SETIT 2004
International Conference: **S**ciences of **E**lectronic,
Technologies of **I**nformation and **T**elecommunications
March 15-20, 2004 – TUNISIA

Application Télécom Pour Processeur Réseaux

Saifeddine Berrayana, Etienne Faure, Daniela Genius, Frédéric Pétrot

Departement ASIM du LIP6 ; Université ParisVI

{Saifeddine.Berrayana, Etienne.Faure, Daniela.Genius,Frederic.Pétrot}@lip6.fr

Résumé—Nous étudions, dans cet article, les spécifications d'une plateforme matérielle pour exécuter des applications télécom. La description dans le langage de haut niveau SystemC comporte suffisamment de détails pour obtenir des résultats de simulation très précis. Ces résultats indiquent les performances atteintes par notre système et mettent également en relief les points critiques de ce type de plateformes, principalement des problèmes de contentions lors des accès à la mémoire.

Mots Clés: — Processeur réseau, simulation cycle.

1 INTRODUCTION

Les processeurs réseaux (*network processors*) sont des processeurs spécialisés utilisés dans les équipements télécoms. Ces processeurs sont des systèmes intégrés (*System on Chip*) qui possèdent un parallélisme matériel très élévé, puisqu'un processeur réseau peut contenir plusieurs dizaines de coeurs de micro-processeurs. Ces processeurs exécutent des applications logicielles qui possèdent elles-mêmes un fort parallélisme, puisqu'une application peut comporter plusieurs dizaines ou centaines de tâches qui s'exécutent en parallèle.

Notre objectif est de décrire une telle plateforme, composée d'une partie matérielle, ainsi que d'une partie logicielle, dans un langage de haut niveau. Le but que nous poursuivons est double, il s'agit d'une part de montrer qu'un tel système peut facilement être spécifié et implémenté à l'aide du langage SystemC [1], au niveau cycle. SystemC se compose d'une bibliothèque C++ comprennant les classes et méthodes permettant l'instanciation et l'affectation des modèles et des signaux, ainsi qu'un moteur de simulation permettant de faire fonctionner le système ainsi décrit. Notre second objectif est d'étudier en détail le fonctionnement de cette architecture, en nous intéressant particulièrement aux problèmes posés par le partage des ressources.

Cet article va suivre le plan suivant: d'abord nous exposerons le méthode de conception que nous avons employée. Ensuite nous présenterons l'application IPv4, actuellement cible de notre recherche, puis l'architecture matérielle utilisée. Nous reviendrons ensuite sur le portage de l'application sur la plateforme cible, avant d'exposer la méthode de validation employée, ainsi que les résultats obtenus. Nous terminerons par un aperçu des limites et perspectives ouvertes par cette méthode.

2 METHODE DE DESCRIPTION

Une des plateforme de développement pour de telles architectures est la plateforme SteNP [2], développée par STMicroelectronics. Son objet est l'étude d'architecture des processeurs réseau, en se plaçant au niveau de simulation transactionnel [3]. Ce niveau de simulation a l'avantage de réduire considérablement le temps nécessaire à la simulation, en contrepartie d'une certaine perte de précision.
Or, notre but étant de pouvoir quantifier les

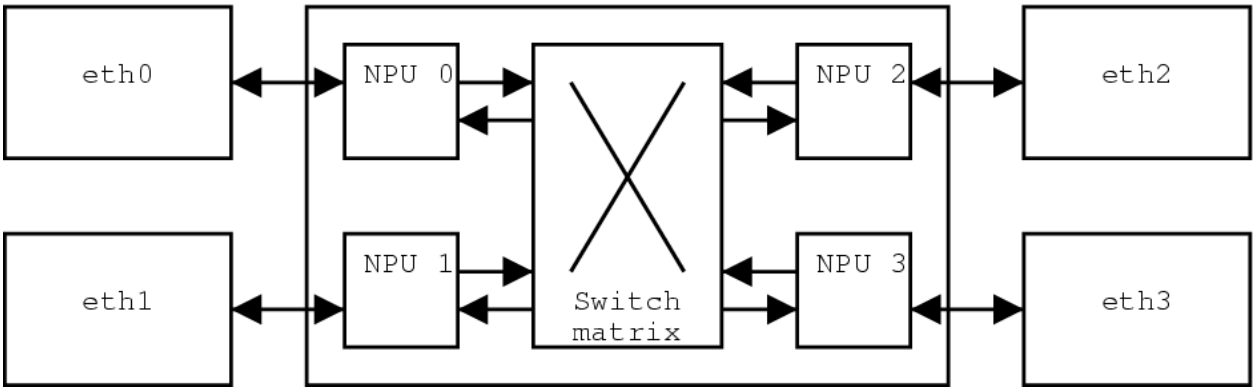

Figure 1: Modèle de routeur à quatre interfaces réseau.

