Projet ILIAD :
Impact des liaisons séries haut débit sur l'architecture des machines parallèles

IRIT, équipe M3S,
IRISA équipe CALCPAR,
IEF département AXIS,
IOTA,
MASI équipe CAO et VLSI,
LGI groupe GRAM,
Groupe Rumeur de PRS

11 mars 1996

Programme de recherches Inter-PRC 93

Thème : Nouvelles technologies pour l'Architecture

Rapport Final

Ce document a été rédigé par les personnes suivantes :

- G. Berger Sabbatel (LSR/GRAM, anciennement LGI/GRAM),
- P. Sainrat (IRIT),
- A. Seznec, Y. Jégou (IRISA).
- P. Garda, E.Belhaire, P.Lalanne, P.Chavel (IEF-IOTA),
- B. Zerrouk, A. Derieux (MASI),
- Denis Trystram (LMC-IMAG), Michel Syska (I3S - Sophia Antipolis), Philippe Michallon (ETCA/CREA/SP),

Table des matières

1 Introduction

Dans le domaine du calcul parallèle, il est d'usage d'opposer deux types d'architecture : les systèmes distribués sur réseaux locaux, et les machines parallèles. Cette dichotomie semble cependant susceptible d'être remise en question.

La technologie des réseaux haut débit - dont l'ATM - fait entrevoir la possibilité d'implémenter des interfaces de communication dont les performances - et en particulier la latence d'application à application - soient proches de celles des réseaux d'interconnexion des machines parallèles. Ainsi, le projet NoW, de Berkeley, se propose d'implémenter des environnements de calcul parallèle sur réseaux de stations de travail dont les performances soient du même ordre que celles des machines parallèles pour un coût presque nul, puisqu'il s'agit d'utiliser la puissance de calcul inutilisée sur les stations de travail.

On rencontre par ailleurs la démarche inverse avec les technologies de réseau Myrinet ou S-connect : l'utilisation des technologies d'interconnexion des machines parallèles pour l'implémentation de réseaux locaux à haut débit.

On voit ainsi apparaître une assez large gamme de solutions pour le calcul à hautes performances : stations de travail, multiprocesseurs, super-calculateurs, machines massivement parallèles... Ces différents éléments peuvent être inter-connectés par des réseaux à haute performance pour composer des systèmes parallèles de tailles et de puissances arbitraires.

Un certain nombre de technologies émergentes sont donc susceptibles d'avoir une incidence importante sur l'architecture de tels systèmes, parmi lesquels nous noterons :

- Les technologies de communication à haut débit, qui offrent différentes solutions pour les interconnexions : routeurs rapides, transmissions série à haut débit, interconnexions optiques, protocoles de transmission à haut débits tels que l'ATM,
- Les architectures de processeurs intégrés, qui peuvent permettre de réaliser des systèmes multiprocesseurs plus efficaces (tolérance des latences mémoires élevées),
- Les architectures des hiérarchies mémoires : caches, mémoires multi-ports séries...

Ces technologies ont, à divers titres, fait l'objet d'études de la part des équipes participant au projet ILIAD. L'objectif scientifique du projet est donc de déterminer ce que ces technologies peuvent apporter à l'architecture des systèmes parallèles, et de voir dans quelle mesure leur utilisation combinée (lors-qu'elle est possible) permet d'améliorer les performances de ces systèmes, leur modularité, et leur capacité de communications avec l'extérieur, ou de dégager de nouvelles solutions pour le calcul à hautes performances.

2 Cadre de l'étude

La figure 1 donne le schéma général d'un système parallèle. Celui-ci est composé de groupes locaux, inter-connectés sur un réseau. Chaque groupe local comporte un ou plusieurs processeurs, de la mémoire, un interface de communication avec le réseau d'interconnexion (PC), et éventuellement des périphériques. Il existe par ailleurs, comme nous l'avons déjà mentionné, de nombreuses solutions possibles pour l'implémentation du réseau d'interconnexion.

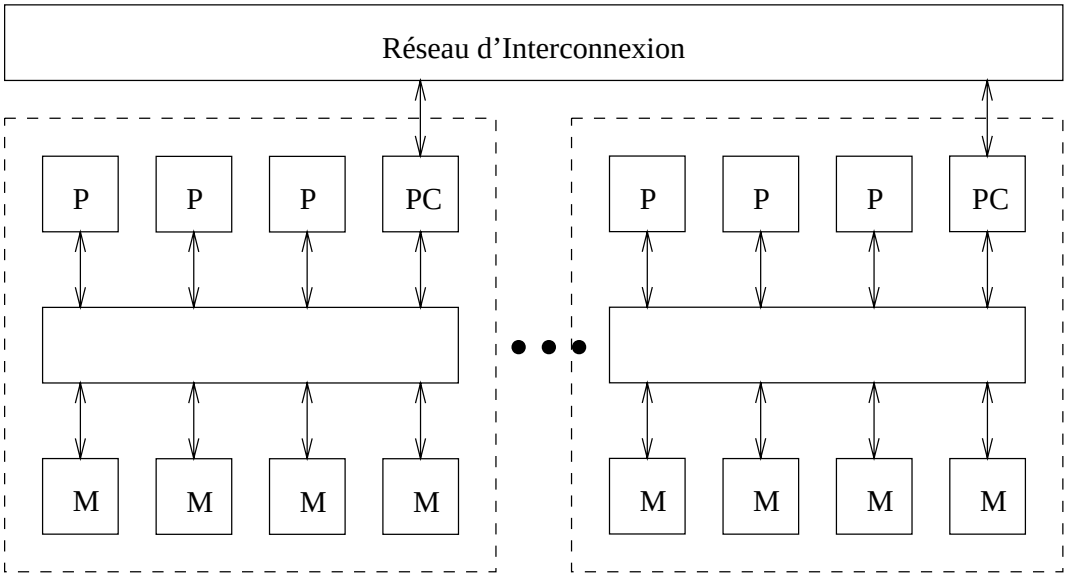


FIG. 1 – *Système parallèle*

2.1 L'architecture des groupes locaux

Les groupes locaux peuvent être d'architectures très diverses. Jusqu'ici, dans la plupart des machines parallèles, ce sont de simples noeuds uni-processeurs, comportant éventuellement un coprocesseur vectoriel. L'interface de communication avec le réseau peut être un simple interface passif, dont les registres de commande sont mappés dans l'espace d'adressage du processeur (comme sur la CM5 ou l'IBM SP1), ou être muni d'un processeur (comme sur la Paragon ou l'IBM SP2). Dans la SP2, les noeuds de traitement sont des stations de travail monoprocesseurs complètes, avec leurs périphériques. Dans la machine DASH, de Stanford, les groupes locaux sont des stations de travail quadri processeurs, munies d'un interface de communication permettant d'implémenter une mémoire distribuée partagée par cohérence de caches.

Les architectures de machines multiprocesseurs sont très diverses, en particulier au niveau du système d'interconnexion entre les processeurs et les mémoires. Le bus partagé reste la solution dominante, même si cette solution est limitée, de par le partage de la bande passante du bus, à des configurations de quelques processeurs: l'optimum semble de situer entre 4 et 8, avec des variations en fonction de la technologie.

Pour des configurations plus importantes, on trouve des cross-bars, ou combinaisons de cross-bars. Enfin, le projet M3S, de l'IRIT, a proposé une alternative, avec l'utilisation de liens série en points à points entre les mémoires et les processeurs. D'après les simulations effectuées, cette solution permettrait des configurations de plusieurs centaines de processeurs. Deux variantes sont possibles : soit les liens série servent à la fois à la transmission des adresses et des données, soit ils sont limités à la transmission des données, les adresses restant transmises par le biais d'un bus commun. Cette seconde solution permet de profiter des avantages de la diffusion des adresses pour le maintien de la cohérence des caches, mais ne permet pas des configurations aussi importantes que la première.

L'augmentation des fréquences d'horloge des processeurs est beaucoup plus rapide

que la diminution des temps d'accès aux mémoires RAM dynamiques (pour ces dernières, les progrès vont plutôt dans le sens de la capacité). Avec la mémoire distribuée partagée par cohérence de caches, les latences peuvent être de l'ordre de la microseconde. L'utilisation de plusieurs niveaux de caches devient donc incontournable (deux niveaux sur la plupart des machines actuelles), et le temps de traitement des défauts de caches devient tel qu'il apparaît nécessaire de le masquer, en le recouvrant par un traitement utile non affecté par le défaut de cache. On fait donc appel à des techniques telles que le désordre d'exécution, le préchargement, ou le support de flots d'instructions multiples. Les mémoires caches elles mêmes peuvent être rendues plus efficaces par l'utilisation des techniques telles que les caches "skewed set-associatives", ou "semi unifiés" étudiées dans l'équipe CAPS de l'IRISA.

2.2 Réseaux d'interconnexion

Dans le modèle de système parallèle présenté ci-dessus, on peut distinguer deux niveaux de réseau d'interconnexion : le réseau interne des groupes locaux, et le réseau d'interconnexion entre groupes locaux. Ce dernier niveau offre un assez large éventail de solutions, depuis les techniques utilisées dans les réseaux locaux, jusqu'aux techniques plus spécialisées dédiées généralement aux machines massivement parallèles.

Au niveau de la communication entre les groupes locaux, l'utilisation d'un protocole standardisé tel que l'ATM semble intéressante, et a fait l'objet des travaux du groupe GRAM du LGI. L'ATM, outre le fait qu'il s'impose comme standard de communication entre ordinateurs, et outre ses performances, apporte un certain nombre de propriétés utiles au transport de données multi-média. L'utilisation de l'ATM dans les réseaux locaux est susceptible d'apporter un gain de performances sensible par rapport aux technologies de réseau les plus courantes aujourd'hui. Mais les limites des possibilités des réseaux ATM pour le calcul parallèle à grain fin restent à déterminer.

Une autre approche est celle étudiée dans l'équipe CAO et VLSI du MASI : l'utilisation de routeurs rapides reconfigurables, implémentables en VLSI. Le routeur Rcube, conçu par cette équipe utilise le routage par intervalles, permettant un routage adaptatif avec des tables de routage compactes, et permet d'atteindre des débits de 2 Mégabits/s.

La technologie des transmissions sur liens séries rapides optiques ou électriques a été étudiée d'une part par les équipes de l'IEF et de l'IOTA, et d'autre part par Bull. En ce qui concerne l'optique, son domaine d'utilisation théorique permettrait de l'utiliser à l'intérieur des groupes locaux, par exemple pour les liaisons processeurs-mémoires. L'utilisation de l'optique peut apporter un meilleur bilan énergétique, et permettre d'atteindre des débits plus élevés, tout en étant relativement indépendant de la distance. Il s'agit cependant de la limite du domaine d'utilisation de l'optique, dans la mesure où des solutions économiquement abordables sont à définir. Par ailleurs, si l'impact quantitatif de l'optique semble clair, il reste difficile de déterminer si les technologies optiques permettent d'envisager des solutions architecturales nouvelles.

Enfin, à partir des propriétés topologiques des réseaux d'interconnexion, et des performances de liens de communication, la modélisation permet de prédire les performances du système. L'étude des outils théoriques pour cette modélisation fait l'objet de la contribution du groupe Rumeur.

2.3 Modèles de programmation

Les modèles de programmation des systèmes parallèles se répartissent en deux grandes catégories : la mémoire partagée, et la communication par messages. Si la communication par mémoire partagée est souvent plus intuitive, la communication par messages s'impose dans certains cas, et est souvent plus efficace. L'idéal est donc d'intégrer les deux.

Dans les machines à mémoire distribuée, la mémoire partagée peut être émulée par le biais de la cohérence de caches, ou par le biais de la mémoire virtuelle répartie. Ces deux modèles demandent généralement, pour être efficaces, que le maintien de cohérence ne soit pas aussi strict que pourrait le permettre une vraie mémoire partagée : on parle donc de cohérence faible, ou au relâchement, la cohérence n'étant garantie qu'en certains points, en fonction des synchronisations du programme. Ces modèles sont d'ailleurs également appliqués pour améliorer l'efficacité des caches dans les multiprocesseurs à mémoire commune.

En ce qui concerne la communication par messages, deux standard coexistent : PVM, standard de fait actuel, et MPI, standard émergent.

L'ensemble de ces points a fait plus particulièrement l'objet de travaux de la part du LGI et de Rumeur.

3 Mémoires multiports séries intégrés

3.1 Problématique

L'évolution actuelle et prévisible de la technologie montre un accroissement régulier et important en matière d'intégration mais aussi de fréquence de fonctionnement. Ceci conduit à des processeurs ayant des fréquences de fonctionnement de plus en plus grandes (275 MHz pour le microprocesseur Alpha) qui, de plus, sont maintenant capables de traiter plusieurs instructions simultanément, quatre aujourd'hui voire 6 et très probablement huit demain.

La conséquence directe de cette évolution est un accroissement de la bande passante nécessaire pour les accès mémoire renforcé par le fait que les processeurs chargent aujourd'hui plusieurs instructions simultanément. En environnement monoprocesseur, les bus actuels permettent de répondre à cette demande mais, en environnement multiprocesseurs, la bande passante du réseau et de la mémoire deviennent des facteurs essentiels de l'architecture et du coût de la machine.

Une autre conséquence de l'accroissement de la fréquence de fonctionnement est le coût temporel des accès à la mémoire centrale. Les performances des mémoires RAM dynamiques ne suivent pas la même évolution que pour les processeurs. Les temps d'accès à ces mémoires diminuent, certes, mais peu comparé au temps de cycle des processeurs. Il en résulte des temps de latence (temps d'accès à la mémoire plus temps de transfert sur le réseau) qui pénalisent fortement le processeur.

3.2 État de l'art

Le point de passage obligé aujourd'hui est un élargissement de la hiérarchie mémoire afin de diminuer la latence moyenne des instructions. C'est ainsi qu'une hiérarchie de

la mémoire possédant deux niveaux de caches est aujourd'hui le cas le plus courant et l'on parle maintenant de trois niveaux.

Le premier niveau de cache est situé sur la puce et est, le plus souvent divisé en deux caches, un pour les instructions et un pour les données. Sa caractéristique majeure est de fournir une voire plusieurs données en un cycle processeur. Il est très petit (quelques Kilo-octets) pour trois raisons. D'une part, étant sur la même puce que le processeur, la surface de silicium est limitée. D'autre part, il est réalisé en mémoire statique pour respecter un temps d'accès inférieur au cycle du processeur, ce qui limite sa taille pour des considérations de dissipation thermique. Enfin, et c'est la principale raison, afin de fournir un accès en un cycle, la taille de la mémoire doit être restreinte pour que les temps passés dans les mécanismes d'adressage soient suffisamment courts.

Le deuxième niveau de cache est de taille plus importante (de 1 à quelques Méga-octets) mais son temps d'accès est de plusieurs cycles processeurs.

Enfin, un niveau intermédiaire peut être envisagé d'une centaine de kilo-octets et répondant en quelques cycles. En fait, ce niveau supplémentaire dépend de la taille du cache de premier niveau. Il est nécessaire lorsque le cache de premier niveau n'est pas suffisant. Il peut être également interne comme dans l'Alpha du fait de sa grande fréquence de fonctionnement.

Ces mémoires caches diminuent la bande passante mémoire nécessaire en profitant de la localité spatiale et temporelle du code produit par les compilateurs. Elles diminuent aussi la latence moyenne des instructions puisque les données sont souvent présentes dans la mémoire cache. Toute solution pour améliorer les taux de présence des données dans les caches est donc intéressante (voir la partie sur les skewed-associative caches de l'IRISA). Toutefois, en environnement multiprocesseurs, même en présence de mémoires caches, les bandes passantes du réseau d'interconnexion et des mémoires centrales ne sont pas suffisantes compte tenu des progrès des microprocesseurs. D'autre part, en environnement monoprocesseur, le coût des mémoires caches est suffisamment important pour rechercher une solution au niveau de la mémoire centrale qui soit plus rentable.

Quelques efforts sont faits en ce sens par les constructeurs et nous proposons une solution originale qui allie une structure de mémoire et de réseau d'interconnexion pour les processeurs à mémoire partagée.

3.3 Solutions

Les deux problèmes majeurs auxquels on doit faire face sont donc la bande passante et la latence de la mémoire et du réseau d'interconnexion.

3.3.1 La latence d'accès aux données

Comme nous l'avons déjà dit, la présence de mémoire caches réduit la latence moyenne des instructions. Mais l'accès à la mémoire reste fortement pénalisant. Or, en environnement multiprocesseurs, ces accès sont plus fréquents du fait des données invalidées dans les caches par l'algorithme de cohérence des données.

Les conflits, lors d'accès aux données, se situent à deux niveaux : le réseau d'interconnexion et la mémoire. Le plus souvent, le réseau des machines à mémoire partagée est un bus partagé, économique certes mais source maximale de conflits. De plus, si le bus est calibré pour un nombre de processeurs N (rarement plus d'une quinzaine), son

coût n'est rentabilisé que pour les configurations à N processeurs. Le bus est sur-calibré pour de petites configurations et son coût représente la majeure partie du coût d'une petite configuration la rendant par là même inintéressante. Enfin, d'un point de vue technologique, on ne sait pas réaliser de bus pour un grand nombre de processeurs. De nombreuses autres approches ont été proposées, que ce soit des réseaux statiques ou dynamiques, présentant des caractéristiques plus ou moins intéressantes que ce soit en termes de "scalabilité", de performances... Aucune n'a réellement émergé.

En ce qui concerne la mémoire, la réduction des conflits passe par un découpage de celle-ci en bancs (ou modules) entrelacés. Dans les super-calculateurs, la mémoire était entrelacée au niveau du mot. Dans les multiprocesseurs à mémoire partagée, celle-ci doit être entrelacée au niveau du bloc de données du fait de la présence des mémoires caches. Enfin, une solution très étudiée aujourd'hui qui peut être matérielle, logicielle ou une combinaison des deux consiste à pré-charger les données dans le cache avant quelles soient utilisées. En environnement multiprocesseurs, ce problème est loin d'être trivial du fait de la cohérence des données. En effet, les données ne doivent pas être chargées trop tôt sous peine d'être invalidées avant leur utilisation créant ainsi un accès mémoire inutile et un message d'invalidation. Il s'en suit une demande en bande passante supplémentaire. Enfin, si le préchargement n'est pas fiable à 100%, il en résulte un besoin en bande passante supplémentaire. L'IRIT et l'IRISA travaillent sur le préchargement. Ce projet a permis de confronter nos vues en ce domaine et permettra peut-être un travail conjoint à l'avenir sur ce sujet brûlant.

3.3.2 La bande passante

Différentes solutions existent aujourd'hui qui ont permis d'accroitre la bande passante des mémoires dynamiques (voir le rapport intermédiaire), elles consistent essentiellement à exploiter mieux le nombre important (de 1024 à 8192) de bits d'une ligne de DRAM par exemple en recopiant lors d'un accès à un mot toute la ligne correspondante dans une SRAM. Toutefois, ces solutions ne sont pas suffisantes. En effet, si l'on suppose un temps de cycle de 120 ns et une ligne de 1024 bits, on atteint une bande passante de 8 Gbit/s. C'est nettement mieux que des DRAM classiques qui atteignent péniblement 20 Mbits/s. Toutefois, ce n'est pas suffisant pour un multiproceseur.

