Parallélisation d'un routeur XTP

Jean-Marc Jézéquel

Parallélisation d'un routeur XTP

Jean-Marc Jézéquel[1]

I.R.I.S.A./C.N.R.S.
Campus de Beaulieu
F-35042 Rennes Cedex, *FRANCE*
E-mail: jezequel@irisa.fr
Tel: +33 99 84 71 92 ; Fax: +33 99 38 38 32

Résumé. *Nous étudions dans cet article la faisabilité d'utiliser des machines parallèles à mémoire distribuée à usage général pour réaliser des fonctions de réseaux haut débit. Nous nous sommes intéressés au problème de la commutation dans le cadre du protocole XTP (protocole de gestion des couches réseau et transport pour les réseaux haut débit). Cette étude s'appuie sur les outils généraux développés dans le projet PAMPA (méthodes et outils de parallélisation et de validation) et a donné lieu à une évaluation de performance en fonction des types et tailles de machines utilisées.*

Abstract. *We study in this paper the feasibility of using general purpose distributed memory parallel computers to implement some functions of high speed networks. We concentrate on the switching problem of the XTP high performance protocol (dealing with the network and transport layers of the OSI stack). We use methods and tools developed within the PAMPA project (parallelizing methods, formal specification tools) to parallelize and prototype an XTP switching system on parallel architectures, and to make performance evaluations.*

1 Introduction

Au cœur des réseaux actuels de communication par paquets, on trouve des *commutateurs* chargés de mettre en relation des équipements communicants et d'assurer le transit des données. Ces commutateurs sont en général des machines multi-processeurs plus ou moins spécialisées munies de fortes capacités d'entrées-sorties.

[1]Ce travail a bénéficié du soutien de France Telecom/CNET au titre du marché n. 92 1B 178 *Conception formelle de systèmes de coopération haut débit multimédia*

Lorsque les processeurs communiquent par mémoire commune, leur débit est souvent limité par le goulot d'étranglement constitué par l'accès au bus. Ces architectures classiques ne pourront donc que difficilement absorber les débits envisagés pour les futurs réseaux haut débit basés sur la technologie ATM.

C'est pourquoi nous pensons qu'il serait intéressant d'étudier quelles sont les possibilités de réaliser de tels commutateurs et serveurs à l'aide de machines parallèles à mémoire distribuée. En effet, le problème du goulot d'étranglement constitué par l'accès au bus devient caduque et leur puissance est par construction facilement modulable. Nous nous proposons ici d'étudier la faisabilité d'utiliser des machines parallèles à mémoire distribuée à usage général pour réaliser une fonction simple des réseaux haut débits. Nous nous sommes intéressés au protocole XTP (eXpress Transfert Protocol) qui est un protocole de gestion des couches réseau et transport (couches 3 et 4 du modèle OSI) pour les réseaux haut débit (100 Mbits/s à 10 Gbits/s). Sa conception a été motivée par les besoins des systèmes distribués temps réel, transactionnels et multimédia, actuels et futurs [4].

Cette étude s'appuie sur les outils généraux développés dans le projet PAMPA (méthodes et outils de parallélisation, de validation et de compilation de spécifications formelles) et a donné lieu à une évaluation de performance en fonction des types et tailles de machines utilisées.

Nous présentons dans la section 2 notre modélisation d'un réseau XTP organisé autour d'un routeur, ainsi que sa spécification formelle et son implantation. En section 3 nous décrivons les techniques de parallélisation que nous avons appliquées à notre routeur XTP, puis nous présentons en section 4 des résultats expérimentaux obtenus sur une machine parallèle à mémoire distribuée représentative, l'hypercube Intel iPSC/2.

2 Modélisation d'un réseau XTP

Il s'agit de modéliser le fonctionnement d'un réseau XTP c'est à dire d'un ensemble de sites (i.e. machines autonomes ou "gateway" de réseaux locaux, appelés dans la suite Equipement XTP ou EXTP) connectés à un routeur XTP et dialoguant à l'aide d'un sous ensemble du protocole XTP.

2.1 Présentation du protocole XTP

XTP (eXpress Tranfer Protocol, décrit dans [10]) est un protocole de transfert conçu pour être utilisé dans les réseaux haut débit, de 100 Mbit/s jusqu'à 1Gbit/s. Il incorpore des fonctionnalités des couches *Réseau* et *Transport* telles que définies par le modèle de référence OSI [11] de l'ISO.