engorgements qui se produisent lorsque plusieurs processeurs requièrent la même ressource, nous souhaitons justement avoir une simulation fine des échanges entre les différents agents. Pour cette raison, nous choisissons un niveau de simulation plus précis, le niveau cycle. Ce niveau de simulation permet, en effet, de savoir, à chaque cycle d'horloge, quelles sont les valeurs présentes sur les connecteurs de chaque composant. Ce niveau d'abstraction est différent du niveau RTL (*Register Transfert Level*) en ce sens que l'on ne cherche pas à représenter fidèlement l'architecture interne du circuit.

Notre système est décrit par une netlist de composants SOCLIB [4]. SOCLIB est une bibliothèque de modèles de simulation CABA, *Cycle Accurate, Bit Accurate*, pour précision au bit et au cycle près. Cette bibliothèque fournit des modèles pour le simulateur SystemC. Ces modèles ne sont pas des modèles RTL, ils ne sont donc pas synthétisables. Ils sont néanmoins équivalents aux modéles synthétisables pour ce qui est de leur architecture externe, avec, en outre, l'avantage de nécessiter beaucoup moins de puissance de calcul pour leur simulation. La création de notre processeur réseau sera, en fait, l'instanciation et l'interconnexion de modèles issus de la bibliothèque SOCLIB et de modèles ad hoc écrits pour cette application. Les connexions entre ces modèles sont rendues très simples par le fait que tous utilisent la même interface pour leurs communications. Il s'agit de l'interface VCI (*Virtual Componnent Interface*) [5]. Cette interface définit le format des échanges entre les différents éléments, aussi bien au niveau protocolaire qu'au niveau du nombre et de la signification des signaux utilisés.
Cette caractéristique essentielle de la bibliothèque SOCLIB permet une grande souplesse dans les essais d'architecture, puisque dès lors que le modèle souhaité existe, il est aisé de l'instancier dans un système.

3 L'APPLICATION: LE ROUTAGE IPV4

L'application que nous avons choisi de porter est le routage IPv4. Il s'agit d'une application connue [6], représentative des applications réseau. La description fonctionnelle initiale a été faite en utilisant Click. Click est un routeur modulaire pour PC [7]. L'un de ses principaux attraits est sa grande flexibilité, et l'écriture d'une application consiste simplement en la jonction de blocs fonctionnels simples. Les choix d'architectures initiaux de notre plateforme, expliqués au paragraphe suivant, sont tels que le processeur réseau auquel on destine cette application n'a que deux interfaces réseau. Pour cette raison, la structure de notre application ne comporte que quatre points d'entrées/sorties: deux entrées et deux sorties.
Les spécifications de notre application sont donc représentées dans le diagramme de la figure 2. Cette configuration est le point de départ de notre application. Nous avons utilisé comme référence le routage IPv4 fourni avec Click que nous avons modifié afin de le faire correspondre à nos besoins. Les modifications fonctionnelles apportées sont au nombre de deux.

La première concerne les entrées/sorties. En effet, la plateforme sur laquelle nous faisons fonctionner cette application est un élément d'un processeur réseau (voir figure 1), et de ce fait, ses interfaces d'entrée et de sortie de paquets ne sont pas symétriques. L'une est un lien Ethernet, et est donc déjà correctement traitée par les modules Click, tandis que l'autre est une jonction avec une matrice d'interconnection sur puce. Et cette matrice utilise également des en-têtes de routage; en-tête qu'il nous faut construire et ajouter au paquet lorsqu'il est émis, ou au contraire supprimer lorsque le paquet arrive.