L'accroissement de la bande passante de la mémoire passe aussi par un découpage de celle-ci en modules, c'est même la principale raison de ce découpage, la réduction des conflits étant une conséquence. L'inconvénient de ce découpage est que le réseau s'en trouve plus complexe puisqu'il faut autant de points d'accès au réseau qu'il y a de modules mémoire. Chaque point d'accès étant un bus parallèle de 32 ou 64 bits, il en résulte une logique qui doit être très intégrée et des problèmes d'interconnexion du fait du nombre de fils et de points de connexions sur les commutateurs. C'est pourquoi nous proposons l'utilisation de liens série à très haut débit qui réduisent énormément le nombre de points d'interconnexion. Ces liens peuvent conserver la même bande passante que des bus parallèles car l'aspect point à point et asynchrone permet de les rendre ultra-rapides. De tels liens existent aujourd'hui. Il n'en reste pas moins que la mémoire doit présenter une structure telle qu'elle permette la réception de requêtes venant des différents liens série et fournir les données aux processeurs en temps voulu. La mémoire doit donc présenter une structure multiport série.

3.3.3 La structure multiport série

Cette structure se base aussi sur l'utilisation optimale des lignes de bits. Outre la mémoire proprement dite (le tableau de cellules DRAM) et la ligne d'éléments d'amplification, le module comprend des registres de réception des requêtes (*reqn*), des registres de réception (*recn*) et d'émission (*emin*) des données, un arbitre des requêtes et une logique de séquencement. La taille des données transférées est celle d'un bloc de cache (quelques octets). Une ligne de mémoire contient donc plusieurs blocs. Contrairement aux autres circuits, ce composant peut servir simultanément plusieurs requêtes, les temps d'accès à la mémoire étant recouverts avec les temps de transfert des données. On suppose toutefois que les processeurs n'émettent qu'une requête à la fois attendant la réponse à une requête avant d'en envoyer une autre. Pour surpasser cette limitation, les registres devraient être remplacées par des files, ce qui est plus coûteux en termes de surface de silicium.

Les résultats de simulation analytique de ce composant ont montré que pour des liaisons à 500 Mbits/s, c'est à dire aisément réalisable en CMOS aujourd'hui, un minimum de 8 ports est nécessaire pour avoir un débit identique à une DRAM classique et un temps de réponse inférieur. Avec une liaison à 2,5 Gbit/s (existant déjà en AsGa et que l'on saura réaliser dans un futur proche en BiCMOS), cette solution est meilleure dès qu'il y a deux ports série à la fois en temps de réponse et en bande passante. Cette solution est donc très intéressante puisqu'elle fournit une bande passante supérieure et un temps de latence inférieur. Il reste toutefois à quantifier son prix. En effet, les liaisons à 2,5 Gbit/s demandent une dissipation thermique importante réclamant des boîtiers en céramique ou des modules multi-puces dont le prix commence seulement à baisser.

3.4 Résultats

La structure des mémoires multiports série a été étudiée sur le logiciel de CAO Cadence et les performances ont été évaluées à l'aide d'un modèle analytique résolu par un réseau de files d'attente. Ce type d'évaluation a l'avantage de constituer une évaluation grossière pour un investissement humain pas trop important. Toutefois, il n'est pas suffisamment crédible en soi et demande des simulations complémentaires. Un simulateur de machine multiprocesseurs est actuellement en cours d'écriture. Ce simulateur est modulaire afin de pouvoir faire varier différents paramètres de l'architecture et de pouvoir étudier, par exemple, l'impact de la présence de mémoires caches dans un environnement multiprocesseurs. De plus, ce simulateur est dirigé par le programme afin de s'abstraire des distorsion dues à la récolte de traces sur une machine différente de la machine à simuler. Enfin, un noyau de système existe et est simulé. Ceci permet une simulation complète des programmes et une vérification des résultats fournis par le simulateur et donc une preuve du bon fonctionnement de celui-ci. Actuellement, ce simulateur fonctionne en environnement monoprocesseur avec des caches traditionnels. Des caches non bloquants sont en cours de test. En ce qui concerne la version multiprocesseurs, la structure globale est en place y compris au niveau système.

4 Structures de caches, processeurs tolérant les latences mémoire élevées

Dans le projet ILIAD, l'équipe CAPS de l'IRISA étudie les mécanismes qui permettraient d'améliorer le comportement d'une architecture multiprocesseurs en la rendant moins sensible aux latences nécessairement élevées des accès à une mémoire commune. Cette contribution est complémentaire de celle de l'équipe architecture multiprocesseurs de l'IRIT qui adresse le problème du débit de la mémoire dans la machine M3S.

Pour être moins sensible aux phases peu parallèles des algorithmes, un calculateur parallèle doit être construit à partir de processeurs les plus puissants possibles. Si les progrès technologiques ont permis un accroissement soutenu des performances des processeurs pendant ces dix dernières années, sur les mémoires, ces même progrès ont surtout apporté des gains de capacité ; mais les gains dans leurs vitesses de commutation sont restés faibles. Sur les microprocesseurs modernes, le service d'une requête à la mémoire principale peut correspondre à l'exécution de plusieurs dizaines d'instructions. Certaines architectures parallèles permettent l'accès de l'ensemble des mémoires à chaque processeur. Mais les délais de traversée des réseaux d'interconnexion font qu'il faut prévoir plusieurs centaines de cycles pour des accès non locaux. D'autres architectures parallèles limitent l'espace adressable directement par un processeur à la seule mémoire locale. Les accès distants doivent alors être traités par logiciel. L'insertion de cette couche de traitement supplémentaire dans l'accès aux données entraîne une latence très élevée.

Plusieurs solutions ont été imaginées pour faire en sorte que l'émission d'une requête sur la mémoire n'entraîne pas une inactivité systématique du processeur. Ces solutions visent à traiter le problème de la latence au niveau de

- l'architecture du processeur : mémoires cache, préchargement, découplage, désordre d'exécution, multiples flots d'instructions,
- des exécutifs : flots d'instructions multiples,
- de la compilation : préchargement, ordonnancement d'instructions, multi-tâches

4.1 Mémoires cache

Les mémoires cache cherchent à diminuer la latence pour l'accès à des adresses qui présentent certains types de localité. La localité temporelle est issue de la réutilisation d'une même adresse mémoire dans un délai assez faible. Les réutilisations à travers un cache évitent de repayer la latence mémoire du premier accès. La localité spatiale provient en général du parcours des structures de données en mémoire. Ce parcours entraîne des accès à des adresses voisines. Dans ce cas, l'accès à une donnée provoque le préchargement automatique de données voisines dans le cache. La latence de l'accès à ces données voisines est réduite.

Le fonctionnement des mémoires cache est basé sur un principe de base simple : placer les données les plus utilisées dans une mémoire plus rapide et plus proche du processeur. Mais les caches ont une capacité limitée, le chargement d'une donnée entraîne nécessairement l'élimination d'une autre donnée. Dans les architectures multiprocesseurs à mémoire commune, les accès aux premiers niveaux de cache sont purement

locaux et permettent donc un gain supplémentaire en latence et en débit. Mais il faut intégrer des protocoles de maintien de cohérence entre les caches locaux des processeurs.

Au cours de deux dernières années, les travaux ont porté sur les caches dits "skewed-associatifs". Il a été en particulier montré que ces caches ont un comportement indépendant du placement précis des données en mémoire, a contrario des caches conventionnels dont le comportement est particulièrement erratique (variation de 1 à 5 du temps d'exécution en fonction du placement des matrices pour une multiplication de matrices, par exemple).

Le comportement des caches est fortement influencé par le choix de la longueur de ligne. A volume de cache constant, l'exploitation de la localité spatiale est favorisée par un allongement des lignes alors que la localité temporelle se comporte mieux avec des lignes plus courtes. La structuration des caches en lignes virtuelles permettrait de profiter des avantages des lignes longues (effet de préchargement, augmentation de la localité spatiale) tout en évitant les phénomènes de pollution. Des gains allant de 17 à 64% ont été observés sur des *benchmarks* classiques.

4.2 Préchargement

Les défauts de cache initiaux sont dus au premier accès à une adresse en mémoire. Aucune optimisation basée sur la conservation de données accédées dans le passé ne peut permettre d'améliorer le comportement du cache sur de tels accès : il faut faire appel à des mécanismes de prédiction. La prédiction la plus simple, et utilisée sur la plupart des caches, consiste à charger des données d'adresses voisines lors d'un défaut de cache. Mais cette exploitation de la localité spatiale ne suffit pas toujours. D'autres mécanismes ont été étudiés qui cherchent des régularités dans les adresses émises pour faire des prédictions (préchargement spéculatif). Des instructions de préchargement (non bloquantes) peuvent également être exploitées par les compilateurs.

Nous avons montré qu'il est possible de détecter automatiquement le parcours de séquences d'adresses en progression arithmétiques et de déclencher le préchargement de ces données.

De tels ajouts dans les processeurs sont gourmands en place sur le composant, nous avons aussi étudié des structures d'étiquettes pour les caches moins gourmandes en place ("Decoupled Sectored Caches").

4.3 Découplage, ordonnancement d'instructions

Pour ce qui concerne la limitation de l'effet des latences mémoire, le découplage de l'exécution et des accès à la mémoire est une technique matérielle de préchargement des données qui n'est pas basée sur l'utilisation d'un cache. Cette technique a pour effet de provoquer, dynamiquement à l'exécution, un décalage entre l'exécution des instructions provoquant des lectures mémoire et l'exécution des instructions consommant les données lues par répartition de ces instructions sur deux unités d'exécution découplées.

Les techniques d'ordonnancement des instructions à la compilation comme le pipeline logiciel (*software pipeline*) cherchent à provoquer un tel déphasage dès la génération du code.

4.4 Le désordre d'exécution

Les nouveaux microprocesseurs intègrent des mécanismes d'exécution dans le désordre des instructions. Ceci permet de continuer l'exécution d'un programme bien que des instructions antérieures ne sont pas achevées, en particulier l'accès à la mémoire. L'un des objectifs de l'étude est de voir quel niveau de latence mémoire ces mécanismes permettent de masquer.

4.5 Les flots d'instructions multiples

La technique du *multithreading* matériel consiste à maintenir plusieurs flots d'instructions actifs dans un même processeur. Ces flots peuvent être issus d'une même application ou non. Ils peuvent être séquencés simultanément ou alternativement. Dans tous les cas, lorsqu'un flot est bloqué pour cause d'attente de réponse à la suite d'un accès mémoire, le séquencement des autre flots peut être poursuivi. Le "multithreading" peut être implémenté matériellement de plusieurs manières, soit en permettant une commutation rapide du flot en cours d'exécution, soit en sélectionnant les instructions à exécuter au cours du prochain cycle parmi plusieurs flots. Des expérimentions ont été menées sur un séquencement de type super-scalaire dont les instructions élues viendraient de plusieurs flots différents. Les premiers résultats de simulation montrent que l'introduction d'un deuxième flot d'instructions dans une architecture RISC classique permet de passer d'une durée moyenne de deux cycles par instruction (CPI) à 1,2 cycles par instruction. Mais cette technique doit être confrontée aux nouveaux processeurs super-scalaires.

Un simulateur multithread (simultané) est actuellement en cours de validation. L'objectif de cette étude est de montrer que les unités fonctionnelles présentes sur un processeur super-scalaire de large degré (8-16) peuvent être mieux exploitées par plusieurs processus que par un seul ; ces processus pouvant soit être issus d'un même père, soit être totalement indépendants

Une forme de multithreading logiciel peut aussi être mise en œuvre pour masquer les latences plus élevées (plusieurs centaines de cycles) dues aux échanges inter-processeurs dans une architecture parallèle. Ce type de changement de contexte ne peut bien entendu pas être mis en œuvre sur des défauts de cache, mais il permet de libérer le processeur au cours des accès distants.

5 Apports de l'optique

5.1 Introduction

Les performances maximales des machines parallèles résultent d'une part de leurs architectures logicielle et matérielle et d'autre part de leurs technologies de réalisation. Plus précisément, les technologies de réalisation déterminent les performances maximales des principaux éléments de l'architecture matérielle : processeurs, mémoire et réseau d'interconnexion.

En ce qui concerne les processeurs et la mémoire, les technologies de réalisation de circuits intégrés CMOS se sont progressivement imposées chez tous les constructeurs d'architectures parallèles. Ils progressent selon une loi géométrique énoncée par Moore

en 1974 et vérifiée au cours des 30 dernières années. Ils continueront à progresser au cours des dix prochaines années selon la même loi.

Cependant, les réseaux d'interconnexions constituent un goulot d'étranglement très important dans les architectures parallèles, et cette situation devrait être aggravée par les progrès des processeurs et de la mémoire. Cette situation conduit à envisager de nouvelles technologies pour réaliser les interconnexions. L'optique est la principale alternative aux technologies d'interconnexions électroniques. Elle s'est déjà imposée pour les transmissions à longue distance dans le domaine des télécommunications.

Dans le cadre du projet ILIAD, l'Institut d'Électronique Fondamentale (IEF) et l'Institut d'Optique Théorique et Appliquée (IOTA) ont étudié l'apport de l'optique à la réalisation des interconnexions dans les machines parallèles.

5.2 Utilisation de l'optique dans les architectures parallèles

Différentes utilisations de l'optique peuvent être envisagées pour réaliser les interconnexions dans les machines parallèles. D'une part, ces interconnexions peuvent se situer à différents niveaux dans la machine. D'autre part, ces interconnexions peuvent intervenir dans différentes architectures de machines parallèles.

L'utilisation de l'optique peut être envisagée à différents niveaux dans la réalisation d'une machine parallèle, selon la granularité des modules mis en communication. L'étude a porté sur la réalisation des interconnexions entre processeurs, routeurs ou bancs mémoire. Leur réalisation électronique se présente avec les densités d'intégration actuelles sous la forme de cartes, voire de modules multi-puces (MCM) ou de monopuces complexes. Deux autres domaines d'utilisation de l'optique pour des communications ont été écartés, bien qu'ils soient voisins du domaine étudié, parce qu'ils ne sont pas spécifiques aux machines parallèles. Il s'agit d'une part de la réalisation des connexions internes à une puce, qui n'est pas spécifique aux machines parallèles avec les densités d'intégration envisagées à court et moyen terme. Il s'agit d'autre part de la réalisation des communications à longue distance entre machines parallèles, qui concerne tout autant les architectures distribuées et les télécommunications.

L'utilisation de l'optique peut aussi être envisagée dans des architectures de machines parallèles qui diffèrent par leur organisation mémoire. En effet, il peut s'agir d'architectures à mémoire partagée ou à mémoire distribuée. Dans les architectures à mémoire partagée, les communications sont établies entre processeurs et bancs mémoire, et leurs caractéristiques ont un impact sur les performances des accès à la mémoire effectués par les processeurs. Dans les architectures à mémoire distribuée, les communications sont établies entre processeurs ou routeurs, et leurs caractéristiques ont un impact sur les performances des échanges de messages effectués par les processeurs. Ces deux architectures ont été envisagées dans le cadre du projet ILIAD, la première étant représentée par la machine M3S de l'IRIT et la seconde par la machine MPC basée sur le routeur RCUBE du MASI.

L'utilisation de l'optique peut enfin être envisagée dans des architectures de machines parallèles qui diffèrent par leur réseau d'interconnexion, qui peut être un bus, un réseau multiétage ou un réseau maillé.

La communication par un bus peut être réalisée de façon électronique ou optique. Cependant, les réalisations électroniques de bus ont été analysées exhaustivement dans le cadre du projet M3S de l'IRIT, qui a étudié leurs caractéristiques et leurs évolutions.

Par ailleurs, les réalisations optiques de bus ont été étudiées par la communauté des opticiens, qui a montré la faisabilité scientifique et technique de bus optiques, en particulier dans le cadre du projet européen ESPRIT II OLIVES. La connaissance de ces travaux antérieurs a conduit à écarter la communication par un bus du domaine étudié dans le cadre du projet ILIAD.

Dans le cas des réseaux multiétages, l'ensemble du réseau peut être réalisé optiquement, et les topologies de réseau envisageables peuvent donc être très différentes de celles qui sont réalisées de façon électronique. Elles peuvent bénéficier des possibilités offertes par tous les composants optiques existants.

Dans le cas des réseaux maillés, le réseau est constitué de liaisons point-à-point, et l'optique n'intervient que dans la réalisation des liaisons physiques entre des noeuds qui sont pour leur part réalisés électroniquement. Les topologies de réseau envisageables sont donc identiques à celles qui sont entièrement réalisées de façon électronique. Dans ce cadre l'optique peut améliorer les performances des liaisons point-à-point.

L'apport de l'optique aux réseaux multiétages a été plus particulièrement étudié en collaboration avec RUMEUR dans le cadre du groupe de travail ROI, et les travaux effectués dans ce cadre sont présentés dans la section 8 et l'annexe 6. L'apport de l'optique aux réseaux maillés a été plus particulièrement étudié en collaboration entre IEF et IOTA, et les travaux effectués dans ce cadre sont présentés ci-après dans les sections 5.3 et 5.4, et dans l'annexe 3.

5.3 Impact de l'optique sur les liaisons point-à-point

5.3.1 Introduction

Différentes études ont été menées dans le cadre de ILIAD pour évaluer l'apport de l'optique aux liaisons point-à-point dans le cas de machines parallèles. Une première étude a permis de faire un état de l'art des liaisons électroniques et optiques à haut débit dans le cadre des machines parallèles existantes. Une seconde étude a porté sur la comparaison entre liaisons électroniques et optiques. Une dernière étude a été effectuée sur les interfaces entre électronique et optique et les perspectives de leur intégration.

Les liaisons point-à-point possèdent de très nombreuses caractéristiques. Les plus importantes sont celles qui permettent d'évaluer l'impact de la technologie sur l'architecture et les possibilités de débouché industriel aux avancées scientifiques. Elles peuvent être regroupées en quatre classes :

- caractéristiques architecturales : latence et débit;
- caractéristiques électroniques : consommation et fréquence de fonctionnement;
- caractéristiques signal : taux d'erreur et capacité;
- caractéristiques pratiques : distance maximale de communication et coût.

Il est malheureusement très difficile d'obtenir l'ensemble de ces informations pour chaque exemple de liaison. En fait il est principalement possible de déterminer les caractéristiques architecturales et les distances maximale de communication.

5.3.2 État de l'art des liaisons point-à-point électroniques et optiques

Un état de l'art détaillé des liaisons point-à-point électroniques, et optiques a été effectué, et il est présenté dans la section 2 de l'annexe 3.

Ces liaisons diffèrent par leurs origines et leurs caractéristiques (note sur les unités utilisées pour les débits : b/s pour bits/s, débit brut des trames ; o/s, pour octets/s, débit utile des données). Les compromis entre débit, distance maximale de transmission et support physique sont illustrés par les tables jointes dans l'annexe.

- SDH/SONET pour ATM, liaisons de 155,5 à 622 Mb/s, voire 2,4 Gb/s, distance maximale jusqu'à plusieurs milliers de km;
- HIPPI : liaison à 800 Mb/s, latence de quelques μs, distance maximale de 25 m sur câble et de 10000 m sur fibre optique;
- Fibre Channel : liaison de 132 à 1062 Mb/s, distance maximale de 50 m sur câble et de 10000 m sur fibre optique;
- SCI : liaisons à 1 Go/s sur câble et 1,25 Gb/s sur fibre optique, latence *entre processus utilisateur* de quelques μs; la norme inclut un protocole de cohérence de caches et elle est particulièrement bien adaptée aux architectures parallèles ;
- HIC : liaisons de 200 Mb/s à 1 Gb/s sur câble ou fibre optique, distance maximale de 8 m à 3000 m ; cette technologie développée par Bull est exploitée dans le routeur RCUBE du MASI présenté dans la section 6;
- Optobus : liaison optique parallèle à 1,5 Gb/s sur 10 fibres optiques, latence de quelques μs, distance maximale de 30 m;
- le réseau local à très haut débit Myrinet de Myricom utilise des liaisons sur câble pour des transferts bidirectionnels de 1,28 Gb/s jusqu'à 25 m ; au-delà de cette distance une paire de fibres optiques est utilisée pour les transmissions avec une interface spécifique ;
- le premier prototype du réseau de stations de travail NOW de l'Université de Berkeley était constitué de stations de travail reliées par FDDI.

Deux points ressortent de cette étude :

- l'optique est envisagée dans toutes les normes de liaisons pour des débits ou des distances élevés
- des liaisons à haut débit Fibre Channel et SCI sont déjà utilisées dans des ordinateurs parallèles industrialisés : serveurs multiprocesseurs Enterprise de HP, Exemplar SPP1200/XA de HP/Convex et NUMA-Q de Sequent.

5.3.3 Comparaison des liaisons point-à-point électroniques et optiques

Une comparaison détaillée des liaisons point-à-point électronique et optique a été effectuée, et elle est présentée dans la section 3 de l'annexe 3.

Les apports fondamentaux de l'optique ont été étudiés depuis plusieurs années, en particulier par les opticiens. Trois grandes classes de propriétés ont été étudiées.

Premièrement, tous les moyens physiques de propagation de l'information produisent une atténuation de l'onde transmise. Cette atténuation est proportionnelle à la distance de transmission. En général, cette atténuation dépend aussi de la fréquence de l'onde transmise. La relation entre l'atténuation et la fréquence dépend de la technologie de réalisation de la connexion.

D'un autre coté, la bonne réception du signal transmis impose une valeur maximale à l'atténuation totale introduite par une liaison. Ceci induit une relation entre la fréquence de l'onde transmise et la distance maximale de transmission, qui dépend du support physique de transmission.

Dans les technologies de réalisation électronique, l'atténuation augmente rapidement en fonction de la fréquence de l'onde transmise.

Au contraire, dans le cas de connexions optiques, l'atténuation est constante et indépendante de la fréquence de transmission. Des valeurs typiques de l'atténuation sont de l'ordre de 4 dB/km pour des fibres multimodes.

Cette première propriété de l'optique est la plus importante parce que c'est celle dont les conséquences pratiques apparaissent le plus rapidement. Elle explique que des communications à grande distance soient nécessairement optiques, dans le domaine des télécommunications. Elle explique aussi que des communications optiques à courte distance permettent des fréquences de transmission plus élevées que des communications électroniques. Ceci se traduit sur le plan architectural par un débit des transmissions plus élevé en optique qu'en électronique.

La seconde propriété est plus surprenante dans son énoncé comme dans ses conséquences. En effet les connexions optiques permettent la conception d'interfaces dont l'impédance est plus élevée que les liaisons électroniques. Les conséquences pratiques seront que les interfaces avec des liaisons à très haut débit seront plus faciles à concevoir avec des liaisons optiques qu'avec des liaisons électroniques et que leurs consommations seront plus faibles. Le gain est cependant très difficile à quantifier en l'absence de modèle suffisamment précis.

La troisième propriété tient à la nature du support des communications optiques : les communications optiques ont une plus grande capacité que les connexions électroniques.

5.3.4 Étude des interfaces

Au vu des avantages que présentent les interconnexions optiques, il est naturel de se demander pourquoi ces interconnexions ne sont pas davantage exploitées dans les ordinateurs.

La réponse tient naturellement à la nécessité d'interfaces électro-optiques et opto-électroniques qui assurent la transduction entre signaux électriques et optiques. Ces interfaces sont réalisées dans des technologies électroniques III-V différentes des technologies CMOS utilisées pour réaliser les processeurs et les routeurs. De ce fait leur utilisation implique un sur-coût qui limite la généralisation du recours à l'optique. De très nombreux travaux de recherche actuels visent à diminuer le coût de ces interfaces. Deux exemples sont présentés ci-dessous.

Des travaux sont effectués à IBM pour intégrer complètement des interfaces électro-optiques et opto-électroniques à des circuits de traitement, y compris les deux transduc-

teurs. Dans le premier cas, il s'agit de technologies III-V, dans le second de technologie BiCMOS. Les résultats sont plutôt satisfaisants en terme de performances, mais ils s'appuient sur des technologies qui sont marginales pour la réalisation de processeurs. Ceci diminue la portée pratique de ces études.

D'autres travaux sont menés à Université Catholique de Leuwen en liaison avec Alcatel. Ces travaux ont cette fois pour but d'intégrer tous les éléments des deux interfaces électro-optiques et opto-électroniques à des circuits de traitement, sauf les deux transducteurs. Cette fois l'intégration est réalisée dans une technologie CMOS avancée, ce qui présente l'avantage de permettre une intégration avec les processeurs ou les routeurs. L'inconvénient de cette approche est que les débits accessibles à court terme sont limités par les circuits de traitement à quelques centaines de Mb/s.

En ce qui nous concerne, nous avons mené des travaux destinés à évaluer les possibilités d'exploiter des cellules photo-sensibles réalisées dans des technologies CMOS numérique standard pour la réception des transmissions optiques parallèles. Ces travaux ont comporté une modélisation électronique des photo-détecteurs et la conception d'un prototype de validation. Les conclusions de cette étude sont que ces photo-détecteurs ont des fréquences maximales de fonctionnement trop faibles pour être utilisés utilement pour cette réception. De plus les évolutions des technologies de réalisation CMOS n'amélioreront pas ces fréquences maximales. Il n'y a donc pas d'espoir d'intégrer simplement des récepteurs à fréquence maximale de fonctionnement élevée dans des technologies CMOS standards.

5.4 Conclusions et perspectives

L'étude rapportée dans la section 5.3 a porté sur la comparaison des liaisons point à point optiques et électroniques du point de vue matériel dans la perspective de leur exploitation dans des architectures parallèles. Le débit des communications est la principale caractéristique de ces liaisons point à point pour laquelle l'optique est intrinsèquement supérieure à l'électronique. Cette supériorité peut se traduire concrètement de trois manières :

- les liaisons optiques permettent des débits de communication plus élevés que les liaisons électroniques à distance égale;
- les liaisons optiques permettent des distances de communication plus grandes que les liaisons électroniques à débit égal;
- les liaisons optiques sont le seul moyen de communication utilisable pour des distances de communication et des débits très élevés.

Par ailleurs, l'optique peut permettre de diminuer la consommation des interfaces avec des lignes à fréquence de transmission élevée, et de faciliter leur conception.

Cependant, un handicap majeur des liaisons optiques réside dans le fait que les technologies CMOS utilisées pour la réalisation des processeurs et des routeurs ne se prêtent pas à l'intégration complète des interfaces électro-optiques et opto-électroniques. Cette situation ne devrait pas changer malgré les progrès importants de ces technologies. L'étude de ce problème majeur est un axe de recherches futures très important dans le domaine des liaisons point à point à haut débit.

La latence des communications est l'autre caractéristique des liaisons optiques importante pour l'architecture des machines. Les latences matérielles, interfaces incluses, des liaisons point-à-point optiques ne devraient pas être sensiblement différentes des latences des liaisons électroniques (c'est-à-dire de l'ordre de quelques centaines de nanosecondes).

Néanmoins, il est clair que ces latences matérielles sont négligeables par rapport aux latences logicielles liées à l'exécution des procédures de communications, qui sont de plusieurs ordres de grandeur supérieures. La diminution de ces latences logicielles ne dépend pas directement du choix de l'optique pour la réalisation des liaisons. Elle peut être obtenue soit par une évolution de l'architecture logicielle comme les messages actifs, soit par l'adoption de protocoles de communication à très faible latence comme SCI.

Les perspectives des travaux portant sur l'impact de l'optique sur les liaisons point à point se situent d'une part sur le plan des débouchés industriels des avancées scientifiques, d'autre part sur le plan des recherches scientifiques elles-mêmes. En ce qui concerne le premier point, les perspectives à court terme sont une généralisation de l'optique dans les liaisons point à point utilisées dans les systèmes de haut de gamme, ces systèmes étant en particulier des réseaux locaux, des réseaux de stations de travail ou des ordinateurs parallèles basés sur l'interconnexion d'hypernoeuds composés de grappes de stations ou de multiprocesseurs à mémoire partagée. En ce qui concerne les recherches scientifiques sur les liaisons point à point, les perspectives portent principalement sur l'étude des interfaces opto-électroniques et électro-optiques, et de leur intégration dans des technologies de réalisation de circuits intégrés avancées. Par ailleurs, les études portant sur l'impact de l'optique sur les réseaux multiétages sont aussi porteuses de nombreuses perspectives de recherche, qui sont développées dans la section 8.3.

6 Routeurs intelligents

La contribution du MASI (Équipe Architecture) dans le projet consiste en l'étude de l'utilisation du routeur RCube (Routeur Rapide Reconfigurable) pour la construction de réseaux performants pouvant coupler soit des stations de travail ou bien de simples cartes PC. Le but est donc d'évaluer concrètement les capacités du routeur RCube en se basant sur des topologies de réseaux cibles.

Nous rappelons ci-dessous les caractéristiques de RCube:

1. Il comporte huit ports de communication à base d'une technologie de liaison série CMOS bidirectionnelle à 1 Giga-bits/s (par direction) développé par l'équipe de Mr. Roland Marbot à BULL.

2. Il Implémente un contrôle de flux en pipe-line (*worm-hole*). Un message peut être de n'importe quelle taille et possède un format qui peut encapsuler des messages typés (par exemple des cellules ATM).

3. Il supporte le schéma de routage par intervalle (un intervalle d'adresses est associé à chaque lien de sortie).

4. Il supporte le schéma de routage à adressage préfixé. Une adresse peut contenir des sous-adresses correspondant aux ports de transition dans un sous-réseau dans une organisation hiérarchique.

5. Il a une faible latence : 150 ns (pire cas avec les liaisons séries) entre l'entrée du début d'un message et sa sortie du routeur (sans contention).

6. Il supporte des schémas de routage adaptatif: un intervalle d'adresses peut être associé à plusieurs liens de sortie.

7. Il est reconfigurable : implémente des tables de routage compactes initialisables.

8. Il intègre un traitement d'erreurs sur la parité, le format de message, le contrôle de flux ainsi que les erreurs dues aux liaisons séries.

9. Il est réalisé en technologie CMOS 0.5 micron.

Ce routeur a été envoyé au fondeur (au moment de la rédaction de ce rapport).

6.1 Outil d'évaluation

Afin de permettre une utilisation efficace du routeur RCube nous avons conçu et développé un environnement de simulation, appelé MILE (*Methodology for Interconnection networks Labeling and Evaluation*). Ce développement entre dans le cadre du projet Européen OMI/MACRAMÉ en association avec d'autres partenaires (dont SGS-THOMSON et SINTEF-Oslo).

L'environnement MILE permet de construire graphiquement (voir figure 2) ou textuellement moyennant un langage procédural au dessus du langage C, des réseaux d'éléments communiquants, pouvant être de grande taille (de l'ordre de plusieurs centaines d'éléments). Ces réseaux sont bâtis en utilisant des modèles de composants (processeur, routeur, etc ..). Le comportement d'un composant est décrit en langage C. MILE permet de de simuler le fonctionnement global d'une topologie donnée et d'en extraire des mesures caractéristiques. Dans le cadre de notre étude, nous avons développé deux modèles C de composants, un modèle précis du routeur RCube et un modèle de processeur simple incluant un générateur de messages.

6.2 Réseaux simulés

Nous avons mené notre évaluation de performances sur deux types de réseaux que l'on peut construire à base de routeurs RCube.

- Réseaux en grilles 2-D : ce type de réseau est très populaire du fait de son faible coût. Un exemple de machine utilisant ce type de réseau est la Paragon ou encore la machine DASH. Le choix de la grille 2-D nous permet d'étudier qualitativement le comportement de RCube sur cette topologie. Nous avons considéré une taille unique de réseau (8×8) et implémenté l'algorithme déterministe classique (D) et trois algorithmes adaptatifs proches de ceux étudiés par Glass et Ni : West-First (A1), Last-North (A2) et Negative-first (A3). Sachant que l'utilisation de l'étiquetage par intervalles entraîne une perte d'information due au codage lui-même,

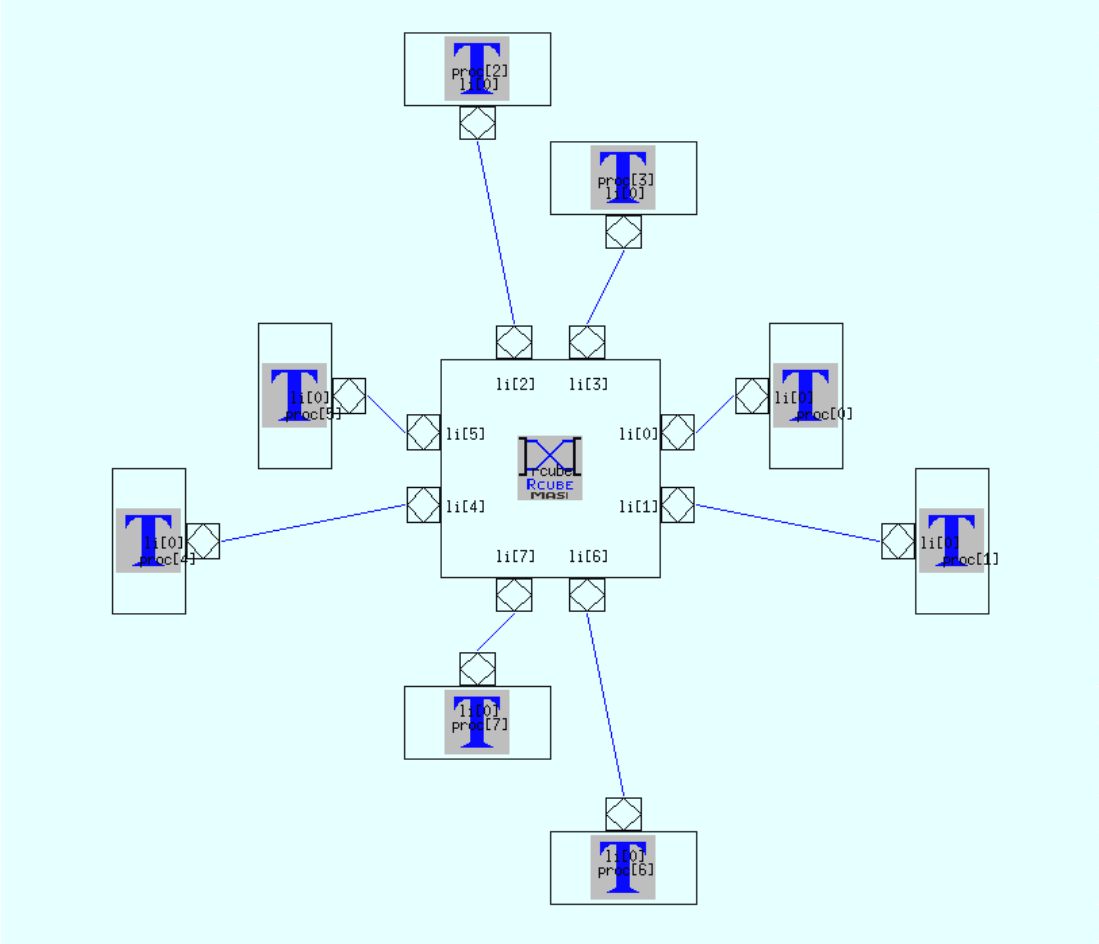


FIG. 2 – *Exemple de description graphique d'un réseau sous MILE. Cet exemple comprend un routeur RCube et huit instances d'un modèle de processeur (ou Transputer).*

l'algorithme Negative-first oblige à utiliser une grille telle que les liens horizontaux sont dédoublés, afin d'assurer un routage n'entrainant pas d'interblocage. La même topologie est utilisée pour comparer les différents algorithmes.

– Réseaux multi-étages de type *fat-tree* partiel que nous pouvons également appeler réseaux en papillon pliés (*folded butterfly networks*): le type de réseau que nous avons évalué dans cette catégorie est très proche de celui du réseau de la machine SP2 d'IBM. Une vue simplifiée mais non optimale est donnée par la figure 3. Cette évaluation nous permet d'établir une comparaison directe entre RCube et le circuit de routage (Vulcan) utilisé dans le réseau d'une machine SP2. Nous avons retenu trois tailles de réseaux (32, 64 et 128 processeurs).

6.3 Modèle de charge et paramètres

La méthode de simulation que nous avons adoptée est exactement celle utilisée par Stunkel et al pour le réseau de la machines SP2 d'IBM. Le modèle de génération de

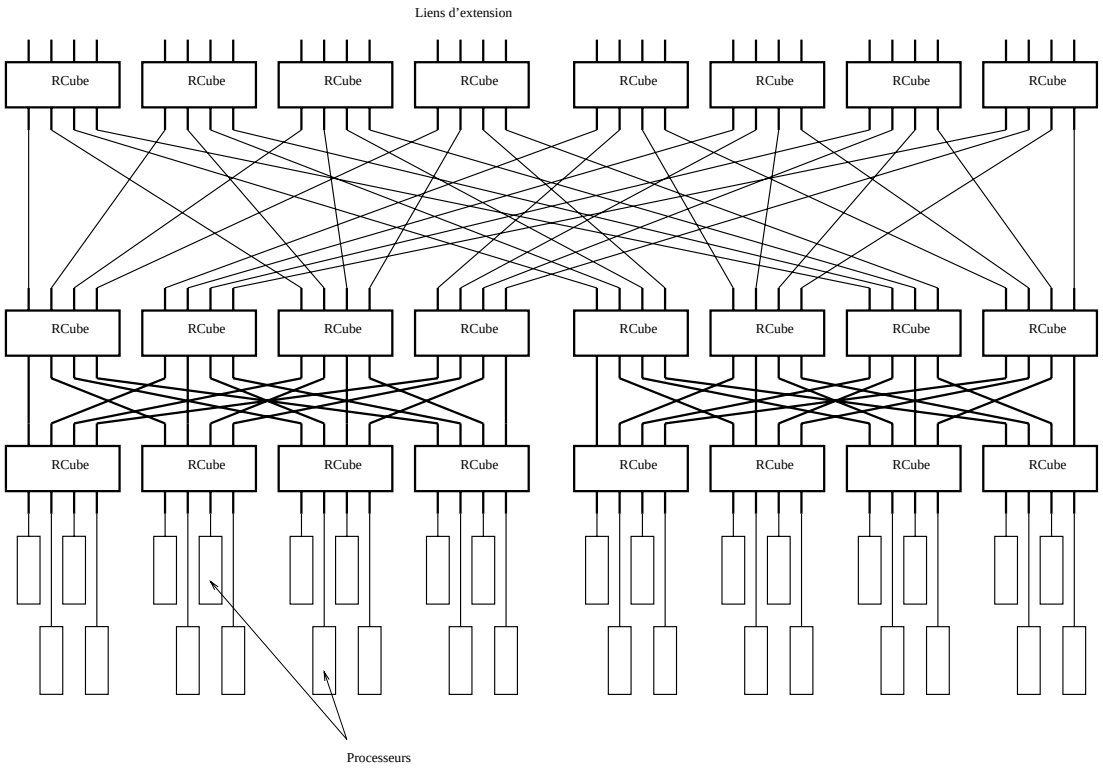


FIG. 3 – *Réseau multi-étage de la famille fat-tree à base de routeurs RCube.*

messages est uniforme. Chaque noeud d'un réseau tire au hasard et avec la même probabilité un noeud de destination parmi les autres noeuds du réseau. Les messages sont générés à des intervalles de temps déterminés par une loi de distribution exponentielle. Les messages générés sont mis immédiatement dans une file d'émission de longueur infinie (en attente de leur envoi sur le réseau). La consommation d'un message reçu est supposée sans attente.

Le générateur de messages possède deux paramètres de base:

- La taille des messages: dans notre cas nous avons simulé des envois de messages de 64, 128 et 500 octets (les messages de taille 500 octets ne sont pas découpés en messages de taille plus petite comme l'ont fait Stunkel et al pour la SP2).

- La charge: Ce paramètre spécifie le nombre de caractères effectifs injectés dans le réseau. Par exemple une charge de 0.5 implique la génération de 500 octets tout les 1000 cycles en moyenne à mettre dans la file d'émission. Cette entité est normalisée en tant que fraction du débit maximal (du lien) processeur (un caractère par cycle).

La mesure caractéristique d'un réseau est la relation entre la latence moyenne d'un message et la charge. La latence moyenne est le temps qui sépare la génération d'un message et sa réception. Ce temps inclut l'attente dans une file d'émission et la traversée du réseau jusqu'à son arrivée complète à sa destination.

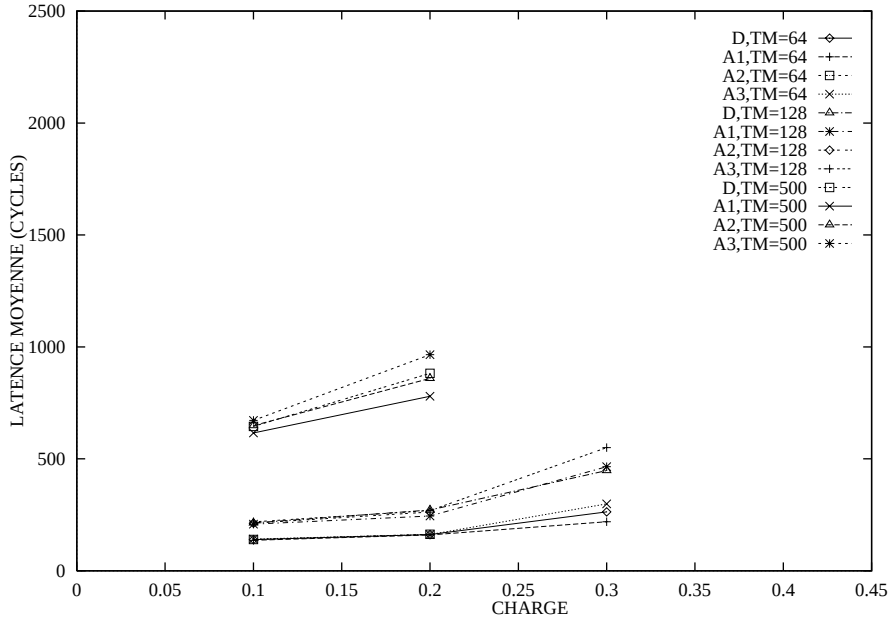


FIG. 4 – *Latence en fonction de la charge pour grille 2-D.*

6.4 Résultats

Les résultats obtenus pour la grille 2-D sont donnés par la figure 4. La latence y est exprimée en nombre de cycles.

- Message de taille 64(tm=64): Pour des charges jusqu'à 0.2 les quatre algorithmes sont comparables. Au delà de cette charge, le réseau atteint le point de saturation avec l'algorithme A2. Les algorithmes A1, D et A3 ne saturent le réseau qu'au delà de 0.3. Dans cette plage de charge l'algorithme déterministe occupe une place médiane entre A1 et A3.

- Message de taille 128: Le comportement des quatre algorithmes ne change pratiquement pas dans ce cas et est très semblable au cas précédent.

- Message de taille 500: Dans ce cas, les quatre algorithmes atteignent la saturation pour une charge supérieure à 0.2. De 0.1 à 0.2, les résultats montrent bien l'efficacité de l'algorithme A1 par rapport aux autres (dans l'ordre A2, D et A3).

Notons que les résultats ainsi décrits ci-dessus contredisent ceux qui ont été avancés par Glass et Ni. Ils montrent aussi la capacité du routeur RCube à tirer profit des algorithmes adaptatifs malgré la perte d'information due au codage.

Quant aux résultats obtenus pour des réseaux semblables à ceux implémentés dans les différentes configurations de machines SP2 (voir figure 5), ils nous permettent d'affirmer qualitativement qu'avec RCube nous pouvons obtenir des résultats comparables pour des charges inférieures à 0.5. Il est à noter que le routeur d'IBM, contrairement à RCube, intègre une file centrale (*central queue*) dont le rôle est de permettre de meilleures performances pour des grandes charges. Remarquons aussi, que cette com-

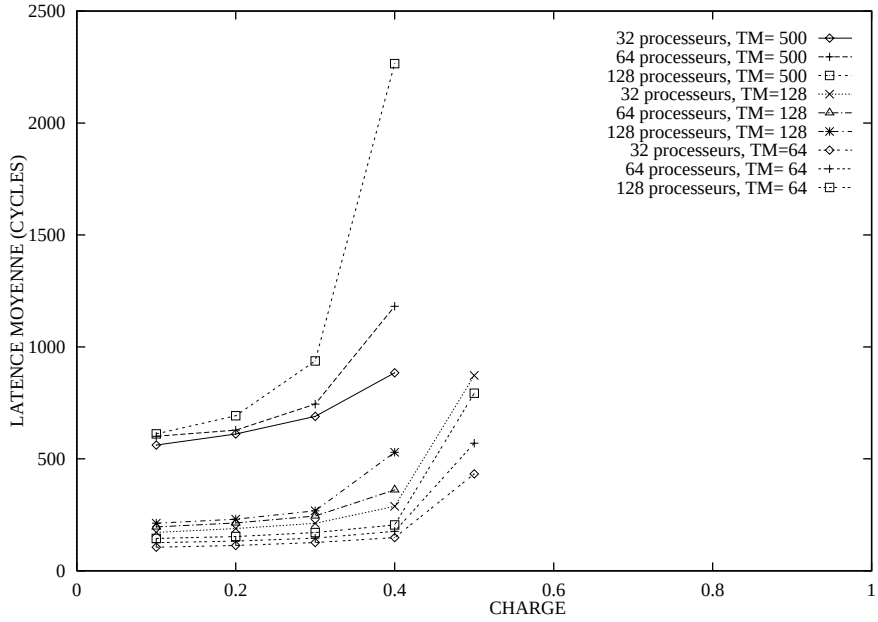


FIG. 5 – *Latence en fonction de la charge pour des réseaux en papillon.*

paraison est faite sur la base du cycle de fonctionnement. Cette comparaison aurait un tout autre sens si nous rappelons qu'un cycle RCube vaut la moitié d'un cycle du routeur Vulcan.

7 Utilisation de l'ATM

La contribution du LGI porte sur l'étude de l'utilisation de l'ATM dans les liaisons entre groupes locaux.

L'ATM a été conçu pour répondre à une large gamme de besoins de transfert d'informations à haut débit, avec des propriétés telles que l'allocation de bande passante, le support de réseaux virtuels, la gestion de données isochrone et asynchrone, la diffusion de groupes, et la garantie de qualité de services. Il s'est beaucoup développé dans le domaine du transfert de données, et en particulier dans les réseaux locaux. Avec l'utilisation des technologies de transmission optique, on voit alors se réduire l'écart qui existe, en termes de débits, entre les différents niveaux de communication des systèmes informatiques, depuis les réseaux à grande distance, jusqu'aux bus des stations de travail - en passant par les réseaux d'interconnexion des machines parallèles.

Malgré ses limites, dues au fait qu'il est un compromis entre les besoins de divers types de communications, l'ATM pourrait constituer un protocole de communication unifié et universel, des protocoles spécifiques n'étant plus nécessaires que lorsque des besoins critiques se font sentir. Ceci pourrait présenter un certain nombre d'avantages : interconnexion simple avec les réseaux locaux ou de télécommunications, utilisation des propriétés de l'ATM dans le domaine du multi-média ou du temps réel, et diminution des coûts du fait de la standardisation.

7.1 ATM et calcul parallèle : état de l'art

Les caractéristiques de l'ATM ont fait surgir un certain nombre de questions concernant son efficacité pour le calcul parallèle :

- L'ATM fonctionne en mode connecté, alors que les réseaux couramment utilisés dans le domaine du calcul parallèle fonctionnent en mode non connecté. Cependant, certains résultats de mesures, publiés en particulier dans le cadre du projet SHRIMP de l'université de Princeton, montrent que les applications parallèles possèdent le plus souvent les propriétés de localité nécessaires à l'utilisation efficace d'un système de communication en mode connecté. Une autre difficulté résidait dans la nécessité de réserver la bande passante utilisée par chaque connexion : la prévision de cette bande passante est en effet difficile dans le cas des applications de calcul parallèle. L'évolution du standard ATM a apporté la solution de ce problème fin 94, avec la définition de classes de services dites élastiques, permettant d'utiliser la bande passante disponible sans réservation préalable. Avec ces classes de service, le risque de congestion du réseau est minimisé au moyen d'un contrôle de flux par régulation du débit d'émission à la source.

- L'ATM transmet les informations par petites unités, appelées cellules, transportant 48 octets de données utiles. Cette taille peut paraître excessive pour des applications parallèles où les messages très courts peuvent être fréquents. Cependant, les statistiques concernant la distribution de la longueur des messages qui ont été publiées montrent que cet inconvénient est acceptable pour la majorité des applications parallèles. En outre, des niveaux de communication comme PVM ou MPI permettent de transférer en un seul message des ensembles de données qui ne sont pas rangés en mémoire de façon contiguë : blocs de matrices, ou structures de données arbitraires.

Les travaux effectués jusqu'à présent dans le domaine du calcul parallèle sur réseau ATM démontrent qu'on ne peut pas tirer pleinement parti des performances du réseau avec les matériels et logiciels standard : ainsi, sous TCP/IP, le gain de performances obtenu par rapport à Ethernet est le plus souvent très inférieur au rapport des débits. L'avantage de l'ATM se limite le plus souvent à l'absence de contention sur un réseau maillé. Les temps de traitement logiciels sont en effet tels que les latences d'application à application ne sont pas améliorées - quand elles ne sont pas dégradées.

Certains travaux ont cependant implémenté un niveau de communication intégré finement dans le système d'exploitation, en s'attachant à minimiser (ou supprimer) les interventions du système d'exploitation et les recopies de données (messages actifs, émulation d'accès mémoire à distance). Les latences et débits de communication obtenus sont alors proches de ceux qui sont obtenus sur les machines parallèles.

7.2 Primitives de communication

Les résultats publiés jusqu'à présent montrent l'intérêt de disposer de primitives de communications qui permettent de minimiser le délai de traitement logiciel des communications et puissent s'interfacer efficacement avec l'ATM (ou les couches d'adaptation à l'ATM) et en exploiter au mieux les propriétés,

L'étude des primitives de communication nous a amené à nous intéresser aux deux modes de communication utilisés dans le domaine du calcul parallèle : la mémoire par-

tagée, et la communication par messages.

La mémoire partagée peut être émulée sur les machines sans mémoire commune, soit par le biais de la cohérence de cache, soit par le biais de la mémoire virtuelle distribuée.

La cohérence de cache pourrait être envisagée pour un second niveau de communication, avec des applications présentant une bonne localité des références. Ceci suppose toutefois que les latences de communication soient fortement diminuées par rapport à ce qui se fait actuellement dans les réseaux ATM. De plus, l'utilisation d'un réseau ATM pour implémenter la cohérence de caches amène à reprendre à la base l'étude du système (gestion de répertoires, modèles de cohérence, etc...). Dans l'état actuel de ce domaine de recherche, une telle étude dépasse de loin le cadre d'un projet de deux ans, et nous avons donc abandonné cette voie.

La mémoire virtuelle distribuée, par contre, est un mécanisme souvent proposé au dessus de réseaux locaux, et son implémentation au dessus d'un réseau ATM semble plus naturelle. Notre étude dans ce domaine s'est bornée à l'implémentation d'un simulateur permettant d'évaluer le comportement de la mémoire virtuelle distribuée au dessus d'un réseau quelconque, en permettant de faire varier les paramètres de gestion de la mémoire virtuelle et les caractéristiques du réseau.

En ce qui concerne la communication par messages, nous nous sommes intéressés à deux systèmes : PVM et MPI. MPI est un interface de communication standardisé dont les spécifications définitives ont été publiées début 94. Il présente un certain nombre de caractéristiques intéressantes, telles que les types de données dérivés (permettant de spécifier des ensembles de données arbitraires placés en mémoire de façon non contiguë), la notion de groupes de processus et de contextes de communication, les topologies virtuelles. Cependant, MPI est assez complexe, il est encore sujet à évolutions, et il n'en existait que deux implémentations dans le domaine public, toutes deux basées sur des niveaux de communication sous jacents.

Nous avons donc préféré nous intéresser à PVM, dont les implémentations avaient plus de maturité, et pour lequel il existe de nombreuses applications. De plus, une implémentation de PVM au dessus de l'ATM est disponible.

En l'absence d'un réseau ATM opérationnel dans notre laboratoire à ce stade de l'étude, nous avons choisi de faire tourner PVM au dessus d'un réseau ATM simulé. Ceci permet, outre l'étude des opérations de communication de base effectuées par PVM, d'évaluer le comportement du réseau ATM avec la charge induite par le calcul parallèle.

L'implémentation du simulateur est en voie d'achèvement, mais des problèmes de performances ne permettent pas, actuellement, l'exécution de programmes PVM de taille suffisamment significative. Ces problèmes semblent liés en particulier à un trop bas niveau de simulation de certaines fonctionnalités, ce qui pénalise inutilement les performances. On peut, par ailleurs, envisager le portage du simulateur sur une machine plus puissante (multiprocesseurs, ou machine parallèle type SP1), pour permettre l'exploitation du simulateur par une campagne de meures systématiques.

Les primitives de communication sous-jacentes à des systèmes tels que PVM ou MPI sont l'envoi et la réception (bloquants ou non-bloquants) de messages, éventuellement dispersés en mémoire. Un interface de communication "intelligent", permettant de minimiser les délais de traitement logiciels devrait donc avoir les caractéristiques

suivantes :

- permettre le transfert de données directement entre l'espace utilisateur et l'interface de transmission, sans assemblage intermédiaire des données.
- minimiser, voire supprimer, les interventions du système d'exploitation. Les émissions ou réceptions devraient pouvoir être déclenchées en mode utilisateur, tout en assurant les protections nécessaires.

Le premier point demande un système de DMA dispersé (scatter-gather DMA), opérant en pipe-line avec la segmentation/réassemblage des messages. Le second point n'est possible que si les commandes de l'interface de communication sont "mappées" dans l'espace d'adressage de l'utilisateur par le biais des tables de pages, de telle sorte que seules les commandes autorisées soient accessibles à l'utilisateur. Ceci est d'ailleurs la solution adoptée dans SHRIMP, dans lequel il est, de plus, établi une correspondance entre les zones mémoire d'émission et de réception, les modifications apportées à une zone pouvant être répercutées, explicitement ou automatiquement, dans l'autre zone.

Il n'a pas été possible de pousser plus avant la définition de l'interface de communication dans le cadre du projet ILIAD. Les travaux sur ce point sont toujours en cours.

7.3 Réseaux ATM et calcul parallèle

Les réseaux ATM développés jusqu'ici sont destinés aux réseaux locaux ou à grande distance. Ils sont basés sur des modèles de trafic valides dans de tels environnements, mais qui ne le sont peut être plus dans le cas de réseaux supportant le calcul parallèle où les sources de trafic ne sont pas indépendantes. Il reste donc à vérifier que les algorithmes de contrôle de flux et le dimentionnement des mémoires tampons couramment adoptés dans les réseaux ATM soient compatibles avec la charge générée par des processus coopérant à une tâche de calcul parallèle. Ceci est une des retombées que l'on attend du simulateur de réseau mentionné précédemment.

La conception de réseaux ATM optimisés pour le calcul parallèle, en termes de coûts et de performances, et permettant d'aborder le domaine du parallélisme massif, pourrait par ailleurs, être l'objet de futurs travaux de recherches dans ce domaine.

Un réseau ATM dédié au calcul parallèle pourrait être basé sur des technologies de transmission et de routage déjà proposées pour les machines parallèles, comme le Rcube conçu par le MASI. Le Rcube pourrait en effet transmettre des cellules ATM, à la condition de "brider" sa capacité de routage adaptatif, de façon à garantir la délivrance des messages dans l'ordre de leur émission. Ceci permettrait d'implémenter les classes de services nécessaires au transfert de données, mais il reste difficile de garantir la qualité de service requise par les applications multi-média (délai borné, bande passante réservée).

La nécessité de fournir une telle qualité de service au niveau du réseau d'interconnexion d'une machine parallèle reste matière à discussions. Il est cependant probable que des contraintes temps-réel devront pouvoir être respectées dans certains types d'applications. Les travaux effectués dans ce domaine ont généralement abouti à proposer des routeurs comportant au moins deux files d'attente par lien, avec un système de priorité permettant l'acheminement prioritaire des messages comportant des contraintes de

délais. Il est probable qu'un principe analogue pourrait être adapté à la transmission de flux ATM multi-média, mais cela devrait faire l'objet de travaux de recherches ultérieurs.

8 Modélisation des Communications

8.1 Outils classiques pour la modélisation

8.1.1 Importance des communications dans les machines parallèles

L'existence des machines parallèles actuelles est motivée par la nécessité de résoudre des problèmes de très grande taille en calcul intensif. L'espace mémoire requis n'est pas disponible sur les calculateurs vectoriels conventionnels et implique l'usage d'une mémoire distribuée. Les données ainsi réparties sur cette mémoire distribuée sont le plus souvent accessibles par passage de messages à travers le réseau d'interconnexion reliant les différents modules.

Les communications constituent alors un facteur prépondérant pour la performance de ces machines parallèles, et plus spécifiquement, celles du type MIMD. Une telle machine est vue comme un ensemble de processeurs coopérant via un réseau d'interconnexion. On distingue essentiellement deux types de réseaux, statiques (ou réseaux point-à-point) et dynamiques (réseaux multi-étages).

Dans le premier cas, les processeurs sont connectés les uns aux autres directement ou par l'intermédiaire de composants de routage, dans le second cas, on assemble un certain nombre de commutateurs élémentaires pour relier un processeur à un autre. Dans les deux cas le réseau constitue un goulot d'étranglement qui régule fortement la puissance potentielle de calcul du système.

8.1.2 Étude des propriétés intrinsèques des réseaux

L'efficacité intrinsèque du réseau conduira en général à des applications plus ou moins performantes. Il est à noter que les réseaux multi-étages ont été beaucoup étudiés il y a une vingtaine d'années pour relier des unités de traitements à une grande mémoire divisée en bancs (premiers ordinateurs parallèles à mémoire partagée ou vectoriels). C'est aussi une technologie largement utilisée pour les réseaux de télécommunications. Ils composent les réseaux de plusieurs machines parallèles actuelles (par exemple: IBM SP1 et 2, Meiko CS2, NEC Cenju 3, et sous une certaine forme, la CM5 de TMC).

Les réseaux point-à-point s'apparentent à des réseaux qui ont été étudiés dans un contexte différent dans le domaine de la théorie des graphes. De cette expérience, on peut extraire des paramètres "qualitatifs" qui permettent de donner une première idée des performances que l'on peut attendre d'un réseau modélisé par un graphe. Les principaux paramètres caractérisant ces graphes sont le diamètre du graphe (qui est la plus longue distance entre une paire quelconque de nœuds –processeurs–), le degré (égal au nombre maximal de voisins directs), la bissection (nombre minimum d'arcs -liens- à enlever pour déconnecter le graphe en deux parties de tailles égales), ou encore la congestion (qui mesure l'encombrement maximum des liens de communication). Ces paramètres sont intrinsèques, au sens où ni l'algorithme, ni la machine (et tous les dispositifs de communications associés) ne sont vraiment pris en compte. Parmi

les machines parallèles à réseau d'interconnexion point-à-point, citons par exemple la Paragon d'Intel ou le CRAY T3D.

Du point de vue de l'utilisateur qui désire faire tourner une application particulière, les paramètres précédents sont insuffisants. Pour prédire les performances d'un algorithme sur une machine donnée, une voie raisonnable est de modéliser le comportement du réseau sur un ensemble de procédures structurées collectives, représentatives de classes d'algorithmes ou bien sur des communications aléatoires (irrégulières). Un des apports important du Groupe RUMEUR dans ce projet a consisté à faire un état de l'art sur les techniques fondamentales d'algorithmique parallèle et leurs liens avec les problèmes d'optimisation des communications. Du point de vue du concepteur de programmes parallèles, cela permet de prédire assez précisément les performances pour choisir un algorithme ou un autre adapté à la topologie du réseau. La contribution de RUMEUR sur ce point a porté sur l'étude approfondie des principaux réseaux existants (Hypercube, Grilles toriques, de Bruijn, ou encore, réseaux multi-étages), leur modélisation et la donnée de protocoles efficaces de communications globales.

A chaque type de génération de machines parallèles (et de réseaux d'interconnexion) correspond un modèle particulier de communications (type de commutation, temps de transfert entre processeurs, nombre de ports, etc..). Sur les principaux modèles de communications et les différents modes de routage disponibles aujourd'hui, nous avons obtenus des bornes inférieures du temps des procédures ainsi que des algorithmes de communications collectives sinon optimaux du moins, qui s'approchent de ces bornes dans la plupart des cas. Ceci s'appuie sur l'étude des technologies de routage et leurs modélisations.

Notons que ce type de résultats ne peut être obtenu que pour certains problèmes spécifiques. Pour des applications quelconques, il est nécessaire de développer une approche plus quantitative et expérimentale pour modéliser les communications de base. Ainsi, de nombreuses études ont été menées sur les machines existantes dans les différents centres constituant le groupe RUMEUR.

8.2 Modèles pour les nouvelles architectures

Nous présentons ici deux modèles de communication prometteurs pour les futures machines. Le *wormhole* est étudié dans le cadre des commutations de type "circuit" car les expressions des communications associés aux modes *circuit-switched*, *cut-through*, ou *wormhole* sont identiques. Cette étude est reliée aux recherches du groupe du MASI sur le routage, et des travaux ont été initiés suite aux réunions ILIAD pour décrire des protocoles de communications comme la diffusion ou l'échange total qui puissent être implantés avec une fonction de routage donnée (i.e. le routage XY sur la grille, *left-to-right scan* sur l'hypercube ...), ici le routage par intervalles. Le second modèle est motivé par les nouvelles données technologiques provenant de l'optique appliquée à l'informatique, fournies par les deux équipes de l'IEF et l'IOTA. Ces études sont poursuivies intensément au sein du groupe ROI (Rencontres Optique Informatique) nouvellement crée (cf. document ROI en annexe).

8.2.1 Commutation de circuit - wormhole

L'étude des communications globales en mode *wormhole* ou commutation de circuit est récente. Nous allons illustrer les particularités des communications pour ce modèle

en présentant la fonction de coût associée à la transmission des messages, puis en donnant les critères d'optimalité recherchés. Enfin, à titre d'exemple, nous décrirons brièvement les résultats obtenus pour la diffusion et l'échange total dans le tore. Ces résultats sont les meilleurs connus à l'heure actuelle.

Dans le modèle commutation de circuit, transmettre un message à distance d n'est pas équivalent à répéter d fois un processus de communication aux voisins. La modélisation communément admise est : $\alpha + d\delta + L\tau$

où L est la longueur du message à envoyer, $\frac{1}{\tau}$ la bande passante des liens, et α un temps de start-up dû au processus qui envoie le message. Le délai δ est un temps lié aux commutations des routeurs sur chaque nœud intermédiaire.

Dans un graphe G de degré maximum Δ, pour une commutation de type circuit, en supposant que les routeurs peuvent communiquer sur tous leurs ports en même temps, le nombre maximum de nœuds qui peuvent être informés en k étapes est $(\Delta + 1)^k$. En effet, à chaque étape, on peut informer au plus Δ fois plus de nœuds. Le nombre total de nœuds qui peuvent être informés à la fin de chaque étape suit une suite géométrique de raison Δ.

La restriction de l'analyse au mode commutation de circuit s'impose ici car, en mode *wormhole*, un nœud peut émettre un nouveau message avant que le précédent ne soit arrivé à destination.

D'après la borne inférieure précédente sur le nombre d'étapes, on peut donc espérer obtenir, un algorithme de diffusion procédant en $\log_{\Delta+1} N$ étapes (pour une machines à N processeurs), à opposer aux D étapes (diamètre, par exemple, de l'ordre de $\sqrt{N}$ dans le cas d'une grille) nécessaires en mode *store-and-forward* (qui constituait l'essentiel des communications dans la génération de machines précédente). Le gain peut donc être significatif dans le cas de réseaux de grand diamètre comme l'anneau ou la grille, et dans le cas où c'est le temps d'initialisation des communications qui domine celui de la transmission.

Si l'on considère les autres facteurs de la fonction de coût, on en déduit que le temps total de la diffusion dans ce modèle comporte trois termes : la somme des temps de *start-up* (facteur en α), le temps total de la transmission du message (facteur en $L\tau$) et le temps total de routage dans les nœuds intermédiaires (facteur en δ).

Dans un graphe sommet transitif (ce qui est le cas de la plupart des réseaux utilisés en pratique) G d'ordre N, de degré Δ, d'arête connexité λ, et de diamètre D, le temps minimun pour diffuser un message de longueur L est :

$$\max\left(\lceil \log_{\Delta+1} N \rceil \alpha, D\delta, \frac{L\tau}{\lambda}, \alpha + D\delta + \frac{L\tau}{\Delta}\right)$$

Les réseaux bidimensionnels et de faible degré, comme le tore, sont les meilleurs candidats pour des réalisations en VLSI. Dans un mode de routage de type *store-and-forward*, ces topologies sont pénalisées par la durée des communications qui dépend du diamètre ($O(\sqrt{N})$), alors que cette durée n'est que logarithmique pour les réseaux "hypercubiques". Le routage de type *wormhole* ou *circuit-switched* apporte un nouvel intérêt aux réseaux de faible degré en fournissant des schémas de communication aussi performants sur des tores que sur les hypercubes. En particulier, Peters et Syska ont montré que diffuser dans un tore $5^k \times 5^k$, $k \geq 1$, nécessite un temps d'au plus $2k\alpha + (5^k - 1)\delta + 2kL\tau$. Cet algorithme minimise les termes en α et δ pour les tores de

dimensions égales à la même puissance paire de 5. Les pavages mis en œuvre dans le protocole peuvent être adaptés à d'autres tailles, et donner ainsi des protocoles de diffusion optimaux en nombre d'étapes.

8.2.2 Réseaux par Bus

Un réseau d'interconnexion par bus est constitué d'un ensemble d'éléments de calculs (les processeurs) et d'un ensemble d'éléments de communication (les bus). Les processeurs produisent ou consomment des messages et les bus fournissent des voies de communications qui permettent d'échanger des messages entre les processeurs. Chaque bus fourni un lien de communication entre deux ou plusieurs processeurs. Ces réseaux sont une généralisation des réseaux point-à-point et pourraient être utilisés pour les futures machines parallèles. Leur intérêt est encore plus grand aujourd'hui car ils permettent de modéliser les réseaux d'interconnexion optiques.

Plusieurs modèles basés sur les réseaux par bus ont été présentés par les chercheurs qui travaillent sur les réseaux optiques. En particulier Jiang et Stern ont étudié la classe des réseaux du type "multicast multihop lightwawe network".

Dans cette étude, les stations d'accès reliées à la couche physique (fibre optique) peuvent être partitionnées en sous-réseaux de telle façon que tout signal émis par une station du sous-réseau peut être diffusé à l'ensemble des stations qui le constitue (multicast).

La couche logique est formée par un ensemble de sous-réseaux de ce type et peut être modélisée par un bus (où les stations correspondent aux processeurs tandis que les bus représentent les sous-réseaux). Le modèle abstrait pour ces sous-réseaux est celui des hypergraphes où les sommets représentent les processeurs alors que les hyper-arêtes représentent les bus.

Un premier problème consiste en l'étude de construction de graphes qui permettent d'interconnecter un grand nombre de processeurs pour un degré maximum Δ, une taille de bus r et un diamètre D. Ce problème est celui du (Δ, D, r) dans l'étude des hypergraphes.

Certaines constructions sont obtenues à partir de classiques réseaux point-à-point en groupant simplement plusieurs processeurs sur un bus. On a par exemple généralisé les grilles et les tores, ou encore les hypercubes. D'autres constructions sont obtenues grâce à des outils spécifiques de la théorie des hypergraphes. Par exemple on peut utiliser de résultats sur les design ou la dualité. Le *dual* d'un hypergraphe $H = (\mathcal{V}(H), \mathcal{E}(H))$ est l'hypergraphe $H^* = (\mathcal{V}(H^*), \mathcal{E}(H^*))$ où les sommets de H^* correspondent aux arêtes de H, et les arêtes de H^* correspondent aux sommets de H. Un sommet e_j^* appartient à une arête V_i^* de H^* si et seulement si le sommet v_i appartient à E_j de H. En particulier, les duaux de graphes décrivent des réseaux par bus dans lesquels chaque processeur est connecté à seulement deux bus. On peut aussi utiliser des techniques de compositions de graphes (*compounds*) et on peut enfin obtenir de bons résultats en mixant toutes ces constructions ensemble.

Enfin nous avons donné quelques constructions directes comme celles d'hypergraphes de de Bruijn ou Kautz. En fait, il est préférable de les traiter comme des hypergraphes orientés. Le modèle des hypergraphes orientés paraît bien adapté aux applications dans lesquelles on peut distinguer de émetteurs et des récepteurs. De plus, ils peuvent servir

à étendre le modèle de graphe de tâches à celui d'hypergraphe de tâches.

Les problèmes de communication étudiés dans ce modèle considèrent différentes hypothèses. Par exemple les bus peuvent être du type lecture ou (exclusif) écriture, ce qui signifie qu'un seul processeur à la fois peut écrire sur le bus alors que tous peuvent lire le message écrit. Comme dans les réseaux point-à-point on peut supposer que soit un processeur peut écrire sur un seul bus à la fois ou bien sur tous ceux auxquels il est connecté (1-port ou Δ-port). Enfin pour les problèmes d'échange total on peut autoriser ou non la concaténation des messages.

8.3 Technologies émergentes

La confrontation de diverses communautés au sein d'ILIAD a permis au groupe RUMEUR de définir de nouveaux axes de recherche très prometteurs comme :

- la modélisation des communications de type ATM
- les réseaux tout-optiques
- l'exploitation efficace des fermes de station de travail

Ces trois points feront l'objet de cours spécifiques lors de la prochaine école d'automne RUMEUR à Arcachon en septembre 96.

De plus, les modèles mis au point sur ces nouvelles technologies devraient permettre de mieux appréhender le comportement des communications sur les machines de demain et de construire des algorithmes parallèles plus efficaces.

8.3.1 Optique ou Électronique ?

L'utilisation de plus en plus fréquente de l'optique dans les réseaux de télécommunications a permis de développer des composants qui pourraient aujourd'hui apporter des solutions dans le domaine des machines parallèles (i.e. les commutateurs *tout optique*).

Ce n'est pas la vitesse de la lumière qui fait l'intérêt des supports de communication optique mais la large bande passante qu'ils offrent.

Même si aujourd'hui, notamment sur des courtes distance - au niveau d'une carte - les liaisons électroniques offrent encore de meilleurs débits, les performances atteignent un seuil dû à des problèmes d'impédance. Un autre intérêt de la transmission par voie optique est de pouvoir "facilement" transmettre les bits en parallèle, ou encore d'utiliser une surface plus réduite dans les circuits (MQW).

Le problème de la topologie du réseau d'interconnexion se pose différemment en optique : il n'y a plus de contrainte bidimensionnelle (on peut même traverser le silicium sans perturbation du circuit pour certaines longueurs d'onde), les liaisons peuvent être de longueurs plus importantes et les courbures autorisées sont quasi libres pour les fibres. De plus, en optique libre les faisceaux peuvent se croiser sans problème d'interférence.

En résumé, on peut espérer exploiter une plus large bande passante et avoir une plus grande flexibilité pour ce qui est des technologies.

Autre point qui intéresse les concepteurs d'algorithmes de communication pour les machines parallèles, le schéma *multicast*. Cette fonction, dont l'utilité a été démontrée largement, est simple à implanter en optique alors que ce n'est pas le cas en électronique.

Cependant on ne doit pas oublier certains problèmes inhérents à l'optique: la mémoire et les processeurs courants sont purement électroniques et les conversions du monde optique au monde électronique sont encore coûteuses en temps. Ceci est principalement dû au fait que le silicium ne permet pas l'émission de lumière, la solution est peut être le passage de la technologie CMOS à l'AsGa?

8.3.2 Composants optiques pour construire des réseaux

Nous avons pu - au cours des rencontres avec les opticiens - dégager un certain nombre de composants qui pourraient être les nouvelles briques de base des réseaux de machines parallèles. Afin de (re)diriger la lumière, on peut utiliser aujourd'hui:

- des hologrammes ou des photo-cristaux
- des composants acousto-optique
- des matrices de SEEDs
- des matrices de smart-pixels

Pour ce qui est des émetteurs et des récepteurs on utilise des LED et des diodes photo-sensibles.

Dans l'étude du stage de DEA de Bruno Pineda (DEA informatique UNSA-I3S 1995) nous avons proposé une architecture de commutateurs optiques de type multi-étages. L'élément de base considéré est un composant acousto-optique tels que ceux utilisés à l'institut d'optique d'Orsay. Grâce à ce type de composant il serait possible par exemple de réaliser des cross-bar 180x180 en seulement trois étages. De plus ces commutateurs peuvent implanter le multicast qui n'est pas disponible usuellement sur les *switches* électroniques.

8.3.3 Perspectives

La rencontre de groupes aussi diversifiés que ceux au sein du groupe ILIAD a permis à RUMEUR de recadrer certains de ses axes de recherches soit en préservant sa nature théorique d'origine en théorie des graphes et combinatoire (réseaux par bus et hypergraphes), soit en développant de nouveaux centres d'intérêts comme les liaisons optiques (ROI) ou les nouvelles solutions distribuées fortement couplées (Notamment dans le cadre des projets INRIA APACHE et SLOOP).

9 Conclusions

Le projet ILIAD a abordé un certain nombre de problématiques liées à l'architecture des machines parallèles. Or, si ce domaine de recherche était considéré comme prometteur à l'origine du projet, il a, entre-temps, fait l'objet d'une certaine désaffection, motivée en particulier par l'échec commercial d'un certain nombre de machines parallèles. Au terme du projet, il nous a paru intéressant de s'arrêter un peu sur ce point.

L'échec commercial des machines parallèles nous parait avoir plusieurs causes, parmi lesquelles se trouvent l'étroitesse du marché pour des machines d'un coût très élevé, les

progrès de la technologie qui font fondre assez rapidement leurs avantages en performances, le manque de maturité des environnements logiciels pour le calcul parallèle, la difficulté de la programmation parallèle, et la concurrence des environnements de calcul parallèle sur réseaux de stations de travail, tels que PVM.

L'entrée en lice de deux constructeurs puissants, IBM avec la SP2, et Cray avec la T3D, semble laisser peu de parts de marché aux petits constructeurs. L'idée communément admise est en effet que les besoins de très forte puissance de calcul représentent un marché assez réduit, qui ne laissera de place qu'à une poignée de constructeurs, alors que, pour des applications moins demandeuse de temps de calcul, les systèmes de programmation parallèle sur réseaux de stations de travail (type PVM ou MPI) suffiront à répondre aux besoins, le passage aux réseaux à haut débit (ATM, en particulier) permettant de réduire les temps de communication.

Cependant, on sait que les progrès de la technologie sont limités, même si l'on est incapable, à l'heure actuelle, de dire quelles sont ces limites. Par contre, les besoins en puissance de calcul continuent de croître. Certaines applications, en physique par exemple, ne sont limitées que par la capacité des machines disponibles. De nouveaux besoins apparaissent par ailleurs : le développement du multi-média, par exemple, devrait entraîner de nouveaux besoins en traitement d'images, ou en synthèse d'images animées de haute qualité. Pour répondre à ces besoins, le développement d'architectures parallèles nous semble donc rester incontournable à terme.

L'utilisation de systèmes de programmation parallèles sur réseaux de stations de travail fournit une solution pour des applications modérément parallèles, avec des besoins de communication assez faibles. Nous avons vu que le développement des réseaux à haut débit - tels que l'ATM - ne suffit pas à étendre le domaine d'utilisation de ces systèmes, tant que les temps de traitement logiciels restent aussi importants. Il ne sera donc possible de tirer parti des performances de ces réseaux qu'à travers des interfaces particulièrement optimisés qui devront probablement recourir à des solutions matérielles : ceci est un des axes de recherches qui seront poursuivis par le LGI dans ce domaine.

L'implémentation d'interfaces de communication logiciels/matériels plus rapide permettra d'augmenter le degré de parallélisme exploitable sur réseaux de stations de travail (ou de micro-ordinateurs). Cependant, l'utilisation de solutions similaires permettra d'obtenir également une diminution importante des latences de communication sur les réseaux d'interconnexion des machines parallèles, de telle sorte que celles-ci devraient conserver un avantage de performances important, leurs réseaux étant 10 à 100 fois plus rapides que les réseaux ATM actuels. L'utilisation de réseaux spécialisés pour interconnecter des stations de travail ou des micro-ordinateurs, de manière analogue aux machines SP1 et SP2 d'IBM, reste donc une solution intéressante. De tels réseaux pourraient être organisés autour de routeurs rapides tels que le Rcube : un projet sur ce thème a été initié par le MASI. Des améliorations de performances des réseaux ATM peuvent également être attendues dans l'avenir, mais la réduction des latences des réseaux ATM ne nous semble pas devoir être, en soi, un objectif de recherches prioritaire pour l'instant.

Enfin, des solutions architecturales entièrement spécifiques (type Paragon, T3D,...) devraient rester la solution incontournable pour le parallélisme à grain fin. On pourrait donc voir apparaître une assez large gamme de solutions pour le calcul parallèle, depuis les solutions entièrement logicielles sur réseaux haut débit, jusqu'aux machines

massivement parallèles de quelques milliers de noeuds.

L'impact des technologies optiques était un autre des grands thèmes abordés dans le projet, en particulier par l'IEF et l'IOTA. Nous avons vu que l'optique permet d'apporter un gain quantitatif pour les transmissions, en terme de bilan énergétique, de débit, de distances de transmission. Une des difficultés réside cependant dans le fait que les transducteurs électro-optiques (et particulièrement les diodes laser) font appel à des technologies différentes de celles qui sont utilisées couramment pour la logique intégrée (GaAs). Il est donc impossible d'intégrer sur un seul circuit la logique, l'électronique de commande des transducteurs optiques, et les transducteurs eux mêmes. La solution semble alors d'intégrer tout sauf les transducteurs.

L'utilisation des technologies de transmission optique devrait se développer pour des transmissions en point à point. Par contre, les transmissions optiques en multipoint restent un domaine beaucoup plus spéculatif - même si elles pourraient être d'un grand intérêt. L'optique libre peut fournir des solutions intéressantes pour la conception des machines parallèles, dans la mesure où elle permet de s'affranchir relativement de certaines contraintes géométriques des réseaux d'interconnexion.

Un problème qui semble important avec les transmissions à haut débit est celui des erreurs de transmission. Les machines parallèles actuelles considèrent le plus souvent que le réseau est fiable, et n'implémentent pas de protocole de récupération d'erreurs de transmission. Or, avec les technologies de transmission optiques sur diodes laser, le taux d'erreur est de 10^{-9} à 10^{-14}. Avec des débits dépassant de gigabits/s, de tels taux d'erreurs sont inacceptables, et il sera nécessaire de prévoir des protocoles de récupération d'erreurs.

Les relations entre processeurs et mémoires ont fait l'objet de travaux de la part de deux des équipes du projet, avec deux problèmes particuliers : le débit des mémoires (et des liaisons processeurs-mémoires), et la latence des accès. Le débit des mémoires et des réseaux d'interconnexion processeurs-mémoires (bus partagés, en général) constituent une limite au nombre des processeurs pouvant partager un même espace mémoire physique. Le remplacement des bus partagés par de multiples liens série à haut débit est une des solutions qui ont été proposées et étudiées dans le cadre du projet.

Par ailleurs, les latences des mémoires pénalisent les performances de processeurs de plus en plus rapides. Les mémoires caches à deux, voire trois niveaux permettent de palier en partie ce problème, et les caches "skewed associatifs" permettent d'améliorer le taux d'échec des caches, ainsi que la régularité de leur comportement en fonction des applications. D'autres techniques visent à permettre de tolérer les latences des accès mémoire en cas de défauts de cache : préchargement, découplage, désordre d'exécution, flots d'instructions multiples.

Les travaux effectués sur ces questions par l'IRIT et l'IRISA dans le cadre du projet ILIAD sont loin d'avoir épuisé le domaine de recherche, et il s'agit probablement d'un des axes de recherches majeurs en ce qui concerne l'architecture des systèmes multiprocesseurs à mémoire partagée. Il s'agit en outre d'un domaine d'étude tout à fait complémentaire de celui concernant les réseaux d'interconnexion.

La modélisation apparaît comme un outil important pour prédire ou évaluer les performances des réseaux d'interconnexion des machines parallèles, ou déterminer leurs propriétés. Le projet ILIAD a permis aux équipes du groupe Rumeur travaillant dans ce domaine de prendre en compte un certain nombre de problématiques dégagées dans différents domaines des communications, et en particulier au niveau des communica-

tions optiques. En ce sens, on a pu assister à un rapprochement entre un domaine de recherche resté jusqu'ici assez théorique, et des domaines de recherches plus appliquées. On peut en attendre une meilleure prise en compte des problématiques dégagées par les technologies nouvelles, et un apport mutuel aux équipes travaillant dans ces domaines.

Le projet ILIAD a fait naître une importante synergie entre les équipes participantes. Un effort important a été consacré, en particulier durant la première année, à la mise en commun des problématiques et des notions de base propres à chaque équipe. Des collaborations plus approfondies sont apparues entre certaines équipes, parmi lesquelles on peut mentionner les collaborations entre les équipes de l'IRIT et de l'IRISA, ou entre Rumeur, l'IEF et l'IOTA, qui devraient se poursuivre au delà du projet.

Projet ILIAD :
Impact des liaisons séries haut débit sur l'architecture des machines parallèles

IRIT, équipe M3S,
IRISA équipe CALCPAR,
IEF département AXIS,
IOTA,
MASI équipe CAO et VLSI,
LGI groupe GRAM,
Groupe Rumeur de PRS

26 février 1996

Programme de recherches Inter-PRC 93

Thème : Nouvelles technologies pour l'Architecture

Annexes

Contenu :

ANNEXE 1
Application des liens série dans les multiprocesseurs à mémoire partagée, et évaluation des mémoires multiport série intégrées

Pascal Sainrat
Dominique Carrière

20 février 1996

IRIT, équipe M3S

Cette annexe présente d'une part l'application des liens série dans les multiprocesseurs à mémoire partagée et d'autre part une évaluation de mémoire multiport série intégrée.

Table des matières

1 Application des liens série dans les multiprocesseurs à mémoire partagée

1.1 Introduction

Multiprocessors are the natural solution to solve performance problems. Here we will examine open systems, capable of offering a scalable performance adapted to the most widely differing needs. The shared bus is the oldest and most commonly used approach for the interconnection network. This solution has often been criticized, particularly because a shared bus has a fixed bandwidth, on the one hand much larger than needed in the case of a small configuration, and on the other hand limiting the performance in the case of a large configuration. However, the shared bus has properties which make it attractive, especially in the context of current processors which all use caches. The problem of data coherency, produced by the caches of a multiprocessor system is solved simply in shared busses: it behaves like a public place where all operations are carried out openly and publicly, which constitutes a very particular point of synchronization.

Two types of shared bus must be distinguished: those which, like VME [1] or FUTUREBUS+ [2], create a platform which can adapt itself to the majority of the industrial world's needs, and those like XDbus [3] from Sun which are fully optimized to obtain a symmetrical multiprocessor, that is, made up of identical processor boards, with the obvious exception of I/O boards. However in both cases the quest for performance, stimulated by the evolution in technology and needs, leads to a permanent evolution in bus structure and technological advances which do not simplify compatibility problems. Important technological progress implies redefinition and compatibility problems as in the example of the upgrading of FUTUREBUS to FUTUREBUS+. Nevertheless, as regards the shared bus, this technological evolution is often quite limited: use of more powerful bus drivers to reduce an operation's cycle time, and widening busses to increase the amount of data transmitted each cycle. By using private paths to transmit information, workload problems can be solved locally. This implies that the boards concerned must be situated nearby.

The performance of a shared bus is often given in terms of information bandwidth, without taking into account the nature of this information (i. e. addresses or data). The useful bandwidth is the bandwidth where only data is taken into account, addresses being considered simply as a means of access to the data. The useful bandwidth gives a more precise idea of bus performance.

If the very different nature of addresses and data is analyzed, another approach is possible, which consists of using separate paths for each type of information. An address bus will not exceed 40 bits in the foreseeable future and a shared bus is perfectly well suited for address transfer in a multiprocessor environment. Moreover, if the bandwidth of a bus is only used for address transfer, larger multiprocessor architectures can be built. On the other hand, scalable busses must be available for data, able to move large blocks of data, perhaps differing in size, and working simultaneously due to the presence of several processors, which supposes the existence of several data busses. Nowadays this approach is possible, as we will demonstrate, by using ultra high speed serial links (UHSSLs), each one being, in fact, a virtual parallel bus.

Firstly, a basic architecture, using a shared bus for addresses and a set of UHSSLs for data transfer, will be presented. Next, the characteristics of today's shared busses

and their foreseeable evolution will be shown in order to target their limits. The problems inherent in very high speed data transfer will then be described. Conclusions will be drawn from the above in order to propose an architecture with a shared address bus and private serial links for data. The consequences of such an approach on the design of the main memory will then be highlighted. Finally, the use of ultra high speeds for information transfer will be examined, and in conclusion, we will look at the ways in which other hurdles could still be overcome, taking into account prospective developments in technology.

1.2 Conceptual approach

Let us consider a shared bus multiprocessor with the following characteristics:

- The shared bus is only used for address transfer. It is made up, in a classical way, of all the lines found, for example, in a VME bus. A transaction on this bus is a simple address transfer.
- Boards connected to this bus are either processor boards (used for processing or for I/O) or memory boards.
- A processor board consists of one or several high-performance microprocessors with cache memories. The board can include other resources such as memory, but this will be assumed to be private, i.e. inaccessible through the shared bus. So, in its simplest form, the board seems, as regards the shared bus, to be made up of a single cache. Thus, the transfer unit between a processor board and a memory board can still be a cache block.
- A memory board is composed of memory circuits associated in such a way that the memory access unit is the block. In order to simplify the presentation, let us assume that our multiprocessor system has only one memory board.
- Each processor board is connected to the memory board by a private serial link. This serial link is only used for cache block transfer. We thus assume initially that there is a shift register, whose size is that of a block, at each end. Transfer through the serial link seems to be a register to register transfer. Since block transfer must be possible in both directions, the transfer device must be able to work in both directions.

Figure 1 gives a synoptic diagram of this multiprocessor.

1.2.1 How does it work?

In the case of a cache miss, the address of the block requested is transmitted to the memory board via the shared bus. The block is read from the memory and loaded in the memory shift register associated to the requesting processor board, then transferred to the processor board shift register through the serial link and finally transferred into the cache memory.

In the case of a block flush, the block is first transmitted through the serial link into the memory shift register, then a write request with the memory address concerned is

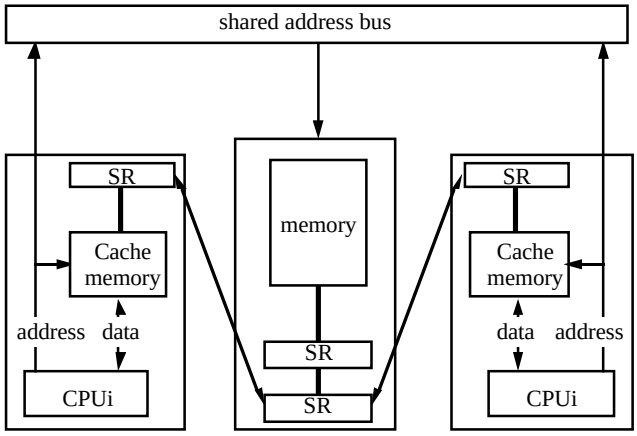


FIG. 1 – *Shared Address Bus and Serial Data Bus Multiprocessor*

transmitted through the shared bus. This request initiates the write of the block into the memory.

The performance, complexity and cost of this kind of architecture are examined below.

1.2.2 Bandwidth

Let us first consider the bandwidth of the address bus. It is allocated wholly to address transfer. The bus has exactly the right width for address transfer. Moreover, it does not seem necessary to seek a maximum performance which would not be used here and therefore a backplane using simple and economical technology is sufficient. From figure 1 we can see that if the whole bandwidth of such a bus is used for addresses, the performance limit relates to the memory. We still assume a single memory board made up of 2 interleaved banks of dynamic RAM with an apparent access time of 70ns. For a data access which is the width of a cache block, in the best case an access is handled every 70ns. The frequency of the shared address bus is then less than 15 MHz. This point will be re-examined in Section 1.4, and we will see that this "low" frequency is attractive for other reasons.

For the data bus bandwidth, we will consider serial links running at a rate of 1.6 Gbit per second, which is the rate of AUTOBAHN circuits [4] proposed for the VME bus. As figure 1 shows, each processor board has its own serial link, each providing a bandwidth of 200 MB/s. For an N-processor machine, the total bandwidth (200*N MB/s) available for data is therefore far superior to all current solutions: 160 MB/s for the 64-bit VME bus in SSBLT mode (Source-Synchronized Block Transfer) citebib5, 250 MB/s for the 64-bit XDbus and 2.5 GB/s projected for a 256-bit bus running at 80 MHz [3]. However, in our example, the bandwidth remains theoretical since it is limited by the main memory. For the values given here and a block size of 32 bytes, the memory bandwidth is still under 500 MB/s. Nevertheless, this result is interesting since it shows that the interconnection network is no longer a bottleneck. This is now transferred to the memory. The consequences of this will be analyzed in Section 1.7.

1.2.3 Latency Time

In order to compare the latency time of the architecture proposed here above with that of a conventional shared-bus architecture, it is sufficient to analyze the data transfer part, since the address transfer part is identical. The figures given in the previous section concerning bandwidth allow the data latency time to be determined for a classical bus: with a bandwidth of 250 MB/s, 64 bits are transferred every 32 ns. Transferring 64 bits serially at 1.6 Gbit/s takes 40 ns. It can be concluded that the latency time for 64-bit data is almost the same in both cases. This comparison supposes that the shift register is designed in such a way as to supply a word of 64 bits as soon as it has arrived. This corresponds to operational reality, as will be seen in Section 1.5.

Serial transfers are thus carried out as quickly as parallel transfers. The explanation is simple in that it is sufficient to go n times as quickly in the serial case, n corresponding to the number of bits transmitted in the parallel case. Now, for reasons that will be analyzed in the following section, a parallel transfer on an physically loaded shared bus will never exceed some tens of MHz. Let us also consider that a 64-bit width is a good compromise which optimizes the use of a classical multiplexed shared bus, for addresses and data. A shared bus with BTL-type drivers citebib5 seems to have an upper limit of 100 MHz. In this case, a 64-bit data bus is equivalent to a 6.4 Gb/s serial link. Such an high bit rate seems well over current technological capabilities, but is in fact compatible with current transmission media like optical fiber. Moreover, it could be compatible with current technology if advanced techniques, which are not yet applied to digital design, were used. In Section 1.8, it will be seen that above a certain rate, the balance between the various components of the multiprocessor leads to the busses being considered as narrower as their performance level increases.

1.3 Conventional Shared Bus: Characteristics And Perspectives

1.3.1 Electrical Characteristics

A shared bus uses a set of transceivers plugged in parallel onto a more or less electrically well-adapted line in order to smooth reflections as much as possible . Dynamic behavior of signals is complex and non-deterministic in the case of an open shared bus. Thus, we must consider a settling time which must be added to other times (driver propagation delay time, rising time, etc...). A detailed analysis can be found in [6]. An physically loaded shared bus, that is with a large number of resources connected in parallel, will therefore always have a performance level very different from that of a perfectly adapted link. Such a link must be a point to point one, for physical reasons. Improvements of the shared bus are nevertheless possible: use of trapezoidal signals, lower voltage swing, low-consumption technology. However, these features need an improved backplane (with, for example, carefully designed groundlayers) and thus a higher cost. A detailed analysis of these points can be found in citebib5.

1.3.2 Logical Characteristics

Characteristics of shared busses are drawn from microprocessor busses: synchronous or asynchronous timing, address and data pins multiplexed or not. A good level of performance requires synchronous timing, which leads to complex central clock distribution problems over a certain frequency. The length of the bus can also lead to

bus segmentation and thus to pipeline transfer. The use of a multiplexed bus optimizes the use of pins, limits the number of drivers and therefore their consumption, and is of little consequence upon performance because addresses and data are never needed simultaneously by the memory, considering a classical non split bus.

1.3.3 Bus Arbitration

The shared bus is a critical resource whose access must be strictly controlled. Only three arbitration techniques exist:

- Serial arbitration ("daisy chain"): a simple, economical and not very powerful solution.
- Parallel arbitration: a powerful solution which needs a pair of signal lines on the backplane for each request and which allows various bus management schemes to be used.
- Associative (Distributed) arbitration: a solution working by successive elimination. This type of arbitration has interesting properties which have generalized its use. It is well documented in [7].

It must be pointed out that the arbitration time is not negligible and increases with the number of bus requesters. This time is partially overlapped if the arbitration can be carried out in parallel with a transaction already under way. It certainly seems that few improvements can be expected in this area.

1.3.4 Transaction Modes

A connected transaction occupies the bus for the whole duration of the transaction. For example, a memory read can be separated into: address transfer - memory read - data transfer. With connected transactions, the bus is busy during the memory read which is the longest operation of the transaction. A split transaction splits the request and response into separate bus transactions. Thus, the bus can be free and used by another master during a memory read. The split transaction is composed of two (or more) connected transactions and each connected transaction needs an arbitration. Bus resources are used in a more efficient way with split transactions.

1.3.5 Cache Coherency Protocols

A bus is a convenient device for ensuring cache coherency because it allows all cache controllers in the system to observe all memory transactions. Thus, any request placed on the address bus and concerning an operation on a cache block lets all other caches check their own cache directory and provokes operations which maintain data cache coherency. This supposes that, during the address transfer time, all caches should be accessed. This mechanism can lead to an operation on the shared bus coming from another processor rather than the processor which initiated the transaction. Two policies have been practised for maintaining cache consistency : write-update or write-invalidate. These policies are analyzed in [8].

1.3.6 Evolution

The evolution of technology and the coming of new ways for the implementation of such techniques as programmable logic lead to the conclusion that printed circuit boards will, in the future, have a small number of different, but highly integrated and powerful circuits: microprocessors, specialized circuits, programmable circuits for glue, and, of course, memory circuits. A board plugged onto a backplane will therefore be itself a multiprocessor and its communication needs with the other boards in the shared bus will be ever-increasing, considering only the effect of clock frequency growth.

The time needed to go through the shared bus will present more and more of a penalty, expressed in processor clock cycle units. The VME bus was, at the outset, designed for a 68000 processor running at 4 MHz, so its frequency was in the range of a few MHz. Its current theoretical limit is 20 MHz citebib5, whereas processors run with 50 to 200 MHz clocks.

The time needed for a memory read through the shared bus has decreased from 400 to 200 ns while the frequency of processors has increased from 4 to 100 MHz. The performance ratio (bus transaction time over processor cycle time) has been increased from approximately 2 (400 ns and 4 MHz) to 20 (200 ns and 100 MHz)! and the gap is still widening.

The overhead to fetch a single word from the memory via the shared bus is high and is almost the same for a word as for a block of words accessed in burst mode: improved bus efficiency is thus obtained with multiple data transfers. This increases the number of bus access types, as can be seen in the analysis of FUTUREBUS+ [2] or the XDbus [3].

In conclusion, the gulf existing between the communication needs and the actual performance of a shared bus is widening drastically. Everything possible is done to reduce the gap in performance between the various components of the system, but the solutions used are simple, and remain highly insufficient: using more powerful transceivers, increasing the volume of data transmitted at each transaction by playing on the width of the data bus and favoring data block transfer.

1.4 Shared Address Bus

If data is transmitted by another path, constraints on the shared bus (in this case used only for addresses) are largely relaxed.

The first consequence is that signals are unidirectional, which gives better electrical characteristics for the bus lines.

The second is that whilst only addresses are transmitted, the advantages of split transaction protocol can be reaped without the disadvantages, i. e. the memory does not have to request the bus. The number of requesters is strictly limited to the number of processors. Since there are less requests on the bus, an address transfer can take more time. The cache coherency mechanism can then be much simpler. For instance duplication of the cache directory can be avoided. This does not hamper performance: memory access can be overlapped with checking of the snoopy cache controllers' directory.

However, if the shared bus is only used for address transfer, then only write-invalidate algorithms can be implemented, since it is no longer possible to transmit a single word. Write-once protocols or their equivalents must be adapted to this new context, which does not pose any particular problem. Nevertheless, a shared data bus

can exist purely for words. Then, these protocols can be used again. This point is discussed in Section 1.6.2.

Even if bandwidth needs are limited for an address-only bus, it is obvious that, the more powerful the address bus is, the lower the latency time.

1.5 High Speed Serial Links

The method for transferring data illustrated in figure 1 is conceptual and is not adapted to current technology and components. However, specialized circuits have the same behavior. They implement a virtual parallel bus, as shown in figure 2.

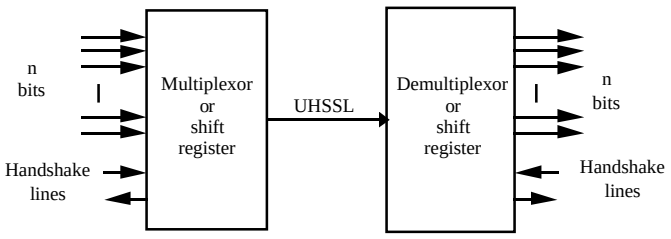


FIG. 2 – *A Virtual Parallel Bus*

Various circuits exist, with characteristics differing according to manufacturers: Gigabit Logic [9], Triquint [10]. Vitesse Technology [11], Hewlett Packard [12], Bull [13], PEP Modular Computer [4]. Here we will look neither at the way in which they work (by bit multiplexing or shift register), nor at internal roles which they fulfil (data encoding, clock extraction, error checking etc.) but at their use.

The use of a single, and therefore bidirectional serial link leads to considering transmitting and receiving logics as being integrated in the same circuit.

We will use a conceptual circuit representing these different circuits in the following. As shown in figure 2, n-bit words are transmitted from the transmitter circuit to the receiver. Two handshake lines ensure data transfer synchronization. Transmission speed on the serial link and word rate are linked by the value n. If n=64 and the serial link rate is 1.6 Gbits per second, then a 64-bit word is transferred every 40 ns (25 MHz). Such a frequency is consistent with today's dynamic RAM memories used in nibble mode. Figure 3 shows the detail of a serial link used for reading a cache data block in this context.

On a cache miss, the address of the block owning the requested word is sent to the memory board with the attribute "block read". The block is read in a traditional way in nibble mode, i. e. with a normal access time for the first word and a shorter access time for the three next words, modulo 4. The by-pass allows, traditionally, the first word to be passed to the processor in order to free the last one as soon as possible. Thus the same level of performance is achieved as with a traditional 64-bit shared bus working at 25 MHz.

The figure is partial and only shows the read path. A pair of identical transceiver circuits must be used for the write, with equivalent functions.

The use of such high frequencies has, of course, very serious implications upon their operation. However, as shown in figure 3, their use is very limited: the coaxial cable (or optical fiber) must be placed in immediate proximity to the corresponding pin (or

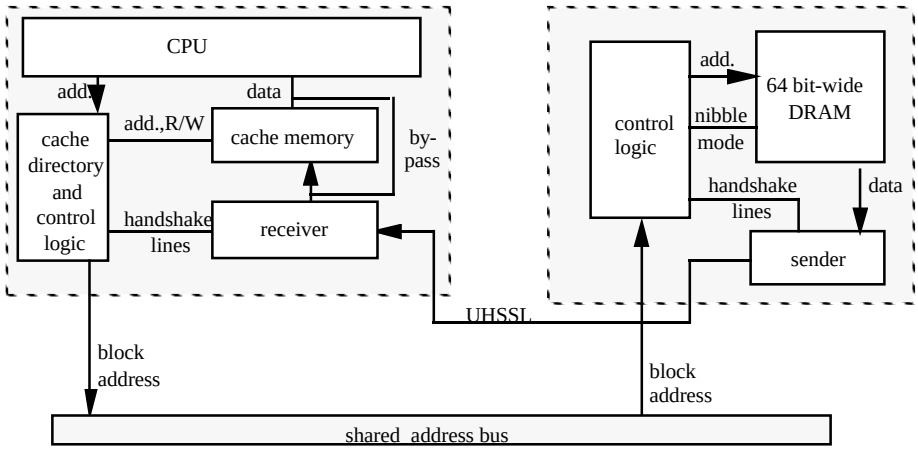


FIG. 3 – *Details of a Block Memory Read*

the two pins in the case of a differential driver) of the circuit and this short link must avoid EMIs by appropriate lines and groundlayers.

The cost of these circuits is certainly high, and this solution is not very economical at the present time. However, the situation should develop very quickly with the intensive use of these circuits in various areas such as telecommunications. The additional cost should soon be compensated for by the use of a shared address bus using traditional and economical technology.

As we have just seen, a single serial link has the performance level of an ordinary shared bus. The shared address bus becomes much less loaded. We now find ourselves with under-used resources in the case of a multiprocessor with a small number of processors. On the other hand the memory becomes a highly critical resource as it quickly becomes saturated. We will now look at the implications of the above.

1.6 Load Analysis

1.6.1 Serial Links

The analysis in the previous section seems to show that serial links used for data transfer are under-used. Indeed, a conventional shared bus is able to serve several processor boards, each UHSSL having a bandwidth equivalent to that of a shared bus and each processor board having its own private serial link with the memory. This is true for the case of a single processor board. It is not possible for a UHSSL to connect up several processor boards since performance depends precisely on the point to point aspect of the link. Nevertheless, the sharing of the UHSSLs can be carried out in a different way. As we have said, the evolution in technology leads to logic becoming denser. Several processors can be placed on a double-size Eurocard sharing transmission circuits. Studies carried out [14] have shown that four ordinary processors working without any local memory other than caches, use almost 100% of the bandwidth of a 1 Gbit/s serial link. The use of shared local memory must lead to a reduction in this workload, but the serial link can or must in this case be used to transfer information blocks from memory to memory. The conceptual aspect remains the same.

1.6.2 Shared Address Bus

The shared address bus used purely for addresses is simple due to the fact that the number of address bits is small (40 bits can address several Terabytes of memory!). However the absence of data busses prevents any write-update coherency protocol from being implemented and also necessitates a slight change to write-invalidate protocols. As long as a processor write only concerns one word at a time, a shared address bus equipped with a data bus solely for processing one-word writes would allow all features of ordinary shared busses to be retrieved. In this case, the bus should not be multiplexed: data transfer can be carried out simultaneously, since any operation is always a write. In the following section, we will show that it is desirable that all memory accesses should be block-sized accesses. So, a shared-memory multiprocessor system needs shared variables to ensure mutual exclusion for resources, resource sharing or simply process synchronization. These variables are bytes (locks) or 32-bit words (pointers). Placed in cacheable areas, these variables often lead to coherency actions and are subject to false sharing (several independent variables allocated in the same block) [15]. It is therefore desirable to consider a shared memory with low capacity, non cacheable, reserved for these variables, and that can be accessed through the shared bus equipped with its separate data bus. It must be noted that this type of access does not lead to coherency actions: it could therefore be interesting, above a certain workload, to consider separate busses.

1.7 Memory Organization

Firstly, we will examine the case of a system having only one memory board. This case would be the most common, taking into account the evolution in technology and board formats which allow a large amount of memory to be placed in a small area, with a certain capacity for scalability. Each (multi)processor board has a private UHSSL with this memory board: the latter must therefore be organized in a serial multiport way, i. e. linked to a set of N transceiver pairs, N corresponding to the number of UHSSLs, as shown in figure 4. We will look at the case where N=16, a value which seems reasonable today, and corresponds to 16 boards with, for example, 4 processors.

The memory is now the bottleneck. The set of UHSSLs indeed allow simultaneous transfer. If the shared address bus runs at 20 MHz, a memory request can be made every 50 ns. In Section 1.5 we looked at a 64-bit memory width, and a serial throughput of 1.6 Gb/s with a block size of 32 bytes. We have seen that, in steady state of nibble mode, the memory and a serial link have an equivalent speed and that the duration of a transaction is therefore 70+3*40 ns (70 ns for access to the first word, 40 ns for access to the 3 next words, in nibble mode) or 190 ns: the shared bus and the set of serial links are under-loaded. In order to better exploit the available bandwidth, memory bandwidth must be increased. For identical technology, the only way of doing this is to enlarge the memory width. However, in this case, each transaction must exploit this wide word. The ideal solution would be a memory width equal to the size of the cache block, 32 bytes in our example. In this case, only one memory access corresponds to each memory transaction, but nibble mode can no longer be exploited and the dynamic RAM's cycle time must be taken into account (which we have overlooked up to now): in fact a single access cannot take place more than every 150 ns (DRAM cycle time), in view of all these constraints. It is therefore desirable to use two or several interleaved

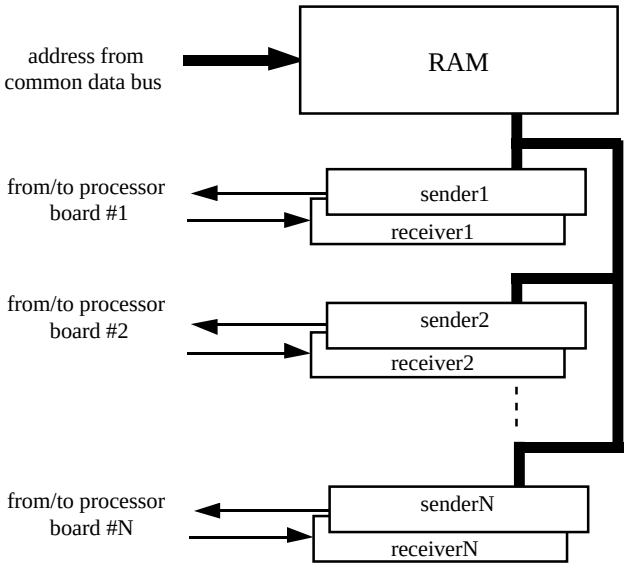


FIG. 4 – *A Serial Multiported Memory Board*

memory banks.

The other approach, allowing nibble mode to be exploited means considering memory banks with a width four times narrower than in this case, and necessitates a crossbar of busses between the banks and the set of transceiver circuit pairs. Whatever the complexity of this solution, it is conceivable because the time spent establishing the path in the crossbar, still considerable, can be overlapped with the access time to the first word of the block in the bank.

This brief analysis shows the different levels of performance that can be envisaged, and which constitute a form of scalability in the memory, as well as the variety of solutions imaginable. At the lowest and simplest level we find the example discussed in Section 1.5 whose performance nearly corresponds to that of today's conventional shared busses. The highest level is that where all serial links work simultaneously: in this case the memory bandwidth must be equal to that of the set of serial links, and the block size transmitted must be increased in order to overcome the limitation of the shared address bus bandwidth.

Let us analyze the latter by considering bidirectional serial links: the maximum useful bandwidth is therefore 16*1.6Gb/s, or 3.2GB/s. For a cycle time of 150 ns (a single memory bank), memory must have a 480-byte width to keep up with this speed. At 20 MHz, the shared bus has a cycle time equal to a third of that of the memory: each transaction must therefore correspond to a third of this memory width, which, in turn, corresponds to the size of an information block, or 160 bytes... It can thus be seen that the shared address bus once more becomes the bottleneck for these limit values. Taking into account the small number of wires in the latter, a technological change in this case is conceivable and a bandwidth of 100 MHz allows reasonable values of 32 bytes per block to be moved.

These figures are only theoretical. It is, of course, undesirable to reach these workloads. However these peak values give an idea of the performance level which can be reached. We have looked at the particular case where the memory is made up of a

single memory board. If we consider that the memory is made up of several boards, it becomes necessary to have available as many sets of serial links as memory boards.

The problem of scalability of the number of links can be resolved simply by using daughter boards, both in the processor and in the memory. The memory scalability problem is conceivable with regard to the board, as described above, and also when adding memory boards. However, taking into account the evolution in memory circuit density, this case will be less and less necessary. Moreover, as we have shown, the shared bus will once more become a bottleneck and other architectures will have to be envisaged. The exclusive use of serial links for data and address transfer, and therefore without shared busses, leads to another type of multiprocessor architecture which allows a massively parallel approach to be envisaged, presented in [16].

1.8 Bus Versus Uhssl Evolution

Undoubtedly, a point to point link will always have a better performance level than a multipoint link. In addition, the temptation to put several serial links into parallel in order to build a shared bus and increase the bandwidth is natural, but difficult to realize and curiously useless above a certain frequency, as we will now illustrate. We consider here frequencies up to 10 GHz, an numerical extreme at present, but for whom operational feasibility studies are being carried out in the realm of telecommunications [17].

The first problem met is that of synchronization. Realizing synchronous transmission supposes the distribution of a central clock to the whole system, which is not easy: at 1 GHz, the electric wave travels 20cms per nanosecond along a copper medium and it is therefore necessary to take account of these distances however small they may be. Transmitting a clock on a link associated to the bus partly solves the problem, but it is still necessary to take the bit skew into account, which takes place on each of the links and thus in this case on that carrying the clock. This last problem can be resolved by using one of the data links to transmit the clock, according to a now traditional encoding mechanism such as that used in FFDI. However, this link must function at a slightly higher frequency than the others and creates an imbalance which, what is more, does not solve the problem of bit skew. The ultimate solution is therefore to transmit the clock with each bit, a solution which is probably mandatory above a frequency of a few GHz. This comes down to considering this bus as a set of serial links.

The second problem met is that of bus efficiency. Let us consider a shared bus working at such high performance levels -if that is possible-. A transaction bus includes the following steps:

1. Bus arbitration, that we will take as having a fixed value of 10ns.

2. Transfer of the address to the bus, address on 40 bits.

3. Memory read or write. We consider a fixed value of 40ns.

4. Data block transfer, that we will consider as 64 bytes.

Table 1 gives us, for some pairs of bus size and frequency values, the efficiency E of the bus: relation between the time spent on the bus and the total transaction time.

Frequency & Size	Bandwidth	Addr. transf. time	Data Transfer time (D)	Latency time (L)	Efficiency (E=D/L)	Number of transfers(T)
500MHz 8 bits	0.5 GB/s	10 ns	128 ns	178 ns	0.72	0.29
500MHz 64 bits	4 GB/s	1.25 ns	16 ns	57.25 ns	0.28	2.3
10GHz 1 bits	1.25 GB/s	4 ns	51.2 ns	95.2 ns	0.54	0.72
10GHz 8 bits	10 GB/s	0.5 ns	6.4 ns	46.9 ns	0.14	5.8
10 GHz 64 bits	80 GB/s	0.06 ns	0.8 ns	40.86 ns	0.019	46.5

TAB. 1 –

It also gives the number T of possible block transfers during a memory access and the latency time L.

At the extreme value of 10 GHz, the efficiency R of a wide bus is low. The latter can, in part, be improved by pipeline transfer, but in this case the memory bandwidth must also be increased, as shown in the column giving the number of possible transfers T during an access. This notion of efficiency must be associated with that of latency time, which must remain reasonable, which is the case shown in column L. This efficiency calculation shows that the bandwidth is badly exploited: it is, then, better to consider several narrow busses, which, besides, increase connectivity. At these speeds (and above), and taking into account a large multiprocessor system, wouldn't it be better to have N UHSSLs available rather than a single N-wire multipoint bus, which in any case would never reach a similar performance level?

1.9 Conclusion

Using busses at their best, and taking into account the evolution in performance needs, leads naturally to their specialization and the use of shared busses for addresses and private busses for data. As has been shown, the use of UHSSLs as private busses brings an elegant solution to the problem of bandwidth, without hampering latency time, and brings an additional advantage with regard to maximum connectivity.

The rapid evolution in technology necessitates technological steps, small steps such as bus enlargement, and leaps such as a reduction in voltage swing, which mean that compatibility cannot always be ensured. Undoubtedly other important technological changes are to be expected. The operation of signals working above 50 MHz on an ordinary printed circuit board already necessitates specially-adapted lines as soon as the wires are more than a few centimetres long. Above a few hundred MHz, it is the medium itself which will have to be closely examined. In that case, the distance between chips will also have to be looked at. The solution to these problems exists through Multi-Chip Modules. We will then come up against the problem of communication bandwidth for the environment and therefore connections. As said in the previous section, a set of serial links is the natural way to solve this problem. In the long term, electrical transmission will itself be a limiting factor: photonic transfer will allow output to be improved by 1/3. There also the serial medium will find its natural place.

If then, we consider the problem of communication bandwidth as being resolved,

the same cannot be said for that of memory bandwidth. For more than thirty years, memory has had the same structure. New organizations are appearing: Cache DRAM, Synchronous DRAM, RAMBUS, etc. [18] [19] [20]. Another step must be taken. Indeed, by the end of the decade, memories of 1 Gbits in capacity will be available. They will have to be wider and wider, in order to maintain a certain bandwidth [21]. However, is it still conceivable that such a large amount of memory should only be accessible by a single entry point? The solution involves multiport memory components, and the limitation of the number of pins naturally suggests the use of serial links.

It must also be said that this problem of bandwidth can also be found in peripherals such as disks. The information is available in a serial form on the medium, and serial coupling is also a natural way to link a set of disks to a set of corresponding processors with a high current need [22]. The serial connection of peripherals has begun with SCSI3 [23] and SSA from IBM [24] and FireWire [25]. This problem of I/O bandwidth can be found everywhere as shown in the development of the VXI bus [1]. There also, the high speed serial link brings original solutions which lift the barrier of performance.

Finally, an important point which has not been discussed here is reliability. For example, UHSSLs can easily be backed up. Direct links between processor boards as well as memory boards can also exist. Furthermore, each serial link represents a complete link between processor and memory: graceful degradation protocols can take the place of a faulty shared bus.

It seems then that the UHSSL could be a promising way to resolve performance problems in shared-memory multiprocessors, and, moreover, creates a bridge with distributed memory systems using networks such as for example SCI [26].

2 Une organisation de DRAM Multiport Série

Dans le domaine actuel des multiprocesseur nous atteignons des limites d'interconnexion sous forme de bus commun au fur et à mesure que les vitesses de processeurs s'accroissent et que les mémoires augmentent en capacité. De nouvelles technologies de DRAM sont apparues. Elles offrent des solutions au problème de débit, mais exploitent encore mal les possibilités intrinsèques du coeur. Après avoir présenté quelques caractéristiques des DRAM vis-à-vis des besoins des multiprocesseurs au §I, nous présenterons une architecture nouvelle au §II. Une implémentation sera présentée au §III ; les résultats d'un calcul de performance au §IV. Enfin, nous concluerons sur les perspectives de développement de ce module mémoire. La gestion de cohérence est écartée de notre domaine d'étude dans cet article ; des précisions sur cet aspect peuvent être trouvées dans [27].

2.1 Les Mémoires Actuelles Vis À Vis Des Multiprocesseurs

A moins de vouloir y consacrer de très gros moyens, les mémoires centrales sont réalisées en DRAM (Dynamic Random Access Memory) plutôt qu'en SRAM (Static Random Access Memory) trop consommatrice en transistors par bit implanté mais pourtant plus rapide. Si ces DRAM ont évolué dans le sens d'une plus grande intégration, le principe d'utilisation était resté le même depuis 1973 [20] ; mais récemment, de nouvelles possibilités ont été adjointes. Les nouveaux modules mémoires tels que les "synchronous RAM", les "Cache RAM" et les "Enhanced RAM" peuvent toutes trois débiter après un temps d'accès de 50 à 80 ns, à une fréquence de 50 MHz et plus [19]. La RAMBus, elle permet un fréquence d'entrée-sortie à 500 MHz, en utilisant d'autres modes d'interfaces bus commun. Elle permettra aussi d'implanter un système multibanque facilement, pouvant servir plusieurs requêtes (Ramlink...). Pour fixer les idées, sur les boîtiers DRAM les plus récents, les débits disponibles les plus grands en mode "fast page" n'excédaient pas 20 Mbit/s.

Tous ces nouveaux modèles ont en commun de contenir un noyau de DRAM classique avec des amplificateurs différentiels "sens amplifier" qui débouchent sur des SRAM. Tous exploitent d'une manière différente le nombre important de bits en parallèle sur une ligne de ces DRAM (1024, 2048, 4096 ou 8192). Cette quantité de SRAM est exploitée par accès rapide et consécutifs à la même ligne et permet donc un débit rapide sans décodage coûteux en temps. En ajoutant un peu plus de SRAM on peut arriver à la construction d'un cache interne d'1/1024 de taille, et donc créer un cache de second niveau avec un faible surcoût en transistor ("cache DRAM").

Quoiqu'il en soit, les modules mémoires ainsi créés n'exploitent pas toute la bande passante disponible au niveau de la ligne d'une DRAM, supposons 120 ns de temps de cycle pour 1024 bits/ligne, de l'ordre de 1024 / 120 ns = 8 Gbit/s. Il faut comparer ce débit à celui des implantations de "Cache DRAM" à 100 MHz de débit 100 Mbit/s. Ce débit ne tient pas compte du temps de monopolisation du bus externe de la DRAM pendant le temps d'accès (les "synchronous DRAM" permettent de réduire cette monopolisation de façon importante en synchronisant les accès sur une horloge externe).

Une caractéristique essentielle de ces mémoires est que les temps d'accès sont toujours apparents : ils ne peuvent être recouverts ou "pipe-liné" avec d'autres transferts.

Dans une organisation multiprocesseur où l'on a besoin d'exploiter la bande passante maximum du bus, l'organisation en banque ou l'entrelacement est impératif et donc, l'implémentation sera complexe.

Pour des multiprocesseurs à mémoire commune à grands nombres de processeurs, un problème de connectique se pose. Comme il est expliqué dans [28], M3S propose une organisation mémoire multiport dont les supports de communications sont réalisés au moyen de liens sériels à haut débit. Nous proposons un modèle et une implantation possible de ce module mémoire en utilisant une structure de DRAM.

2.2 Une Dram Multiport

Nous proposons un module mémoire multiport sériel pour multiprocesseur. Sur le même principe d'exploitation des lignes de 1024, 2K, 4K ou 8K bits de long on peut créer un module multiport série dont la figure 5 présente un synoptique.

Outre la mémoire, tableau de cellules DRAM, et la ligne d'éléments d'amplification-comparaison-rafraîchissement ("Sense Amplifier"), ce module comprend : des registres de réception de requête (*reqn*), des registres de réception (*recn*)et d'émission (*emin*) des données, un module d'arbitrage (*Arb.*) et un module de séquence globale (*contrôle*). Sur le pourtour se trouvent les ports sériels d'Entrées/sorties. La taille des données transférées est le bloc de cache. Une ligne de mémoire DRAM contient plusieurs blocs.

Une requête venant d'un processeur est introduite *via* un port sériel dans un registre de réception des requêtes et présentée à l'arbitre. L'arbitre détermine à partir de la requête le N° de ligne et le N° bloc à l'intérieur de la ligne. Après l'élection par l'arbitre, un traitement est appliqué suivant la nature de la requête :

- lors d'une écriture, les données à écrire sont reçues dans un registre de réception. En parallèle et de manière à ce que les 2 processus s'achèvent ensemble, la ligne qui contient le bloc est sélectionnée dans la DRAM. Le transfert du registre de réception vers la ligne d'amplificateur va inscrire les données reçues en mémoire. Dans ce même temps de repos, les autres données non modifiées de cette ligne sont rafraîchies.

- lors d'une lecture, la ligne qui contient le bloc est sélectionnée dans la DRAM. Lorsqu'elle est disponible, le bloc voulu est recopié dans le registre d'émission voulu qui commence la transmission. de manière concomitante, la ligne qui contient le bloc est rafraîchie.

De manière générale, la DRAM est monopolisée pendant 3 phases : les temps de chargement parallèle du registre d'émission ou de déchargement parallèle du registre de réception à partir de ou vers la mémoire, le temps d'accès à la ligne de la DRAM et le temps de repos consécutif à cet accès mémoire. Le temps de réception et d'émission sérielle des données dans les registres peut être "recouvert" par les autres temps d'accès à la DRAM. Ainsi par recouvrement des temps de transferts des informations, ce moduyle mémoire peut servir plusieurs requêtes en même temps. D'où le qualificatif de multiport que nous lui appliquons.

Pour simplifier l'utilisation du composant, il existe un protocole d'échange de messages entre l'extérieur et la mémoire. Tout processeur émettant une requête attend le résultat avant d'en proposer une nouvelle. Dans le sens extérieur -> mémoire, une demande de lecture est composée de la commande lecture et l'adresse de la donnée. A cette requête, est répondu un message de réponse immédiatement suivi des données demandées. Une demande d'écriture sera composée de la commande, l'adresse de la

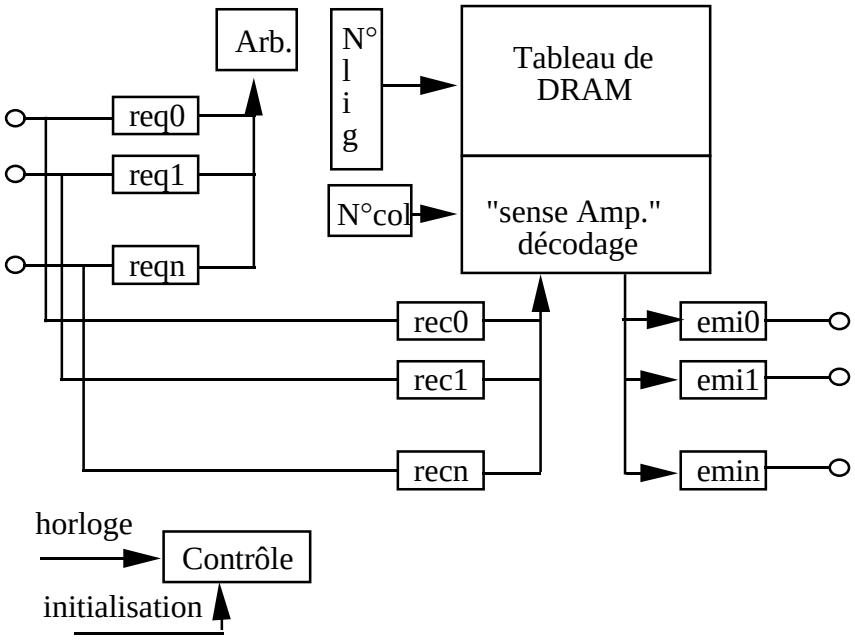


FIG. 5 – *Synoptique de la Mémoire Multiport*

donnée, suivie enfin de la donnée à écrire. Il sera répondu un message d'acquitement d'écriture.

On peut adjoindre dans le but d'accroître la sécurité de fonctionnement; à ce protocole un traitement des erreurs avec correction et des directives de chien de garde pour traiter le cas des trames perdues. Le rafraîchissement est géré au niveau de l'arbitrage et reste transparent aux composants extérieurs. Il serait facile d'optimiser ce rafraîchissement en fonction des accès effectifs (une méthode peu coûteuse est proposée dans [29]). La politique d'arbitrage choisie est le Round-Robin pour ces garanties contre toute famine vis-à-vis d'un lien donné et pour sa simplicité d'implantation [30].

2.3 Possibilités D'implémentation

Il est clair que nous n'avons pas l'envergure de constructeurs de DRAM, ni même d'entreprises de recherche et développement telle que RAMBUS. Nous n'avons ni les moyens, ni même la possibilité de réaliser une DRAM de 16 ou 64 Mbit. De même nous ne disposons pas des technologies d'intégration telle que les MCM qui permettent de réaliser des modules électroniques de grande dimension de manière plus fiable [31]. Mais pour étayer notre proposition, nous juxtaposons quelques faits technologiques.

Constatons d'abord, que les débits des convertisseurs parallèle/série réalisés en technologie CMOS ou AsGa ont augmenté. Dans le domaine des circuits intégrés comme dans celui des MCM (Multi Chip Module), les puissances dissipées dues tant aux E/S qu'aux portes constitutives baissent avec les tensions d'alimentation. Les tensions transportées à l'extérieur des circuits intégrés baissent aussi [19] et ce, sans augmenter les taux d'erreurs. En technologie AsGa, des registres d'E/S sériels à 2,5 Gbit/s sont disponibles [32]. En technologie CMOS, il est envisagé de réaliser des composants mémoires munis de débits à 500 Mbit/s [19]. Dans le domaine de l'intégration, des prototypes de

DRAM 256 Mbit ont été réalisés sans pour autant augmenter le temps d'accès car le plus grand chemin d'accès est voit le temps de transit réduit par la diminution de la technologie [20]. Dans le domaine des temps d'accès, en version 16 Mbit, 35 ns de temps d'accès pour une ligne de 2048 bits sont déjà atteints. Dans le domaine des interfaces de processeur, nous prenons pour exemple celui de Motorola de [33], qui permet le partage multiprocesseur bus commun. Ces informations juxtaposées nous permettent d'estimer les données suivantes :

La mémoire fait au moins 256 Mbit avec 35 ns de temps d'accès-ligne et 35 ns de repos. Suivant la technologie CMOS ou AsGa, les liaisons se font à 500 Mbit/s ou 2,5 Gbit/s. Le nombre de bits alloués à un bloc est de 512, soit 32 octets. L'arbitrage se fait en 20 ns pour au maximum 32 processeurs

Le tableau 2 reprend les principales caractéristiques obtenues.

Le temps de cycle minimum (vu de chaque processeur) est le même que le temps de latence puisque le temps de cycle interne de la mémoire est recouvert par l'ensemble des accès au module mémoire. Toutes ces caractéristiques ne tiennent pas compte du temps perdu dans les connexions d'accès au travers d'un réseau cross-bar, ni de la perte de performance due au rafraîchissement de la mémoire. Un rapide calcul montre d'ailleurs que ce rafraîchissement, pour toutes les DRAM comme pour notre module, ne coûte dans le pire des cas que 3,2 % du débit maximal.

	Liaisons BiCMOS	Liaisons AsGa
Débit	500 Mbit/s	2,5 Gbit/s
Temps d'accès au bloc	1150 ns	280 ns
Temps d'accès au premier mot	190 ns	90 ns
Nombre maximum de liaisons servies simultanément	16	3

TAB. 2 – *Caractéristiques de Communication*

L'aspect calcul de consommation a été écarté faute d'informations. Même si les technologies CMOS et AsGa consomment moins qu'auparavant et encore que l'usage de technologie MCM peut économiser 50 % de l'énergie dissipée par rapport au circuit imprimé [31], le type de boîtier MCM devra être en céramique pour pouvoir dissiper toute l'énergie produite par les différents modules !

2.4 Calcul De Performance

Nous avons étudié le cas d'un multiprocesseur à mémoire commune qui utiliserait un module mémoire tel que nous l'avons décrit. Nous avons établi ces simulations à partir d'un modèle analytique, basé sur des valeurs moyennes d'un microprocesseur 88110 [33], [34]. Partant des caractéristiques des deux versions de module mémoire précédents, nous les comparons à une solution bus commun utilisant des boîtiers DRAM courant. Les processeurs sont estimés sans cache externe de deuxième niveau. Nous n'estimons toujours aucun trafic dû à la gestion de cohérence. La figure 6 présente les résultats de cette étude. La liaison de type CMOS semble insuffisante et "peine" dès le début pour subvenir aux demandes de 2 processeurs. Par contre le module muni de liaisons AsGa fait montre d'une plus grande performance que le module de comparaison ; apportant ainsi un encouragement pour ce choix.

2.5 Conclusion

Nous avons montré qu'à partir des dernières évolutions techniques dans le domaine des mémoires DRAM, des MCM, des technologies CMOS et AsGa, de nouvelles organisations mémoires sont envisageables pour répondre au problème de débit et d'interconnexion entre processeurs et mémoires. Pour valider ce concept, il faudrait d'une part étudier l'intégration d'un mécanisme de gestion de cohérence et d'autres part concevoir le circuit DRAM ; Enfin, en utilisant d'autres circuits intégrés implanter et relier l'ensemble sous forme d'un MCM.

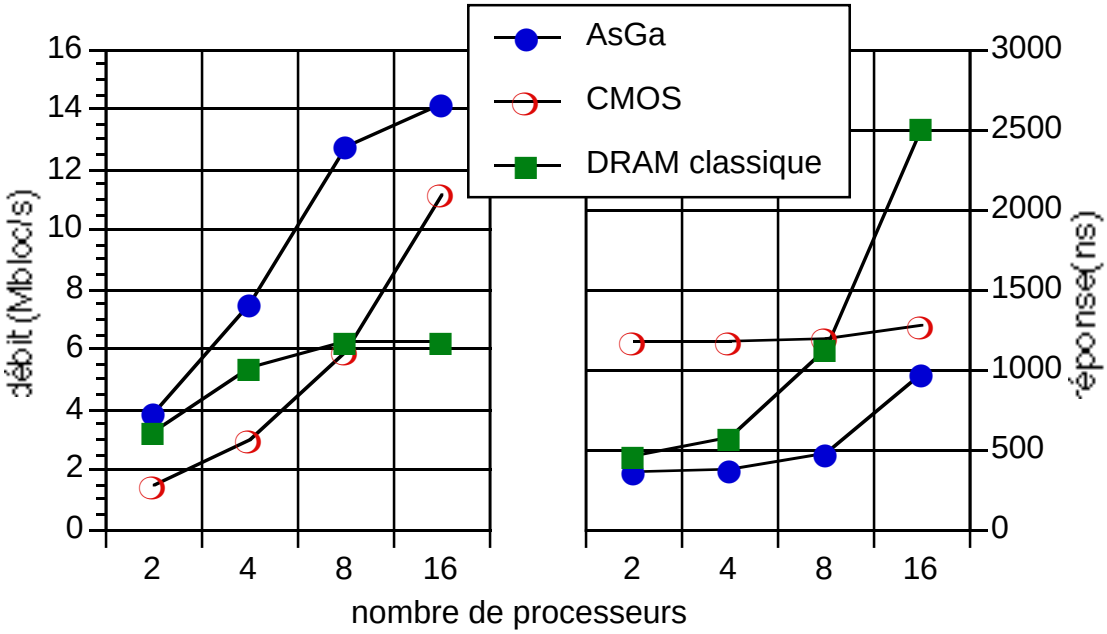


FIG. 6 – *débits*

Références

[1] "Spécifications du bus VME".

[2] FutureBus+, P896.3 . IEEE. New-York. 1991.

[3] P. Sindhu, J.M. Frailong, J. Gastinel, M. Cekleov, L. Yuan, B. Gunning, D. Curry, "XDBus : A High-Performance, Consistent, Packet-Switched VLSI Bus", Proceedings COMPON 1993, pp. 338-344.

[4] "1.6 Gbits/s Spanceivers Hit The Market" Electronic Design, July 11, 1994 Vol. 42, n° 14, pp 27.

[5] "Adding Value to VMEbus", VMEbusiness, september 1992. pp. 18-22.

[6] David B.Gustavson, "Computer Busses-A Tutorial" , IEEE Micro , August 1984, pp. 7-22.

[7] D. M. Taub ,"Arbitration and Control Acquisition in the proposed IEEE 896 Futurebus", IEEE Micro , August 1984, pp 28-41.

[8] P. