plus faibles du résultat du MAC et de la multiplication. La deuxième partie des résultats (24 bits de 40 bits) des opérations MAC et multiplication est à la sortie Mac-result. Toutes les opérations prennent un cycle d'horloge pour s'exécuter sauf `mac` qui prend 2 cycles d'horloge (1 cycle pour la multiplication et 1 cycle pour l'addition).

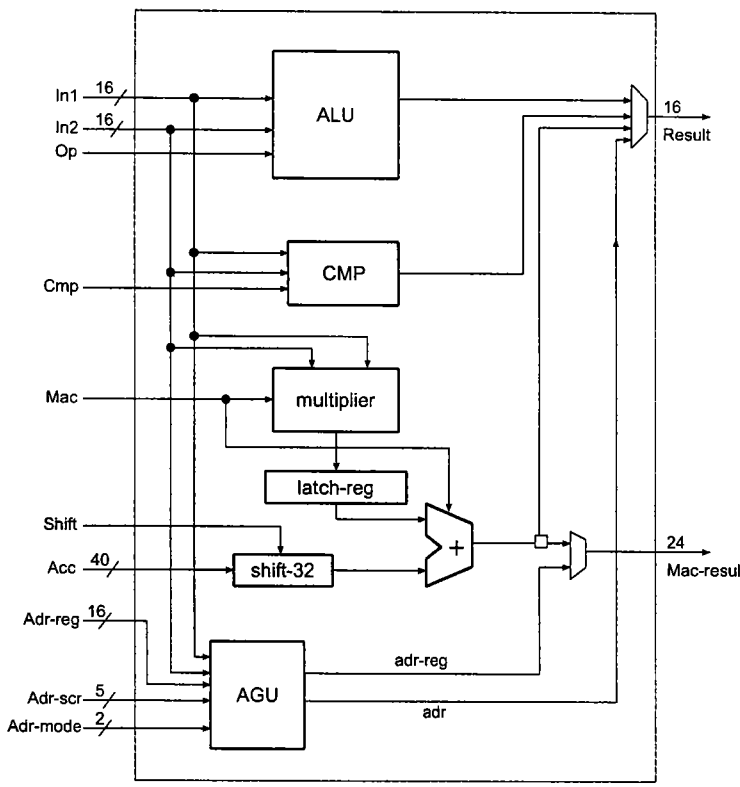


FIG. 5.8: Unité calcul SPU-DSP

L'unité AGU s'occupe de calculer l'adresse d'accès à la mémoire de données selon différentes modes d'adressage. Cette unité calcule l'adresse effective à partir du des informations présentées Adr-reg et Adr-scr, respectivement, le contenu et le numéro du registre d'adresse.

5.3.5 Gestion du pipeline et aléas de données

Comme la figure 5.5 le montre, deux chemins de dérivation ont été ajoutés par rapport à l'architecture RISC. Ces chemins transférent les résultats des instructions `MAC` et `SHA` dans les étages mémoire et écriture à l'étage d'exécution.

Comme pour RISC, cette architecture utilise obligatoirement la suspension du pipeline dans deux cas : quand l'évaluation des instructions branchement exige des informations qui ne sont pas encore disponible, et quand l'instruction suivante une instruction `LD` exige le résultat de `LD`.

5.3.6 Implantation sur FPGA

L'architecture SPU-DSP a été décrite en VHDL au niveau RTL et validée sur FPGA. Nous avons utilisé Synplify-Pro pour synthétiser ce processeur. Une implantation sur FPGA de type Virtex2 (XC2V250) de Xilinx a été effectuée. Selon les outils de conception, le SPU-DSP fonctionne correctement avec une fréquence d'horloge d'environ 66 MHz et exige 2887 LUT pour être implanté. L'architecture a été simulée et testée sur certains algorithmes de traitement du signal. Ces résultats ont été obtenus en acceptant les suspensions pour les deux cas mentionnés dans la section précédente. Le tableau 5.4 montre plus de détails concernant cette implantation.