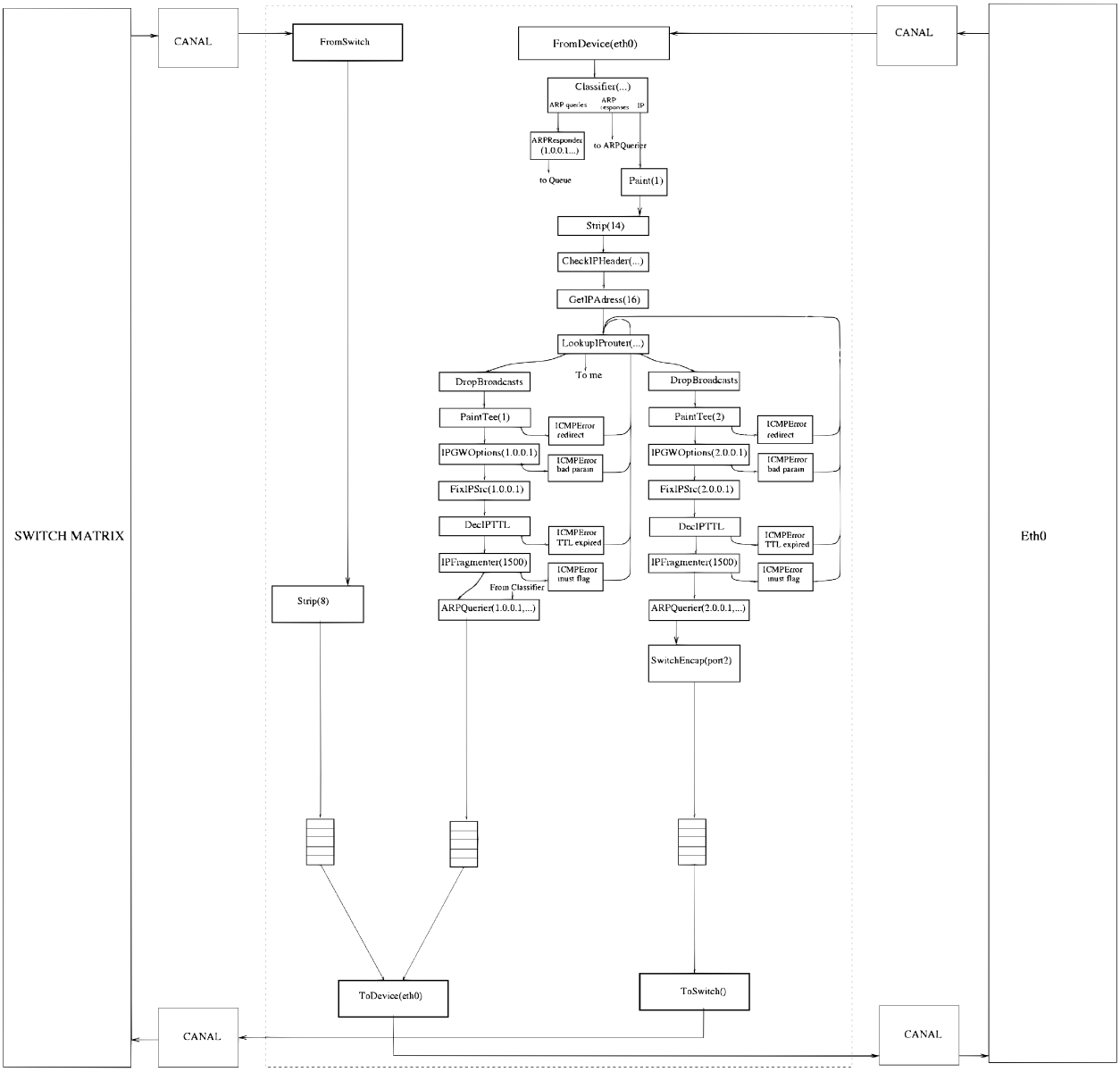

Figure 2: Configuration logicielle du routeur

La conséquence de cette modification est l'ajout de deux modules qui ne figurent pas dans la liste initiale des éléments de Click. Ces deux modules sont *SwitchEncap* et *FromSwitch*. Ils servent respectivement à encapsuler une trame Ethernet dans l'en-tête destiné au routage à l'intérieur de notre puce, et à recupérer une trame Ethernet après ce même routage interne.

La seconde modification est liée à la structure du routeur que nous utilisons. Puisqu'il s'agit d'un routeur à deux interfaces, et que l'une d'elles est connectée à la matrice d'interconnexion, on sait que les paquets qui proviennent de cette interface ont déjà été traités par un autre bloc, et qu'ils sont à destination du lien Ethernet auquel nous sommes raccordés. Ceci signifie que le travail de vérification ainsi que celui de routage ont déjà été effectués par une autre unité, le travail restant à fournir étant de diriger ce bloc vers l'interface de sortie. C'est la raison pour laquelle la figure 2 présente cette ligne droite entre l'entrée *from switch* et la sortie *toeth*. Le seul traitement à faire ici étant de supprimer l'en-tête de routage à travers la matrice d'interconnexion.

4 ARCHITECTURE MATERIELLE

L'un des critères de recherche pour notre architecture concerne la modularité. Après avoir fait en sorte de

maintenir l'application aussi indépendante que possible de son contexte d'utilisation, nous voulions pouvoir exploiter cette souplesse en developpant une architecture matérielle idoine.