Une unité de donnée du protocole (ou PDU, ou paquet) XTP est composée d'un en-tête, d'un segment de contrôle ou d'information et d'une queue. Les paquets XTP qui contiennent un segment de contrôle (CNTL ou RCNTL) sont appelés paquets de contrôle. Les segments d'information contiennent des

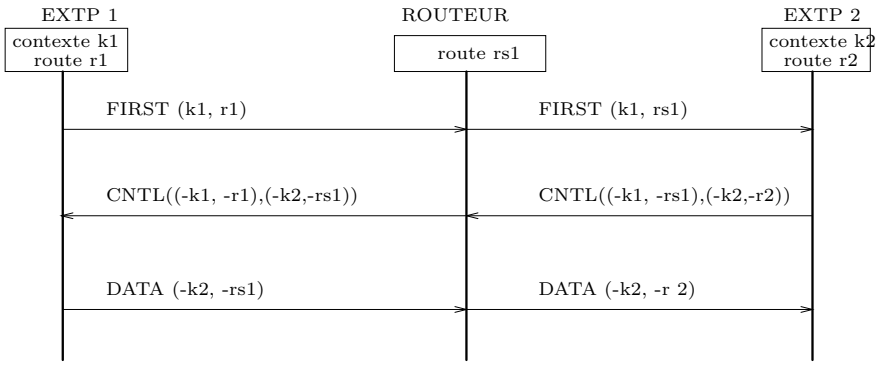

Figure 1 : Ouverture d'une communication - échange des clés - envoi d'un paquet DATA

données : c'est le cas pour les paquets FIRST et DATA d'une part (ce sont les données utilisateurs, i.e. des couches supérieures), et PATH, DIAG, MAINT, et ROUTE d'autre part (ce sont des messages de la couche transport). Un sous-segment d'adresse est nécessaire pour deux types de paquet : FIRST et PATH. Il contient des informations sur le destinataire et sur l'émetteur permettant d'établir un chemin pour les paquets de données et de contrôle.

Détaillons maintenant le principe du transit par un routeur XTP. Le commutateur (routeur) contient une table d'associations, dont chaque élément est un couple de routes: route de l'appelant (initiateur de la communication) et route de l'appelé.

Dans le protocole XTP, la lecture dans les tables (de routes ou de contextes) peut se faire par accès séquentiel à l'information, ou bien par accès direct suivant que la clé utilisée est positive ou négative.

si *route* a une valeur négative, l'accès est direct,–*route* sert d'indice.

si *route* a une valeur positive, l'accès est séquentiel : parcours partiel de la table, avec arrêt en fin de table ou s'il existe une entité qui utilise *route*.

De la même manière, on retrouve ces deux types d'accès pour les tables de contextes, où *key* a remplacé *route*.

La figure 1 illustre l'établissement d'une communication entre deux entités au travers d'un routeur.

Soit un EXTP (que nous appellerons EXTP1) qui désire communiquer avec un EXTP distant (que nous appellerons EXTP2). L'EXTP1 envoie un paquet FIRST dont l'en-tête contient ses identificateurs de contexte et de route *k1* et *r1*. Lorsque le commutateur reçoit le paquet FIRST, il cherche dans sa table des routes une route disponible à utiliser pour cette communication (soit *rs1*).

Il remplace r1 par rs1 dans le paquet FIRST qui est expédié au destinataire suivant, ici l'EXTP2.

Lorsque l'EXTP2 reçoit le paquet, il va chercher un contexte disponible dans sa table des contextes —soit k2 sa référence. Il associe ce *contexte* à la communication qu'il traite ($contexte.key := -k1$) et cherche en même temps une *route* (trouvée en position r2) dans sa table des routes.

A ce stade, l'EXTP2 peut répondre à l'EXTP1 par un paquet CNTL. Quand il répond, comme il connaît son interlocuteur final et le commutateur le plus proche[2], l'EXTP2 va utiliser les clés avec un signe négatif ce qui permet un accès direct dans la structure de données considérée.

Par contre, l'utilisation des clés avec leur valeur positive, implique un accès séquentiel dans la structure de données, l'identification de l'élément recherché se faisant par comparaison sur la valeur de la clé.

L'EXTP2, en répondant, peut effectuer un échange des clés, afin de rendre la communication parfaitement symétrique : il passe dans le segment de contrôle du paquet la valeur de k2 et r2 (en valeurs négatives). Lorsque le commutateur reçoit le paquet CNTL, la clé $-rs1$ lui permet de localiser directement l'élément dans la table des routes; on récupère la valeur associée $-r1$ et le paquet ainsi modifié est envoyé vers l'EXTP1.

Dans le cas où l'on effectue cet échange de clé, le commutateur enregistre $-r2$ et modifie $-r2$ par $-rs$ dans le paquet CNTL. Au niveau de l'EXTP1, on se positionne sur le contexte adéquat (donné par la connaissance de $k1$). Si on effectue un échange des clés, on récupère $-k2$ et $-rs1$.

Après l'échange des clés, la table des routes devient symétrique; et une fois initialisées, route de l'appelant et route de l'appelé ont un rôle identique, ce qui pose d'ailleurs un problème d'identification du sens de commutation au niveau du commutateur.

2.2 Modélisation avec Estelle

Estelle est un langage qui permet de décrire des protocoles de communication et des algorithmes distribués, et dont la sémantique est définie formellement. Normalisé par l'ISO (IS 9074 [6]), Estelle est fondé sur la notion d'automate d'états finis étendu avec le langage séquentiel PASCAL.