Type FPGA	Ressources logiques utilisées	La fréquence maximum
ALTERA FLEX EPF10K100	4008 (LUTs) (80%)	11.7 MHz
Xilinx Virtex2 XC2V250	2887 (LUTs) (93%)	66.8 MHz
ALTERA MERCURY EP1M120	3692 (ATOMs) (76%)	68.1 MHz

TAB. 5.4: L'implantation de SPU-DSP

5.4 Conclusion

Dans ce chapitre, nous avons présenté l'implantation de différents SPU de type SPU-logiciel (ou petits cœurs de processeurs). Sur la base d'une étude quantitative (chapitre 3), une architecture RISC a été conçue et validée sur FPGA. Un jeu d'instructions minimal et essentiel pour exécuter les tâches communes dans le traitement de protocoles a été choisi. Les résultats de simulation confirment la capacité de ce cœur de prendre en charge les tâches générales dans le traitement des protocoles. Ainsi, les résultats de synthèse au niveau de la surface occupée, confirment que l'implantation d'une architecture multi-processeurs avec plusieurs cœurs minimaux et optimaux est tout à fait possible avec une technologie moins chère.

Pour la prise en charge des tâches orientées DSP (le traitement du signal), une architecture DSP-RISC a été conçue et validée sur FPGA. En effet, en gardant les caractéristiques principales d'une architecture RISC, nous avons ajouté les instructions essentielles du traitement du signal et les modes d'adressages spéciaux correspondantes. Ainsi, avec

cette architecture (DSP-RISC), nous avons évité l'augmentation de l'hétérogénéité du système global, susceptible de compliquer la partie logicielle (au niveau du compilateur) du système.

Conclusion générale

Le travail de cette thèse s'intègre dans un projet général au sein du laboratoire LICM concernant la conception architecturale d'une chaîne de transmission à haut débit. L'objectif final est de développer une architecture de processeur spécialisée pour traiter les algorithmes des protocoles de réseaux dans les terminaux utilisateurs. Cette architecture peut servir comme cœur de traitement dans des équipements terminaux tels que les modems ou les cartes réseau.

Aujourd'hui, trois caractéristiques principales dans le secteur des télécommunications sont à prendre en considération : la diversité des protocoles employés, l'hétérogénéité des données et le haut débit. La réalisation d'un système remplissant ces conditions exige une architecture flexible dotée d'une puissance de traitement élevée. Pour développer de tels systèmes, la plupart des recherches à la fois académiques et industrielles se sont concentrées sur les équipements au cœur des réseaux comme les routeurs et les commutateurs intitulés *Network Processors*, mais les équipements terminaux exigent également des architectures performantes. Pour cela, nous avons proposé une méthodologie pour concevoir une architecture de processeur pouvant être utilisée comme processeur de traitement des protocoles de couches basses dans les équipements terminaux.

Cette thèse a été structurée en deux parties : la première partie concernant l'état de l'art sur le sujet est divisée en deux chapitres.

Dans le chapitre 1, les réseaux sont introduits de manière générale. Les trois différents modèles de réseaux sont présentés et leurs principes sont expliqués. Ensuite, les différents types de réseaux sont introduits en insistant particulièrement sur les réseaux haut débit. L'étude des protocoles et de leurs fonctions montre qu'il existe des opérations fréquentes ou ayant peu de différence entre elles dans les différents protocoles. L'identification de ces opérations peut tracer le chemin de recherche pour mettre au point des architectures spécifiques pour le traitement des protocoles. Il existe deux catégories principales dans le traitement des protocoles : le traitement de protocoles de type contrôle et le traitement de protocoles de données. Chaque type d'opération a ses caractéristiques propres. En continuation de ce chapitre, les différentes méthodes de parallélisme sont introduites et leurs avantages et inconvénients sont discutés. Un compromis entre les surcoûts de chaque méthode (comme la communication entre les processeurs et la synchronisation) et le facteur d'utilisation des ressources nous permet de choisir une combinaison de ces méthodes pour atteindre les objectifs fixés.

Le chapitre 2 introduit brièvement les différentes architectures de processeurs. Trois grandes catégories d'architectures sont présentées. Dans la première partie les deux architectures scalaires générales, CISC et RISC, sont étudiées de manière comparative. Dans la deuxième partie nous présentons les méthodes architecturales parallèles. Certaines méthodes ,comme le MIMD et le parallélisme d'instructions, sont plus utilisées et plus per-

formantes dans les domaines d'application généraux, alors que d'autres, comme le SIMD et le parallélisme de données, sont intéressants pour des applications spécifiques telles que le traitement numérique du signal. Enfin, les caractéristiques principales de deux architectures de processeurs sont introduites, correspondant aux applications réseaux, incluant les DSPs et les processeurs de réseau.

Dans la deuxième partie, qui est divisée en trois chapitres, nous introduisons le thème principal de ce travail.