Cette contrainte de modularité que nous nous sommes imposés nous a conduit à vouloir réaliser un sytème extensible.
Et pour matérialiser cette extensiblité, il nous est apparu que le mieux était de définir une "brique élémentaire" dont on instancierait à la demande la quantité requise.
Cet élément de base, cette "brique élémentaire", de notre routeur est lui-même un routeur, et le plus simple qui soit, puisqu'il s'agit d'un routeur à deux interfaces. Aussi, avons nous choisi de developper des systèmes à deux ports, et de les connecter entre eux pour obtenir un système global capable de traiter un nombre variable d'interfaces. Ce choix permet d'une part, de s'en tenir à la conception d'une plateforme ne traitant que deux interfaces réseau, et d'autre part de pouvoir facilement dimensionner notre plateforme en fonction des besoins réels en terme de nombre d'interfaces réseau.

Ces différents éléments, ainsi que leurs interconnections sont représentées sur la figure 1.
L'architecture que nous présentons comporte donc deux éléments distincts:

- Une matrice d'interconnexion reliant entre eux les processeurs réseau. Cette matrice peut être soit un full crossbar lorsque le nombre d'éléments à connecter est faible, soit un fat-tree si ce nombre est plus important.
- Le processeur réseau proprement dit. Il s'agit là de l'unité de traitement de notre architecture. C'est de l'architecture de cette partie que nous traiterons en details.

L'architecture matérielle que nous implémentons s'inspire initialement de celle développée par ST micro dans le cadre de leur plateforme STepNP. Il s'agit d'une architecture multi-processeurs, à deux interfaces réseau, articulée autour d'un réseau d'interconnexion sur puce.

Cette plateforme comprend d'une part des unités de calcul, représentées par des processeurs MIPS R3000, avec leur caches, ainsi que de la SRAM contenant le code de l'application et les données à traiter; et d'autre part des coprocesseurs d'entrés/sorties dont la fonction est d'insérer des trames Ethernet dans le système ou de les en extraire, une fois celles-ci traitées. Ces coprocesseurs sont au nombre de deux par interface réseau, un pour les entrées, et un pour les sorties. On les nomme respectivement *input-engine* et *ouput-engine*. L'architecture de ce processeur réseau est représentée sur la figure 3.
Tous ces éléments sont connectés entre eux par un réseau sur puce, à interface VCI. Il est à noter que l'interconnect utilisé pour les simulations est un modèle virtuel. Il s'agit du VGMN (pour VCI Generic Micro Network), modèle générique de réseau d'interconnexion dont on peut faire varier le nombre de ports, ainsi que la latence d'acheminement des données. Cette souplesse d'utilisation, associée à une faible consommation de ressources pendant la simulation a orienté notre choix. Cette facilité a cependant un revers qui est une perte de précision de notre simulation, puisque nous n'utilisons pas un interconnect réel, mais un émulateur.

Un premier avantage que l'on peut tirer de notre architecture modulaire est que ce nombre de coprocesseurs est fixe sur chaque NPU, et ce quel que soit le nombre d'interfaces que l'on compte connecter au système. Cette caractéristique nous évite notamment d'avoir à prendre en compte les problèmes liés à la saturation de la bande passante de l'interconnect qui serait induite par la multiplication d'injecteurs de paquets.

Cette constatation a également pour conséquence le fait qu'il nous sera beaucoup plus facile d'extrapoler les résultats obtenus sur les performances de notre architecture à un modèle plus grand, constitué d'instances de notre modèle de base, fonctionnant indépendament les unes des autres.

5 LOGICIEL EMBARQUE

L'aspect fonctionnel de notre application étant fixé, nous nous sommes penchés sur son portage sur l'architecture cible, à savoir notre processeur réseau. L'une des caractéristiques de ce système est qu'il est multiprocesseur. Pour profiter du parallélisme matériel que cela permet, il faut que notre application l'exploite.

Les deux grandes options étaient, soit de chercher les traitements indépendant les uns des autres dans l'applications et de les séparer en autant de tâches communiquant entre elles (parallélisme à gros grain), soit de dupliquer l'application elle même en un certain nombre de clônes, traitant chacun un paquet IP de bout en bout. Notre application ne comportant pas de parallélisme intrinsèque fort; le traitement d'un paquet devant s'effectuer pour sa plus grande partie séquentiellement, nous avons opté pour la seconde méthode, et toutes les tâches créées exécutent le même code, mais sur des paquets