Nous voulons pouvoir modéliser à la fois le routeur XTP et son environnement, c'est à dire l'ensemble des machines qui y sont connectées. Comme nous nous intéressons, dans cette étude, à l'interaction EXTPs/ROUTEUR, nous décomposons notre système en :

- un nœud de réseau (commutateur ou routeur).
- un environnement extérieur qui se subdivise en plusieurs systèmes terminaux ou EXTPs (qui représentent les utilisateurs).

[2]L'EXTP2 connaît le contexte appelant par k1 et la route à utiliser dans le commutateur le plus proche rs1.

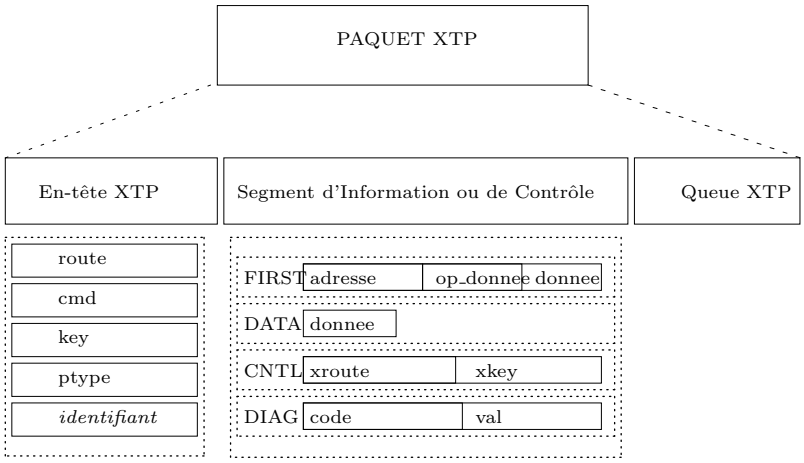

Figure 2 : Détail de notre paquet XTP

Chaque EXTP doit pouvoir demander l'ouverture d'une association (ie : communication), en demander la fermeture, demander l'envoi de paquets : *FIRST, DATA, CNTL, DIAG,...* Une association ou communication est caractérisée par un contexte (et une route). Un EXTP peut ouvrir autant d'associations qu'il possède de contextes.

Le commutateur doit permettre :

- de router un paquet qui arrive sur le nœud de commutation : il s'agit du choix d'un port de sortie en fonction du destinataire final, en vue d'établir un chemin pour tous les paquets de la communication. (Cela implique une stratégie de routage.)
- de commuter un paquet : c'est à dire, de l'aiguiller en fonction de ses paramètres vers le port de sortie adéquat pour qu'il suive le même chemin que ses pairs.
- d'effectuer des allocations et libérations de route.

La méthode de développement utilisée a consisté à construire une spécification du modèle sous forme d'automate à états finis, puis de l'enrichir pour arriver à un modèle en Estelle où chacun des éléments de notre système (commutateur et EXTPs) est représenté par un module.

Des structures de données manipulées par le protocole XTP, nous n'avons gardé que ce qui nous était strictement nécessaire pour notre modélisation. De plus, nous ne nous sommes pas préoccupés du format exact des données (i.e. : alignement sur 4 octets, etc.), ni de la nature de ces données (voir figure 2).

Nous avons aussi été amenés à faire deux petites adaptations sur le protocole XTP : l'une concernant la libération sur le passage d'un paquet DIAG des routes utilisées au niveau des commutateurs traversés ; et l'autre concernant la libération sur le passage d'un paquet CNTL de la route associée au dernier contexte de cette route (cf. [9] pour plus de détails).

2.3 Principes de la réalisation avec ECHIDNA

L'environnement d'expérimentation ECHIDNA est un environnement logiciel développé à l'IRISA pour aider à l'expérimentation des algorithmes distribués sur machines parallèles [8, 7]. Il comprend un compilateur du sous-ensemble dit *statique* du langage Estelle et des noyaux d'exécution distribués pour les machines Intel iPSC, FPS-T40, Telmat T-node, réseaux de Sun (3 et 4), autres machines Unix et même PC ; ainsi qu'un ensemble d'outils logiciels pour observer de manière non intrusive le comportement de l'algorithme distribué qu'on "expérimente" (constructeur de temps global, traces, auto-enregistrement d'événements...).

Dans une première phase (modélisation séquentielle du routeur), le programme réalisé a été exécuté sur station de travail SUN en environnement UNIX (interface graphique X-WINDOW ou Suntools). Nous y avons programmé une trace d'exécution qui permet d'examiner le type des paquets échangés, la valeur des variables de l'environnement par EXTP et par contexte et la valeur des variables du commutateur. Dans une seconde phase (modélisation parallèle du routeur), nous avons utilisé l'iPSC/2 (machine parallèle à base de 32 processeurs Intel 80386 disposés suivant une topologie hypercube).

Pour modéliser les conséquences des échanges de primitives de services entre la couche transport/réseau que nous avons simulée et la couche supérieure (session/présentation/application), utilisatrice du service transport/réseau, nous avons modélisé les transitions idoines par des transitions spontanées de la couche transport/réseau. Le corps de la transition est gardé par un test sur une variable qui indique si on doit effectuer le traitement.