Dans le chapitre 3, nous proposons une méthodologie pour vérifier la performance de diverses architectures pour implanter les protocoles de réseaux haut débit. Tout d'abord, nous étudions les opérations requises pour traiter ces protocoles. Par la suite, nous proposons une méthodologie pour évaluer les performances des architectures pour exécuter ces tâches. Dans cette méthodologie, un compilateur produisant du code virtuel pour un processeur virtuel est développé pour les algorithmes décrits en langage C. Ce code virtuel est ensuite reciblé sur les différentes architectures. Nous utilisons une modélisation à l'aide de chaînes de Markov. À partir des mesures de performance, et à l'aide des méthodes d'analyse multi-dimensionnelle de données, on peut avoir une bonne vision pour choisir les architectures adaptées. Les résultats d'analyse des mesures montrent que les deux architectures RISC et DSP sont les bonnes candidates comme cœurs de processeur pour exécuter les algorithmes représentatifs généraux et DSP, respectivement. Les résultats indiquent que cette méthode peut également être utilisée comme une méthode de répartition des algorithmes entre les différentes architectures.

Le chapitre 4, présente le modèle architectural proposé pour un processeur de traitement de protocoles de réseaux. L'idée principale est l'utilisation d'un modèle multiprocesseurs basé sur de petits processeurs spécifiques. L'architecture proposée, en considérant les caractéristiques du trafic de réseau, permet d'avoir une bonne performance. En effet, ce modèle est basé sur deux notions principales. La première est que si on partitionne les protocoles en fonctions indépendantes, tous les paquets de données ne seront évidemment pas traités par toutes les fonctions. En conséquence, les paquets peuvent être traités juste par les fonctions nécessaires de manière dynamique. La deuxième notion, est l'utilisation du temps de silence entre les trains de paquets. Ce temps permet d'utiliser une architecture pipeline non-linéaire qui introduit une latence additionnelle pour faire circuler les données parmi les étages du pipeline mais avec moins d'exigence au niveau du matériel. Les résultats de la simulation montrent que le modèle est performant, surtout pour le trafic auto-similaire avec une faible charge. Ainsi, ces résultats indiquent que pour le trafic en temps réel et de type flux (*stream*) avec une charge lourde, il faudra chercher des modèles parallèles adaptés. Les résultats de la simulation indiquent également que le partitionnement des protocoles en fonctions et leur allocations aux SPU est très importante pour avoir une bonne performance.

Dans le chapitre 5, nous présentons l'implantation de SPU de type petits cœurs de processeurs. Basée sur l'étude quantitative du chapitre 3, une architecture RISC est conçue et validée sur FPGA. Un jeu d'instructions minimal et essentiel pour faire les tâches communes dans le traitement de protocoles est choisi. Les résultats de simulation confirment la capacité de ce cœur pour effectuer les tâches générales dans le traitement des protocoles. Pour effectuer les tâches orientées DSP, ou bien le traitement du signal, une architecture DSP-RISC est conçue. En effet, en gardant les caractéristiques principales d'une architecture RISC, nous ajoutons les instructions essentielles au traitement du signal et les modes d'adressages spéciaux correspondants. Ainsi, avec cette architecture (DSP-RISC), nous évitons l'augmentation de l'hétérogénéité du système global susceptible de compliquer la partie logicielle du système.

En ce qui concerne les perspectives de ce travail, à court terme, on peut définir trois axes principaux :

- L'étude et le développement d'un réseau d'interconnexion approprié, reliant les SPU et le MU, peut être un premier axe. En effet, aujourd'hui, la communication inter-puces dans les systèmes multiprocesseurs sur une puce est un aspect très important qui peut limiter la performance globale du système. En considérant différents SPU et les unités dédiées ou ADM, un réseau de communication hétérogène peut être utilisé.

- Les SPU doivent échanger le code correspondant à leurs tâches entre les mémoires locales et la mémoire principale. Pour cela, l'organisation de la mémoire et les méthodes d'accès à la mémoire est un autre axe principal qui doit faire l'object d'une attention particulière dans la suite de ce travail.

- Dans ce modèle, le contrôle de la circulation des données est fait par les SPU. Après le traitement d'un paquet, chaque SPU peut déterminer la destination du paquet selon un schéma de partition global du programme préparé par le MPU. Dans ce cas, c'est la dépendance logique entre les différentes étapes du traitement qui précise le plan de circulation des paquets entre les SPU. Tandis que certains SPU peuvent être saturés, la question d'équilibrage de charge entre SPU, en utilisant différentes méthodes, peut être un autre objectif de l'étude.

La suite logique de ce travail à plus long terme est la validation globale de notre modèle par la réalisation matérielle d'un processeur. Il est nécessaire d'implanter la phase de réalisation matérielle en la rendant la plus automatique possible. De cette façon, une architecture paramètrable pourra être réalisée, facilement adaptable, aux différentes évolutions des applications.

Bibliographie

Bibliographie

[Abnous95] A. Abnous, N. Bagherzadeh, "Architectural Design and Analysis of a VLIW Processor", Computer&Electronic Engineering, vol. 21, no. 2, 1995.

[Achenden98] P.J. Achenden, The student's Guide to VHDL, Morgan Kaufman, 1998.

[AhleHagh04] H. AhleHagh, W.R. Michalson, "Statistical Characteristics of Wireless Network Traffic and its Impact on Ad Hoc Network Performance", prceeding of conference in applied telecommunication symposium, Arlington Virginia, April, 2004.

[Art03] F.Arts et al.,"Network processor requirements and benchmarking", Computer Networks, vol. 41, 2003, pp. 549-562.

[ARCHC] S.Rigo et al., "ArchC : A SystemC-based Architecture Description Language", http ://www.archc.org, June 2003.

[Bagnordi97] H. Bagnordi, "Available Instruction Level Parallelism in Multimedia applications", proc. of ICSPAT SanDiego, USA, 1997.

[Bajot01] Y.Bajot, étude et spécification d'un coeur de DSP configurable, thése PhD,université Paris VI, 2001.

[Benini02] L. Benini, G.D. Micheli, "Network on Chips : A New Soc Paradigm", IEEE Computer magazine, pp. 70-78, January 2002.

[Berkeley00] Berkeley Design Thecnology Inc., "The evolution of DSP processors ",April 2000, http ://www.bdti.com.

[Bhugra04] H. Bhugra, "LA-1 : Standardizing the Look-Aside Processor Interface", CommsDesign, on the www : http :www.commsdesign.com/story/OEG20020917S0022.

[Björkman98] M. Björkman and Per Gunningberg, "Performance Modeling of Multiprocessor Implementations of Protocols", IEEE/ACM Transactions on Networking, vol. 6, No. 3, pp. 262-273, June 1998.

[Brenner97] P. Brenner, "A technical tutorial on the IEEE 802.11 protocol" , Breezecom wireless communications, 1997.

[Busby00] M. Busby, Introduction à TCP/IP, Traduit en français par G. Otman, OSMAN EYROLLES MULTIMEDIA, 2000.

[Bux01] W. Bux, W.Denzel, T.Engbersen, A.Herkensdorf, R.Luijten, "Technologies and building blocks for fast packet forwarding", IEEE Communications Magazine, January 2001.

[Cesario02] W. Cesario, A. Baghdadi, "Component Based Design Approach for Multicore SoCs", proc. of Conference in Design Automation and Test, New Orleans, Louisiana, USA, June 2002.

[Chiueh99] T.C. Chiueh, P. Pradhan, "Cache Memory Design for Network Processors", Proc. Sixth International Symposium on High Perfromance computer architecture, vol. HPCA-6, 1999.

[Cho03] Y. Cho, G. Lee,et al, "Scheduling and Timing Analysis of HW/SW On-Chip communication in MP system SoC Design", Proc. of Conference in Design Automation and Test in Europe, Germany, March 2003.

[Comer00] D.E. Comer, "TCP/IP Architecture, Protocoles, Applications", Traduit en français par J.A. Hernandez et al., 3^{e} édition, DUNOD, 2000.

[Comer04] D.E. Comer, Network Systems Design using Network Processors, Pearson Education INC., 2004.

[Elkateeb00] A. Elkateeb, M. Elbeshti, "A Study Using a RISC Core for ATM Network Interface Design", Computer Communication, no. 23, 2000.

[Ewert01] P. M. Ewert, N. Manjikian, "Hardware/sofware tardeoffs for IP-over-ATM frame reassembly in an integrated architecture", Elsvier Computer Communications, vol. 24, 2001, pp. 768-780.

[Fisher96] J.Fisher, P.Faraboschi, "Custom Fit Processors", proc. of 30th annual international symposium on micro architecture, Paris, France, 1996.

[Gloria99] A. Gloria, "Microprocessor Design for Embedded Systems", Journal of Systems Architecture, no. 45, 1999.