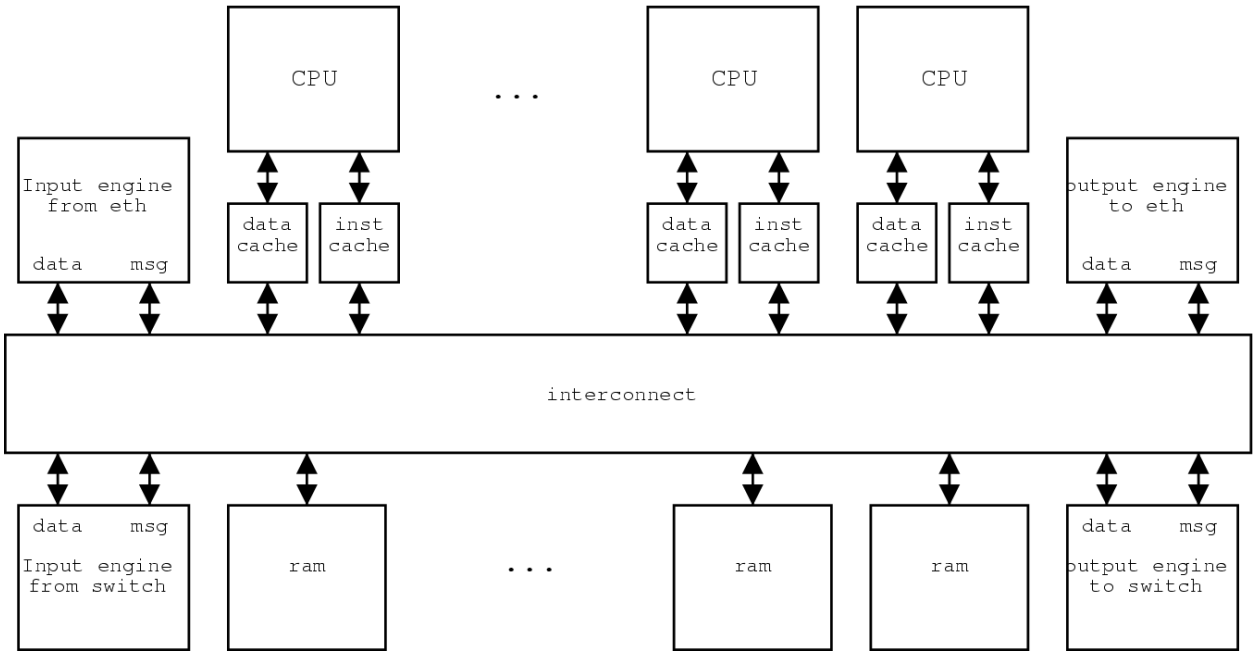

Figure 3: Architecture matérielle du processeur réseau.

différents.

Ces caractéristiques étant fixées, nous avons choisi, pour faire fonctionner ce système, de le compiler pour le micro noyau Mutek [8]. Ce noyau est multiprocesseur, multi-taches, et propose une bibliothèque C standard, ainsi que le support des threads POSIX. Il convient donc tout à fait à nos besoins.

Un autre point qui nous semble important, est que nous souhaitons assigner statiquement les tâches aux processeurs. Ceci afin d'éviter le surcoût induit par la migration d'une tâche vers un autre processeur, et aussi pour ne pas introduire de problème de cohérence mémoire lié à cette migration. C'est une option qui est également fournie par Mutek.

Le noyau et l'ordonnancement choisi, il nous reste à déterminer le nombre de tâches que chaque processeur doit se voir assigné. On peut en effet penser que le temps passé en attente de données pourrait être mis à profit en exécutant plusieurs tâches concurremment sur le même processeur. Ce n'est malheureusement pas le cas, et des tests faisant tourner deux tâches par processeur ont montré que le temps nécessaire au changement de contexte entre les tâches était plus important que celui offert par la latence des données. Nous restons donc finalement sur une architecture logicielle qui comporte autant de tâches que de processeurs, chacune d'elle étant allouée statiquement à un processeur.

6 VALIDATION ET PERFORMANCES

6.1 Validation

Notre processeur réseau comporte de deux parties, une matérielle et une logicielle, et qui doivent être testées ensemble.

La simulation de l'ensemble de la plateforme sous SystemC nous montre que notre architecture fonctionne, et qu'elle exécute l'application spécifiée. Pour valider à la fois le fonctionnement du logiciel embarqué et de l'architecture, nous avons mis en place la procédure de test suivante. Pour mettre en place ce test, nous avons eu recours à Click, ainsi qu'à StepNP. StepNP fournit une interface de communication entre processus nommée SIDL, dont le fonctionnement est voisin de CORBA. Click peut, pour sa part être facilement configuré comme générateur, ou comme analyseur de trafic.