Ainsi, le mécanisme de requête/émission de données va être représenté par une transition spontanée à l'intérieur de laquelle on trouve un traitement gardé par un test sur la fréquence d'émission de donnée (c'est le cas des transitions pour l'envoi de paquet FIRST). Dans notre programme, la transition est un peu plus complexe: en effet, on peut simultanément ouvrir-envoyer-fermer, ou tout autre combinaison intermédiaire.

D'autre part, pour modéliser une temporisation, on utilise habituellement en Estelle la clause *delay*. Le problème est que dans la sémantique du langage, cette clause n'a pas de sens temps réel particulier. De plus, comme nous avons choisi un modèle "probabiliste" pour simuler le trafic (en terme de fréquences moyennes d'occurrences d'événements) il semblait logique de suivre la même approche pour modéliser les événements d'échéance de temporisations.

3 Implantation Parallèle avec ECHIDNA

3.1 Techniques de parallélisation de protocoles

Un des moyens les plus classiques utilisés pour augmenter la vitesse de traitement d'une application consiste à isoler des unités fonctionnelles indépendantes qu'on confie à des processeurs différents. C'est l'approche suivie par exemple dans [2]. On s'y intéresse au traitement parallèle du protocole XTP au niveau d'un système terminal : il s'agit donc d'insérer le protocole XTP dans une implémentation d'une pile de protocole sur une architecture multiprocesseur. Les systèmes de communication sont typiquement structurés en couches de protocoles hiérarchisées (cf. modèle OSI), opérant suivant un principe de "pipe line".

Cette approche est cependant fondée sur un modèle de programmation MIMD, difficile à manipuler dans un cadre de parallélisme massif. En effet, comme une fonction est décomposée en sous-fonctions pouvant être exécutées par de nouveaux flots de calculs, une application se trouve alors découpée en un ensemble de processus communicants dont la structure dépend de l'application et de la façon dont l'utilisateur la décompose. Comme ces processus sont par nature hétérogènes, des problèmes d'équilibrage de charge doivent être réglés soit par le système d'exploitation (ce qui est coûteux) soit par l'utilisateur lui-même.

Ce parallélisme de type fonctionnel n'est pas extensible à volonté : étant lié à la structure dynamique d'une application, peu de structures régulières peuvent être dérivées du programme original lors de la compilation, aussi la plus grande partie du travail doit être menée à l'exécution. Cette nature dynamique implique l'utilisation de mécanismes généraux et coûteux pour permettre par exemple la création dynamique de processus, le nommage uniforme des objets, la migration de données ou l'appel de procédures à distance, l'équilibrage de charge, le scheduling, etc.

Mais d'autres modèles de programmation s'éloignant moins des habitudes des programmeurs ont été proposés par ailleurs pour les architectures parallèles à mémoire distribuée (APMD) : par exemple le modèle SIMD (Single Instruction, Multiple Data) permet parfois de décrire certains algorithmes de manière naturelle. D'importants travaux de recherche sont aussi menés sur le thème de la parallélisation totalement automatique de codes SISD (Single Instruction Single Data) de type FORTRAN. Mais cette approche ne semble pas très bien adaptée aux APMD.

Le modèle SPMD (Single Program, Multiple Data) est une sorte de compromis entre ces deux approches. Ce modèle tente de prendre en compte le fait que la plupart des problèmes qu'on souhaite résoudre sur des APMD est caractérisée par la grande quantité de données à traiter. L'idée de base (décrite dans [3] et formalisée dans [1]) est alors de partitionner cette masse de données, et d'associer statiquement un processus à chacune de ces partitions. Chaque processus exécute le même programme (correspondant au programme utilisa-

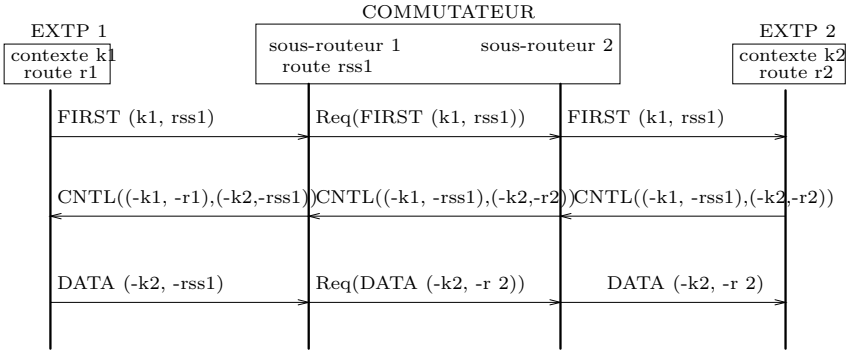

Figure 3 : Ouverture d'une communication - échange des clés - envoi d'un paquet DATA. (modèle parallèle).

teur initial), mais seulement sur sa partition. Ainsi la vue qu'a cet utilisateur de son programme reste séquentielle : une séquence d'actions est appliquée à un ensemble de données ; et le parallélisme découle de la décomposition des données, ce qui conduit à une forme de parallélisme régulière et extensible.

Nous nous sommes donc intéressés au paradigme de parallélisation dirigée par les données au niveau des connexions d'un commutateur de réseau, ce qui est en fait assez orthogonal à la parallélisation d'unités fonctionnelles indépendantes au niveau d'un système terminal. En effet, bien loin d'être incompatibles, ces deux approches de l'utilisation du parallélisme sont au contraire complémentaires.

3.2 Parallélisation du commutateur XTP

Le système considéré précédemment n'est modifié qu'au niveau du commutateur. Vu de l'extérieur le commutateur présente la même interface et utilise le même protocole de communication que dans le modèle séquentiel.

Dans notre modèle parallèle, notre commutateur est divisé en un certain nombre de mini-commutateurs (ou sous-routeurs) qui ont chacun toutes les fonctionnalités et les mêmes structures de données que le commutateur du modèle séquentiel (et en grande partie le même code).

La seule différence réside dans le fait que la table des routes pour un même commutateur se trouve "répartie" entre les mini-commutateurs. A l'intérieur du commutateur les différents mini-commutateurs communiquent entre eux par des liens de communication interne de l'APMD. Nous avons fait en sorte que la répartition soit équitable, que notre système soit statistiquement équilibré : la répartition des liens externes (canaux XTP représentant les lignes physiques) entre les différents sous-routeurs se fait suivant le modulo du nombre de sous-

routeurs.

Cet éclatement de la table des routes nous a conduit à appliquer quelques règles classiques de la parallélisation dirigée par les données (écritures locales et Refresh distant [1]) pour accéder aux contextes non-locaux. Chaque sous-routeur gère de manière habituelle toute nouvelle communication (paquet FIRST) lui parvenant sur ses liens extérieurs, et sera le "propriétaire" de cette communication. Pour les autres paquets, le numéro de route *nbr* contenu dans l'en-tête du paquet reçu va permettre de savoir quel sous-routeur est le propriétaire de la communication, et doit donc s'occuper du paquet.

Le mécanisme de communication s'effectue en deux temps et à deux niveaux :

- la route, contenue dans l'en-tête du paquet XTP, permet, lorsque le paquet entre dans le commutateur, de savoir vers quel sous-routeur il doit aller pour que soit effectuée la commutation.
- le lien (de sortie) permet de connaître le sous-routeur qui renverra le paquet vers l'extérieur.

L'attribution d'une route (locale) de la table des routes d'un sous-routeur est semblable à ce qui se faisait avec le commutateur séquentiel. Cependant, cette route doit être transformée en route à transmettre de manière à ce qu'elle corresponde à une route parmi l'ensemble des routes du commutateur. Réciproquement, les tables des sous-routeurs peuvent être retrouvées en découpant la table globale modulo le nombre de sous-routeurs.

Ainsi, le schéma d'une ouverture de communication qui était celui de la figure 1 dans le modèle séquentiel, devient celui de la figure 3 dans le modèle parallèle.

3.3 Placement sur une machine à topologie hypercubique

Lors des divers essais réalisés sur l'hypercube [5], nous avons attribué les processus du modèle Estelle aux nœuds de l'hypercube en prenant garde de ne pas induire des routages internes supplémentaires. Pour cela, processus "EXTP" et processus "sous-routeur" sont affectés dans des hyperplans distincts.

4 Résultats d'expérimentation

4.1 Conditions expérimentales

La charge du routeur XTP est réalisée par 16 EXTPs contenant chacun 64 contextes et 64 routes (on n'autorise pas le partage de route) : on peut donc obtenir jusqu'à 1024 communications simultanées. Lors de l'ouverture d'une nouvelle communication, l'EXTP correspondant est déterminé de façon aléatoire (loi uniforme) : toutes ces communications se croisent donc dans le routeur de manière aléatoire.

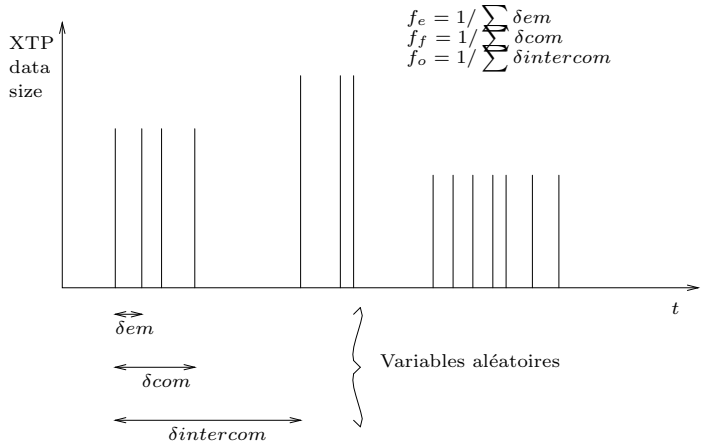

Figure 4 : Profil de trafic pour un EXTP

Le profil du trafic généré pour chaque communication de chacun de ces EXTPs[3] est illustré en figure 4. Il est caractérisé par quatre paramètres :

- la taille du segment de données des trames XTP (paquets FIRST et DATA)
- la fréquence moyenne d'ouverture d'une nouvelle communication f_o
- la fréquence moyenne de fermeture d'une communication f_f
- la fréquence moyenne d'émission d'une trame DATA f_e

Ceci permet d'avoir un trafic de type aléatoire "contrôlé" plaçant le routeur XTP dans des conditions vraisemblables d'exploitation, avec des moyennes paramétrées. Ce n'est pas tant les valeurs absolues de ces paramètres qui présentent un intérêt que leurs rapports respectifs : ainsi f_e/f_f est proportionnel au nombre moyen de paquets DATA émis lors d'une communication XTP, et f_o/f_f est proportionnel au nombre moyen de communications ouvertes par un EXTP.

Un des paramètres fondamentaux que nous voulons pouvoir mesurer est le débit total que peut écouler notre routeur XTP dans des conditions vraisemblables de fonctionnement : c'est en effet ce qui en caractérise la capacité, et qui nous permettra de déterminer s'il est possible d'augmenter cette capacité simplement en lui ajoutant des processeurs.

[3]Ce profil de trafic est donc multiplexé 64 fois sur chacun des 16 "canaux XTP" reliant le routeur aux EXTP

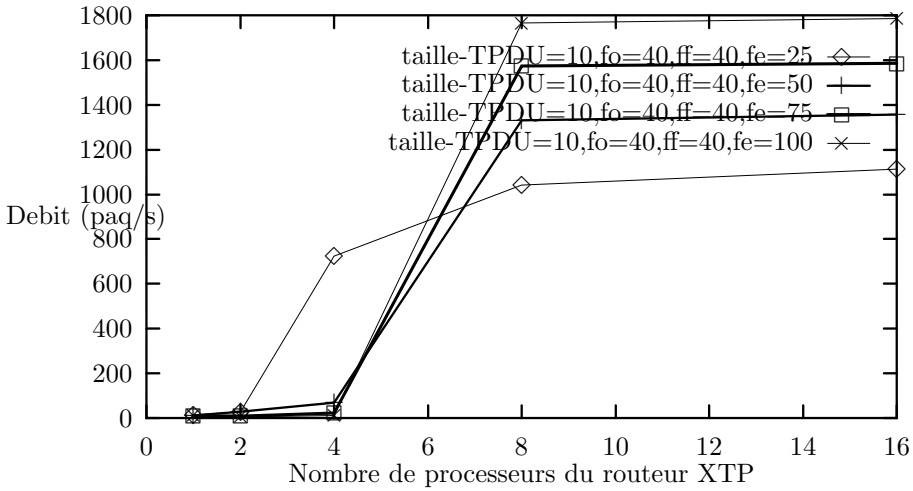

Figure 5 : Débit utile écoulé par le routeur XTP, avec une taille du segment de données des trames XTP égale à 10

Pour cela, nous allons comptabiliser le nombre total de paquets XTP reçus par les EXTP pendant une expérience de durée constante. Nous traçons en figure 5 l'évolution de ce total en fonction du nombre de processeurs (1, 2, 4, 8 ou 16) utilisés dans le routeur XTP, le comportement en émission des EXTP restant statistiquement constant (les quatre courbes correspondent à différentes valeurs du paramètre f_e).

4.2 Mesures et résultats expérimentaux

Nous constatons que nous nous trouvons dans une zone de trafic où le routeur est complétement débordé lorsque son nombre de processeurs est inférieur à 16 (4 pour la courbe $f_e = 25$), mais que le passage à 16 n'augmente pas son débit écoulé.

Le même type de mesures a été fait avec des tailles de segment de données des trames XTP plus importantes : ceci n'altère pas profondément la forme des courbes, si ce n'est que l'écroulement du routeur se produit plus vite : dans certains cas, il faut au moins 16 processeurs au routeur pour lui permettre d'écouler du trafic sans se laisser déborder.

Des mesures dans d'autres zones de trafic nous confirment ce phénomène : notre routeur fonctionne en "tout ou rien". C'est à dire que pour un trafic

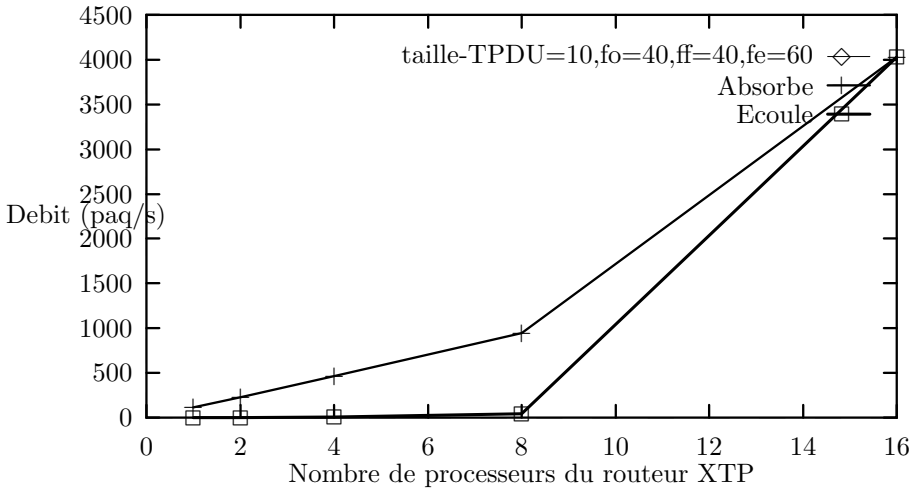

Figure 6 : Différence entre débit absorbé et débit écoulé par le routeur XTP (trafic moyen)

donné, il semble exister un nombre de processeurs en dessous duquel le trafic écoulé est faible (le routeur est débordé), alors qu'au dessus de ce nombre, le trafic est fluide et le fait de rajouter des processeurs ne change rien.

Pour confirmer cette hypothèse, nous avons fait tracer les différences entre débits absorbés et débits écoulés par le routeur XTP pour différents types de trafic (faible, moyen, fort, voir figure 6). Nous constatons qu'effectivement, il existe un nombre de processeurs à partir duquel le trafic écoulé est égal au trafic absorbé. Cet écroulement brutal peut s'expliquer par notre modélisation Estelle du routeur. En effet, une transition Estelle suffit au routeur pour absorber un nouveau paquet XTP émanant de l'environnement, alors qu'il lui en faut plusieurs pour le traiter et le ré-émettre. Comme les transitions sont atomiques et équi-probables, ce déséquilibre explique le fait qu'un routeur débordé absorbe plus qu'il ne peut émettre, jusqu'à saturer sa mémoire, ce qui arrive rapidement (malgré 4Mo de mémoire sur chaque processeur) pour les échelles de temps que nous considérons dans nos expériences. Le ré-équilibrage entre réceptions et émissions est un problème important qui devrait être réglé en cas d'implantation réelle (car il permet un fonctionnement en mode dégradé), mais que nous ne traitons pas ici car il ne change pas radicalement la capacité conceptuelle du routeur.

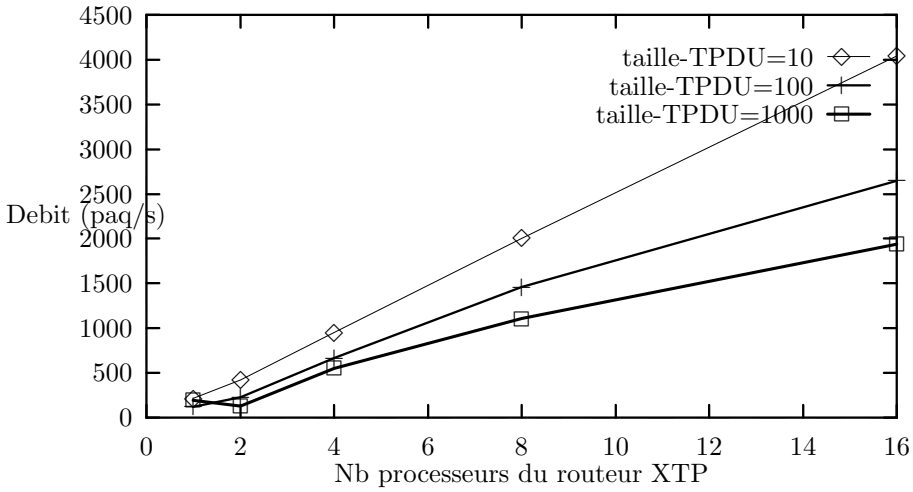

Figure 7 : Meilleurs débits utiles écoulés constatés sur de nombreuses expériences

Par contre il semble à première vue surprenant que le nombre de paquets XTP qui a pu être émis par les EXTP puisse dépendre (et même linéairement en figure 6) du nombre de processeurs du routeur : en effet nous avons précisé ci-dessus que nos expériences se faisaient à environnement statistiquement constant pour chaque point d'une même courbe. Il semble donc au contraire qu'en fait l'environnement soit "asservi" d'une manière ou d'une autre par le routeur, ne serait-ce que parce qu'un routeur à 16 processeurs dispose de 16 fois plus de mémoire (et donc de tampons pour stocker les paquets XTP entrants) qu'un routeur mono-processeur d'une part, et d'autre part parce qu'un routeur débordé induit des retards sur l'environnement lorsque des synchronisations entre EXTP sont nécessaires (à la fermeture d'une connexion par exemple).

Il paraît donc illusoire de vouloir travailler à environnement statistiquement constant : un examen détaillé des nombres moyens de communications XTP par EXTP et des nombres moyens de paquets DATA par communication montre d'ailleurs que ceux-ci varient de manière non aisément prédictible lorsque le trafic n'est pas fluide.

Pour conclure ces résultats, nous présentons en figure 7 les meilleurs débits utiles écoulés que nous avons constatés pour trois tailles de paquets XTP (10, 100 et 1000 octets) lors des milliers d'expériences que nous avons menées dans

des zones de trafic très variables.

Nous remarquons tout d'abord que les meilleurs débits utiles écoulés par le routeur augmentent quasiment proportionnellement à son nombre de processeurs. C'est un résultat extrêmement positif, car en l'extrapolant il signifie que notre routeur semble pouvoir absorber n'importe quel trafic (et ce sans en changer la moindre ligne de logiciel), pourvu qu'il dispose de suffisamment de processeurs. Ceci est confirmé par le fait que le nombre de paquets traités par unité de temps dépend assez peu de la taille de ces paquets : on traite environ seulement deux fois moins vite des paquets de taille 1000 que de paquets de tailles 10 dès que le nombre de processeur est suffisant. Ce ne sont donc pas les débits inter-processeurs qui sont limitatifs, mais bien le temps de traitement du protocole, d'où le gain apporté par le parallélisme de connexion.

Enfin, nous pouvons remarquer que le débit de croisière en situation vraisemblable (sans doute bien inférieur à un débit crête) écoulé par notre routeur à 16 processeurs est au moins égal à 16Mbits/s (2000 paquets de 1ko), soit en moyenne 1Mbits/s utile d'informations de niveau transport dans chaque sens de chaque "canal XTP" reliant un EXTP au routeur. Ceci est à comparer au débit maximum atteignable en monodirectionnel entre deux processeurs voisins de l'hypercube (sans autre charge, avec des messages de 1Mo) de environ 16Mbits/s : malgré le niveau élevé du langage utilisé (Estelle) et notre limitation à des conditions vraisemblables de fonctionnement de l'implantation, nos résultats d'expérimentation restent significatifs compte tenu de la technologie utilisée.

5 Conclusion

Nous avons montré dans cette étude l'intérêt et la faisabilité d'utiliser des machines parallèles à mémoire distribuée à usage général ayant des systèmes d'exploitation relativement standards pour réaliser une fonction simple des réseaux haut débits, ici un routeur de niveau 3/4 pour le protocole XTP.

Nous nous sommes appuyés sur les outils généraux développés dans le projet PAMPA (méthodes et outils de parallélisation, de validation et de compilation de spécifications formelles), et nous avons mené une évaluation de performance d'un routeur XTP parallèle (parallélisme de connexions) par expérimentation en fonction du nombre de processeurs utilisé. Bien qu'il faille se garder de généraliser sans précaution nos résultats expérimentaux, il ressort que dans des hypothèses de trafic vraisemblables, les performances en terme de débit utile écoulé par notre modèle de routeur XTP augmentent proportionnellement au nombre de processeurs de celui-ci. Ceci illustre l'intérêt d'utiliser des architectures parallèles à mémoire distribuée pour réaliser ce type de serveur pour réseaux à haut débit ; car n'étant plus limité par le goulot d'étranglement du bus d'accès à une mémoire partagée, on peut envisager de traiter n'importe quel débit pourvu qu'on dispose de suffisamment de processeurs et de liens de communications internes entre ces processeurs. Pour pouvoir être plus affirmatif,

il faudrait d'une part faire de nouvelles expériences avec des trafics déterministes afin de déterminer les débits crêtes de nos routeurs, et d'autre part se placer dans des cadres d'implantations plus réalistes sur des architectures plus modernes et plus performantes.

References

[1] Françoise André, Claude Jard, Jean-Louis Pazat, and Henry Thomas. Data-driven distribution of programs. In *International Workshop on Compiler for Parallel Computers*, 1990.

[2] T. Braun and M. Zitterbart. Parallel xtp-impementation on transputers. In *THE 1991 SINGAPORE INTERNATIONAL CONFERENCE ON NETWORKS*, pages 91–96, G.S.Poo, Sep 1991.

[3] David Callahan and Ken Kennedy. Compiling programs for distributed-memory multiprocessors. *The Journal of Supercomputing*, 2(2):151–169, 1988.

[4] Protocol Engine. The evolution of xtp. In *Proc. of the Third International Conference on High Speed Networking*, North Holland, 1991.

[5] M. Heath. The hypercube: a tutorial overview. In Michael T. Heath, editor, *Hypercube Multiprocessors 1986*, pages 7–11, Oak Ridge National Laboratory, 1986.

[6] ISO 9074. *Estelle: a Formal Description Technique based on an Extented State Transition Model*. ISO TC97/SC21/WG6.1, 1989.

[7] C. Jard and J.-M. Jézéquel. ECHIDNA, an Estelle-compiler to prototype protocols on distributed computers. *Concurrency Practice and Experience*, 4(5):377–397, August 1992.

[8] C. Jard and J.-M. Jézéquel. *L'expérimentation d'algorithmes distribués sur machines parallèles avec ECHIDNA*. Technical Report 603, IRISA/CNRS, Septembre 1991.

[9] J.-M. Jézéquel, T. Estimbre, and C. Jard. *Etude de l'utilisation de machines parallèles à mémoire distribuée pour la réalisation de commutateurs de réseaux haut débit*. Technical Report 693, IRISA/CNRS, Decembre 1992.

[10] Protocol Engine Project. *XTP Draft, revision 3.6*. 1992.

[11] H. Zimmermann. OSI reference model of architectures for open systems interconnection. *IEEE Transactions on Communications*, COM-29, April 1980.