[Grass01] E. Grass et al., "On the single-chip implementation of a HiperLan/2 and IEEE 802.11a capable modem", IEEE Personal Communications magazine, PP. 48-57, December 2001.

[Hennessy96] D.A. Patterson, J.L. Hennessy, Computer organization and design, Morgan Kaufmann, 1996.

[Hobson99] R.F. Hobson, P.S. Wong, "A Parallel Embedded Processor Architecture for ATM Reassembly", IEEE/ACM Transactions on Networking, Vol. 7, No. 1, February 1999, pp. 23-37.

[Hwang93] K. Hwang, Advanced Computer Architecture : Parallelism, Scalability, Programmability, McGraw Hill, 1993.

[IEEE802.11] IEEE Std 802.11-1997, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. The Institute of Electrical and Electronics Engineers, NJ, USA 1999.

[Intel01] Product Brief, "Intel IXP2400 Network processor for 2.5 Gbps network access", http ://www.intel.com.

[ISC] Internet Software Consortium, http ://www.isc.org.

[Jones03] G. Jones, E. Stipidis, "Architecture and Instruction Set Design of an ATM Network Processor", Microprocessors and Microsystems, No. 27, 2003.

[Jouppi89] N. Jouppi, D.W. Wall, "Available Instruction Level Parallelism for Superscalar and superpipelined machines", 3rd IEEE International Conference on Architecturale Support for Programming Languages, Boston, USA, 1989.

[Kaiserwerth93] M. Kaiserwerth, "The parallel protocol engine," IEEE/ACM Trans. Networking, vol. 1, Dec. 1993.

[Kode01] S. Kode,J. Maheswary, "Traffic Characterization for Heterogeneous Applications", report in www.ee.vt.edu/ Idasilva/6504, 2001.

[Kofman99] D. Kofman, B. Jabbari, Réseaux Haut Débit : Réseaux ATM et Locaux, Paris Dunod, 1999.

[Kouks02] A.K. Kouks, A.F. Evagelatos, "Microprocessor Adaptor for ATM Networks", Microprocessors and Microsystems Journal 26, 2002.

[Kramer97] G. Kramer, "On generating self similar traffic using pseudo-Pareto distribution", technical brief raport, wwwcsif.cs.ucdavic.edu/Kramer/papers.

[Kumar02] S. Kumar, et. al, "A Network on Chip Architecture and Design Methodology", IEEE Computer Society Annual Symposium on VLSI, Pittsburgh,Pennsylvania, USA, April 2002.

[Lampret01] D. Lampret, "Open RISC1200 IP core specifications", www.opencores.org, 2001.

[Leland94] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic(Extended Version)", IEEE/ACM Transactions on Networking, Feb. 1994.

[Lenell94] N. Lenell, N. Bagherzadeh, "A Performance Comparison of Several Superscalar Processor Models with a VLIW Processor", Microprocessor Microsystems, vol. 18, no. 3, 1994.

[Lucent] LUCENT TECHNOLOGIES, DSP16210 Datasheet, http ://www.lucent.com/micro/dsp16000.

[M'sir03] M.A. M'sir, Conception d'Architectures Rapides pour Codes Convolutifs en Télécommunications : Application aux Turbo-codes, Manuscrit de thèse, LICM, Université de Metz, France, 1999.

[Memik01] G. Memik, W.H. Mangione-smith, W. Hu, "NetBench : A Benchmarking Suite for Network Processors", Proceeding of International Conference on Computer Aided Design, San Jose, CA, Nov. 2001.

[Motorola98] Motorola semiconductor technical data,"24 bit general purpose digital signal processor",1998.

[Nicopolitidis02] P. Nicopolitidis, G.I. Papadimitriou, "A high performance protocol for wireless lacal networks", computer communications journal, vol. 25, 2002.

[Patterson96] D.A. Patterson, J.L. Hennessy, Computer Architecture : a Quantitative Approach, Morgan Kaufmann, 1996.

[Peyravian03] M. Peyravian, J. Calvignac, "Fundamental Architectural Considerations for Network processors", Computer Network Journal 41, 2003, pp. 587-600.

[Philip99] S. Philip, Etude et Développement d'une Nouvelle Architecture de Processeur DSP Dédiée aux Applications Modem en Télécommunications sur Câble TV, Manuscrit de thèse, LICM, Université de Metz, France, 1998.

[Pujolle03] G. Pujolle, Les Réseaux, Eyrolles, Paris, 2003.