Nous avons donc modifié nos modèles de coprocesseurs *input-engine* et *output-engine* afin qu'ils exploitent cette interface SIDL. Les premiers recevant leurs paquets d'un générateur Click, les second envoyant les paquets traités par notre plateforme vers un analyseur Click.
Ce dispositif fonctionne en faisant s'exécuter deux générateurs Click (un par *input-engine*), et deux analyseurs Click (un par *ouput-engine*) en plus de notre simulation SystemC. Grâce à cela, nous avons pu faire varier le type et la taille des paquets reçus

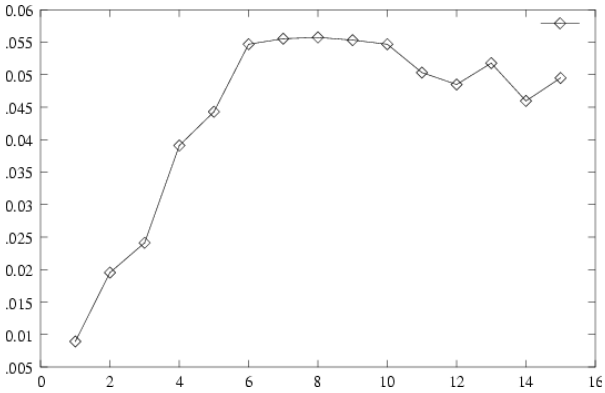

Figure 4: Débit obtenu en bits/cycle, en fonction

par notre routeur, incluant également des paquets erronés ou détériorés, et vérifier aux sorties que les paquets émis sont valides et sortent bien par la bonne interface.
Une fois cette étape passée, nous nous sommes intéressés aux performances de notre système.

6.2 Performances

Les performances que nous avons cherché à mesurer concernent le débit en bits/cycle que peut atteindre notre processeur réseau. Choisir le cycle d'horloge comme unité de temps s'explique par le fait que notre système est globalement synchrone, c'est à dire que tous ses composants partagent le même domaine d'horloge. D'autre part, nous n'avons pas fixé de fréquence de fonctionnement; il va de soi que plus on peut faire fonctionner le système à une haute fréquence, et meilleures sont les performances.

La première mesure que nous ayons prise est celle qui nous servira de référence pour la suite. C'est le débit assuré par le système lorsqu'on n'y instancie qu'un seul processeur. C'est le débit maximal que peut fournir un CPU seul. Cette valeur nous servira de référence pour estimer l'amélioration apportée par le traitement multiprocesseur.

En effet, lorsqu'on ajoute des processeurs au système, on améliore les performances, mais cette croissance ne sera pas linéaire. C'est à dire qu'avec X processeurs, on a un débit inférieur à X fois le débit d'un processeur seul. Cette perte est due au partage des ressources entre les différents processeurs; et l'on souhaite pouvoir la quantifier afin de déterminer la rentabilité de l'ajout d'un processeur.

La méthode choisie pour ce relevé de mesure est la suivante: on fait tourner la simulation pendant 2 000 000 de cycles, et on compte le nombre de mots de 32 bits obtenu sur chacune des interfaces de sortie. On néglige ici l'impact du temps de boot (environ 16 000 cycles) ainsi que le ou les paquets déjà traités mais pas encore sortis. On précise que les paquets IP utilisés pour cette simulation sont de taille minimale: 56 octets, ce qui correspond pour nous au pire cas.

Cette simulation nous donne un total de 689 mots de 32 bits lus sur les sorties, ce qui fait un débit de 0.011 bit/cycle.
Ce résultat peu flatteur s'explique en grande partie par le manque d'optimisation de notre application. On constate en effet que ce faible débit correspond à un temps de traitement moyen d'environ 40 000 cycles par paquet IP. Le détail du temps d'exécution de chaque fonction est résumé dans la figure 5 pour un paquet entrant par le lien Ethernet et sortant par la matrice d'interconnexion. Ce qui correspond au plus long chemin que peut parcourir un paquet entre deux interfaces (environ 110 000 cycles).

C'est néanmoins ce résultat que nous utiliserons comme valeur de référence pour les tests multiprocesseur. Le premier test multiprocesseur consiste simplement à faire varier le nombre de processeurs sur le circuit. La figure 4 illustre les résultats obtenus, en présentant en abscisse, le nombre de processeurs présents, et en ordonnée, le débit obtenu, en bit/cycle.

Les mesures faites montrent que le meilleur débit est atteint pour 8 processeurs, mais que celui-ci est à peine supérieur à celui atteint avec 6 processeurs. Les unités ajoutées après ce sixième processeur n'amènent aucun gain significatif, voire détériorent les performances. Ceci est dû au fait que tous les processeurs se partagent les mêmes ressources, particulièrement les bancs de mémoire où sont copiés les paquets IP avant ou après leur traitement.

Cette constatation nous a amenés à distribuer la mémoire sur l'interconnect.
La répartition de la mémoire s'effectue en remplaçant les quatre gros segments de mémoire utilisés par les coprocesseurs d'entrées/sorties en une multitude de segments plus petits. Comme l'un de ces segments supprimés servait également au stockage des variables locales des threads, sa plage d'adresse doit être conservée. Ce segment est donc réparti en plusieurs bancs de RAM, tandis que les autres sont supprimés. Le choix de la répartition est d'ajouter un nouveau segment par processeur. Ce segment doit pouvoir remplir 3 fonctions:

- contenir les données locales du thread, tâche qui lui était déjà dévolue précédement.
- Permettre aux coprocesseurs d'entrée d'y écrire les paquets IP entrant.
- Fournir un espace pour que le processeur y copie les paquets sortant.

Elément	Temps d'exécution (en cycles)	Pourcentage du total
Acquisition	30534	27,14
Clasifier	608	0,54
paint	25332	22,52
strip	356	0,316
Checkipheader	8506	7,56
Getipaddress	2934	2,61
Lookupiprouter	598	0,53
Dropbroacast	394	0,35
Painttee	346	0,31
IPGWOptions	634	0,56
FixIPsrc	314	0,28
DecIPTTL	998	0,89
IPFragmenter	422	0,38
EtherEncap	15308	13,61
Extraction	15218	22,42
Total	**112502**	**100**

Figure 5: Détermination de la durée de chaque fonction en nombre de cycles (mono-thread, MTU=56)

Concrètement, cela signifie que chacun de ces bancs mémoire est divisé en trois zones ne se recouvrant pas, l'une d'elle (la seconde) étant subdivisée en deux sous-sections: une pour chaque *input-engine*. Aucun de ces deux coprocesseurs n'ayant connaissance de l'autre, on ne souhaite pas compliquer leur fonctionnement en leur faisant partager des variables telles que les adresses qu'ils utilisent.

Cependant, le fait de répartir les bancs mémoire, et ainsi de diviser en segments disjoints les plages d'adresses disponibles pour les *input-engines*, nous a conduit à modifier ceux-ci.

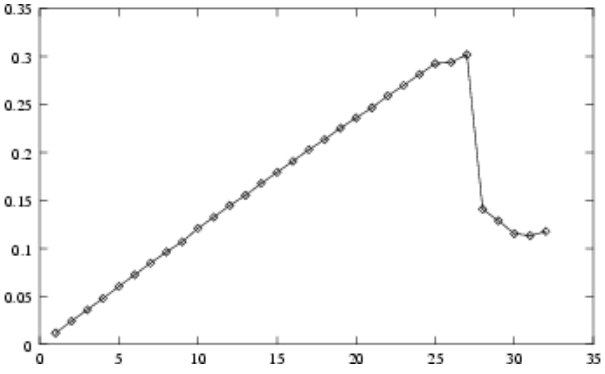

Figure 7: Débit obtenu, en bits/cycle, en fonction du nombre de processeurs, pour une mémoire répartie.

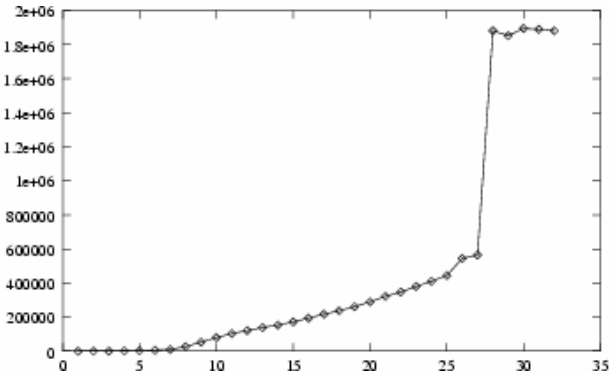

Figure 6: Nombre de cycles de contention sur la RAM DATA, en fonction du nombre de processeurs, pour une période de 2 000 000 de

Premièrement, en leur faisant prendre en compte un plus grand nombre de cibles disponibles, nombre variable en fonction du nombre de processeurs sur la carte.
Deuxièmement, en mettant en place un algorithme de choix d'une cible parmi celles disponibles. Notre choix s'est porté sur l'algorithme round-robin, à la fois équitable dans la répartition de la charge et simple à implémenter.
Cette modification de l'architecture du système a permis un gain de performance tout à fait significatif, et surtout, cela a permis de continuer à voir les performances augmenter pour un nombre de processeurs bien plus important. Pour ce deuxième test (figure 7), la forme de la courbe est semblable à la première, à la différence près que le pic de performance est atteint pour 27 processeurs, avec un débit d'environ 0.3 bit/cycle.

Ici aussi on constate un tassement des performances à partir d'un certain seuil, suivi par une chute. La raison est la même que lors du premier test: il s'agit des contention d'accès aux ressources, et plus particulièrement aux deux segments de mémoire partagés par tous les threads: le segment de code et celui contenant les variables globales du système.

Ces contentions ont pu être mesurées et ces mesures reflètent bien la baisse des performances.
La courbe de la figure 6 montre le nombre de cycles de contentions sur la RAM de données globales pendant un laps de temps de 2 000 000 de cycles. Par contention, on entend qu'il y a plus d'un initiateur tentant d'établir une communication avec la cible. Ce qui signifie qu'au moins un initiateur est bloqué dans l'attente de la libération de la ressource.

A l'examen de cette courbe, on constate que pour 28 et plus processeurs, le nombre de cycle de contention dépasse 1,8 millions, ce qui signifie que le sytème est paralysé.

7 CONCLUSION ET PERSPECTIVES

En spécifiant, puis en écrivant un modèle de simulation de cette plateforme, nous avons pu remplir deux objectifs.

Le premier concerne le choix du système de simulation, ainsi que de son niveau d'abstraction (la simulation cycle par cycle). Ce niveau de simulation représente en effet un bon compromis entre la finesse de la simulation et sa rapidité. La rapidité permet de faire fonctionner une vraie application, tandis que le niveau de détail obtenu permet d'observer finement le comportement du système, et ainsi de remédier à ses insuffisances.

Le second objectif concerne l'architecture elle-même. Nous avons en effet réussi à décrire le coeur d'un processeur réseau, et à faire fonctionner une application logicielle multi-tâche sur cet ensemble. Et si les performances de notre système demeurent modestes, nous pensons, d'une part que cette faiblesse est due à un manque d'optimisation de la partie logicielle, et d'autre part, que les résultats que nous avons obtenus ne souffrent pas, ou peu, de l'instanciation multiple de notre unité.
Les perspectives de développement qui s'offrent à nous empruntent plusieurs directions: l'étude des performances du système, ainsi que sa mise au point ont montré la nécessité de disposer d'outils d'aide au débug. De tels outils devant pouvoir servir aussi bien à mettre au point le logiciel, qu'à en évaluer le temps d'exécution. Il serait en effet intéressant de savoir combien de temps chaque processeur passe dans chaque fonction, et ainsi déterminer précisément quelles sont les ressources critiques.
Un autre axe de recherche concerne la structure de l'application. Celle que nous avons employée pour nos tests est en réalité très simple, et il serait intéressant d'étudier l'impact sur les performances du système de la mise en place d'autres fonctionnalités. On peut entre autres penser à la qualité de service, ou au cryptage.

Cette deuxième perspective en ouvre naturellement une troisième qui est la recherche architecturale en fonction des buts visés par l'application. On peut par exemple penser à des coprocesseurs spécialisés pour assurer le cryptage de données.

Finalement, on peut dire que l'architecture que nous nous sommes employés à mettre au point offre un point de départ pour une vaste gamme d'applications, allant du "simple" routage de paquets IPv4 jusqu'à l'analyse de trafic, ou au cryptage. Cette plateforme est donc très ouverte, et le niveau de simulation choisi permet à la fois une analyse fine de son comportement, tout en conservant de bonnes performances en terme de temps de simulation.

REFERENCES

[1] T. Groetker, S. Liao, G. Martin, and S. Swain, *System Design in SystemC*. Kluwer, 2002.

[2] P. G. Paulin, C. PilKington, and E. Bensoudane, “Plate Forme StepNP”, 2002, Ottawa, Canada.

[3] Lukai Cai and Daniel Gajski, “Transaction level modelling in system design”, University of California, Irvine, Technical Report. March 2003.

[4] S. Consortrium, Projet SOCLIB : “Plateforme de modélisation et de simulation de systèmes intégrés sur puce”, Tech. Rep. [Online]. Available: http://soclib.lip6.fr

[5] VSI Alliance, “Virtual Component Interface Standart (OCB 2.2.0)”, Tech. Rep., [Online] URL=http:// www.vsi.org/library/specs/summary.htm#ocb.

[6] F. Baker. “Requirement for ip version 4 router”, June 1995, Intenet Ingineering Task Force, ftp://ftp,ietf.org/rfc/rfc1812.txt

[7] R. Kohler, “ Click /system free software”, 1995, URL=www.pdos.lcs.mit.edu/click.

[8] P. Gomez, F. Petrot, and D. Hommais, “Mutek : Un noyau multi-tâches, multiprocesseurs pour systèmes embarqués”, Laboratoire LIP6, Departememt ASIM, Paris, Tech. Rep., 2002. [Online]. Available : www.asim.lip6.fr