[[Rabaey00] J. Rabaey et al., "Challenges and Opportunities in Broadband and Wireless Communication Designs", Proceedings of ICCAD, San Jose, Nov 2000.

[Roberts00] L.G. Roberts, "Beyond Moores law : Internet growth trends", IEEE Computer, No.33, January 2000.

[Robin99] P. Robin, Réseaux Haut Débit, Paris Hermès sciences publications, 1999.

[Ross89] S.M. Ross, Introduction to Probability Models, Academic Press, 1989.

[Schaller97] R.R. Schaller, "Moores law : past, present and future", IEEE Spectrum, No.34(6), June 1997.

[Sgroi01] M. Sgroi, et al, "Addressing the System-on-a-Chip Interconnect Woes Through Communication-based Design", 38th Design Automation Conference, June, 2001.

[Silc00] J. Silc, T. Ungerer, "A Survey of New Research Directions in Microprocessors", Journal of Microprocessors and Microsystems, no. 24, 2000.

[Simpson094] W. Simpson, "RFC 1661 The Point-to-Point Protocol", July 1994.

[Simpson194] W. Simpson, "RFC 1662 PPP in HDLC-like Framing", July 1994.

[Sun99] A.Sun, traduction française de H.Sovlard, PPP configuration et mise en oeuvre, édition O'REILLY, 1999.

[Steenkiste92] P. Steenkiste, "Analyzing Communication Latency using the Netcar Communication Processor", ACM SIGCOMM Computer Communication Review, vol. 22, No. 4, pp. 199-209, October 1992.

[Tanenbaum03] A. Tanenbaum, Réseaux, 4th Edition, Pearson Education France, 2003.

[Tell01] E. Tell, A domain specific DSP processor, Master thesis, university of linköping, Swede,2002.

[Tokhi95] A.O. Tokhi, M.A Hossain, "CISC, RISC and DSP Processors in Real Time Signal Processing and Control", Microprocessors and Microsystems, Vol. 19, June 1995.

[Trézéguet02] H.Trézéguet, "Les Processeurs Réseaux pour 1 Gbits/s et plus", Electronique journale, n. 131, Décembre 2002.

[Tuan01] T. Tuan, S. Le, J. Rabaey, "Reconfigurable architecture for Wireless Protocol Processor", proc. of ICASSP2001, Utah, 2001.

[Vachaux02] A. Vachaux, "Modélisation de système intégré numérique", Ecole polytechniques Lausanne, Notes de cours, 2002.

[Vallino99] T. Vallino, "Recherche des Performances dans la mis en œuvres des Codes Linéaires Cycliques en ASIC à Haut Débit", Manuscrit de thèse, LICM, Université de Metz, France, 1999.

[Venkatachalam03] M. Venkatachalam, P. Chandra, R. Yavatkar, "A Highly Flexible Multiprocessor Architecture for Network Processing", Elsevier Computer Network, No. 41, 2003, pp. 563-586.

[Virtanen03] S. Virtanen, et al, "NoC Interface for a Protocol Processor", In Proc. of the 21st IEEE Norchip Conference, Riga, Latvia, 10-11 November 2003.

[Wallace95] S. Wallace, N. Bagherzadeh, "Performance issues of a supersclar microprocessor", Microprocessors and Microsystems, vol. 19, May 1995.

[Willinger97] W. Willinger et all, "Self-similarity Through High Variability : Statistical Analysis of Ethernet LAN Ttraffic at the Source level", IEEE/ACM Transactions on Networking, Vol. 5, No. 1, 1997.

[Wolf00] T. Wolf, M. Franklin, "COMMBENCH : A Telecommunications Benchmarks for Network Processors", IEEE International Symposium on Performance Analysis of Systems and Software, Austin, USA, April 2000.

[XTENSA] "Xtensa-v configurable processor", product specification, http ://www.tensilica.com.

[Yang00] J.Yang,and others,"Metacore :an application-specific programmable DSP Development system",IEEE trans. on VLSI systems,vol 8,no 2,April 2000.

[Zitterbrat91] M. Zitterbrat, "High speed transport components", IEEE Network Magazine, vol. 5, no. 1,1991.

Annexes

Annexe A

Liste des Abréviations

AAL : ATM Adaptation Layer.
ACCM : Asynchronous Control Character Map.
AGU : Address Generation Unit.
AMCC : Applied Micro Circuits Corporation.
AP : Access Point.
ARP : Address Resolution Protocol.
ATM : Asynchronous Transfer Mode.
BCH : Bose Chaudhuri Hocquenghen.
BSS : Basic Service Set.
CDFG : Control Data Flow Graph.
CDMA/CA : Carrier Sense Multiple Access/Collision Avoidance.
CISC : Complex Instruction Set Computer.
CLB : Configurable Logic Block.
CRC : Cyclic Redundancy Check.
CS : Convergence Sub-layer.
CTU : Control Transfer Unit.
DDR : Double Date Rate.
DNS : Domain Name System.
DQDB : Distributed Queue Dual Bus.
DSSS : Direct Sequence Spread Spectrum.
ETSI : European Telecommunication Standard Institute.
FCS : Frame Check Sequence.
FHSS : Frequency Hopping Spread Spectrum.
FPGA : Field Programmable Gate Array.
FSM : Finite State Machine.
HDLC : High level Data Link control.
HEC : Header Error Control.
HiperLAN : High Performance Radio LAN.
HTTP : Hyper Text Transfer Protocol.
IBSS : Independent Basic Service Set.
ICMP : Internet Control Message Protocol.
IFS : Inter-Frame Space.
IGMP : Internet Group Management Protocol.
ILP : Instruction Level Parallelism.
IP : Internet Protocol.
ISO : International Standardisation Organisation.

LAP-B : Link Access Protocol Balanced.
LCP : Link Control Protocol.
LLC : Logical Link Control.
MAC : Multiply-ACcumulate.
MAC : Medium Access Control.
MIMD : Multiple Instruction Multiple Data.
MISD : Multiple Instruction Single Data.
MPU : Master Processing Unit.
NoC : Network on chip.
NP : Network Processor.
NUMA : Non-Uniform Memory Access.
OAM : Operation And Maintenance.
OC : Optical Carrier.
OFDM : Orthogonal Frequency Division multiplexing.
PAN : Personal Area Network.
PM : Physical Medium.
PMD : Physical Medium Dependent.
POS : Packet Over SONET.
PPP : Point-to-point protocol.
QDR : Quad Data Rate.
QoS : Quality of Service.
RARP : Reverse Address Resolution Protocol.
RISC : Reduced Set Instruction Computer.
SAR : Segmentation And Reassemblage.
SDH : Synchronous Digital Hierarchy.
SIMD : Single Instruction Multiple Data.
SISD : Single Instruction Single Data.
SONET : Synchronous Optical Netwok.
SPU : Slave Processing Unit.
STM : Synchronous Transfer Mode.
TC : Transmission Convergence.
TCP : Transmission Control Protocol.
TDM : Time Division Multiplexing.
UDP : User Datagram Protocole.
UMA : Uniform Memory Access.
UNI : User Network Interface.
VLIW : Very Long Instruction Word.
VPI : Virtual Path Identifier.
VCI : Virtual Channel Identifier.
WPAN : Wireless Personal Area Network.

Annexe B

Liste des publications

- A. Ramazani, F. Monteiro, A. Dandache, B. Lepley, " A Methodology to Design Multimedia Processor Cores, 10th IEEE International Conference on Electronics Circuits and Systems " (ICECS'03), Sharjah, United Arab Emirates, December 14 -17, 2003.

- A. Ramazani, C. Diou, F. Monteiro, A. Dandache, " A Novel Processor Architecture for Network Applications ", International Conference on Signals and Electronics Systems (ICSES'04), Poznan, Poland, September 13 - 15, 2004.

- A. Ramazani, F. Monteiro, C. Diou, A. Dandache, "A Multiprocessor Architecture for Fast Packet Processing ", 12th IEEE International Conference on Electronics Circuits and Systems " (ICECS'05), Gammarth, Tunisia, December 11 - 14, 2005.

Résumé :

Le travail de cette thèse s'intègre dans un projet général au sein du laboratoire LICM concernant la conception architecturale d'une chaîne de transmission à haut débit. L'objectif global est de concevoir un processeur spécialisé pour le traitement rapide des algorithmes des divers protocoles présents dans les couches basses des modèles références (OSI, Internet, ITU-T/ATM). L'évolution des technologies et l'élargissement des bandes passantes des réseaux de transmission ont transféré le goulot d'étranglement concernant les débit autorisés vers les équipements constituant les nœuds actifs des réseaux. La prise en charge de la diversité des protocoles employés, de l'hétérogénéité des données et des très forts débits requis, n'est possible que par une forte montée en puissance de la capacité de traitement de ces équipements. Si ce problème est déjà en bonne partie traité en ce qui concerne les routeurs et les commutateurs, beaucoup de chemin reste encore à faire concernant les équipements terminaux de circuits de données (ex: modem, carte réseau) dans le domaine du haut débit.

La conception d'une architecture de processeur spécialisée dépend fortement des caractéristiques des applications auxquelles le processeur est dédié. L'architecture globale choisie pour le processeur est celle d'un ensemble d'unités de traitement généralistes (mini cœurs de processeur) ou spécialisées (modules auxiliaires) interconnectées. Le but est d'offrir une capacité de traitement parallèle élevée. Le développement d'une telle architecture nous impose de définir une démarche méthodologique appropriée. Cette démarche commence par une étude de protocoles de réseaux représentatifs. Le but est tout d'abord d'identifier parmi les principales tâches (opérations) des protocoles, les plus communes et les plus critiques d'entre elles. Les tâches critiques (du point de vue temporel) sont traitées par des modules spécialisés (dont l'étude fait l'objet d'autres travaux). Les tâches restantes sont prises en charge par les unités de traitement généralistes, dont l'étude constitue l'essentiel de ce travail. Les performances potentielles de ces unités généralistes sont évaluées en fonction de différentes architectures cibles (CISC, RISC, superscalaire, VLIW). La technique mise en place, pour l'évaluation des performances temporelles des architectures, repose sur une modélisation des algorithmes par chaînes de Markov. Un banc de simulation a été réalisé implantant la technique. Afin de ne pas favoriser indûment une architecture, nous avons introduit un modèle de processeur virtuel pour coder les algorithmes sans introduire de contrainte lié à l'une des architectures.

L'analyse des résultats obtenus avec le banc de simulation, nous a permis de déterminer les architectures les plus appropriés par type d'algorithme. La performance de l'architecture globale du processeur (fonctionnement parallèle de l'ensemble des unités de traitement) a été évaluée pour différentes conditions de trafic. Un modèle d'interconnexion simplifié (par rapport au modèle final) a été utilisé, reliant les unités de traitement sous forme d'un pseudo-pipeline (linéaire ou non). Enfin, deux types d'unités de traitement généralistes (mini cœurs de processeur) ont été modélisées en VHDL au niveau RTL et validées sur FPGA.

Mots clés : architecture processeur, multiprocesseur, parallélisme, pipeline, protocoles réseaux.

Abstract :

The work presented here is part of a general project within the LICM laboratory, concerning the architectural design of a high data rate transmission system. The main objective is to design a specialized processor for fast processing of the lower layer protocols in the reference models (OSI, Internet, ITU-T/ATM). The technology evolution and increasing of the bandwidth of the physical transmission media have transferred the bottleneck concerning the available data rates of the communication networks towards the network active node equipments. Managing the protocol diversity, the data heterogeneity, and high data rates requires a substantial improvement of the processing power in these equipments. Although this problem has been largely addressed concerning routers and switches, end-user equipments (e.g: modems and network interface cards) are still far from matching the requirement of high data rates.

The design of a specialized processor architecture depends strongly on the characteristics of applications to which the processor is dedicated. The architecture selected for the processor is a set of interconnected general processing units (mini processor cores) or specialized modules (auxiliary modules). The goal is to offer a high level of parallel processing capacity. The development of such an architecture requires a suitable design methodology to be defined, which starts by a study of representative network protocols. First of all, the goal is to identify among the principal tasks (operations of the protocols) the most common and the most critical. The critical tasks (from the realtime point of view) are processed by specialized modules (whose study is not our goal). The remaining tasks are performed by the general processing units, whose study is the main subject of this work. Potential performance of these general units is evaluated for various target architectures (CISC, RISC, superscalair, VLIW). The time performance evaluation of architectures is based on the algorithm modelling using Markov chains which was used to implement the simulation tools. In order not to favour a particular architecture, we introduced a virtual processor model to encode the algorithms without introducing any constrains related to actual architectures.

The analysis of the simulation results allowed us to find good architectures/algorithms adequacy. The performance of the overall processor architecture (parallel operation of the processing units) was evaluated under various traffic conditions. A simplified interconnection model (compared to the final model) was used which allowed to connect the processing units in a pseudo-pipeline (linear or not) chain. Lastly, two types of general processing units (mini processor cores) have been designed in VHDL at the RTL level and validated on FPGA.

Key Words : processor architecture, multiprocessor, parallelism, pipeline, network protocols.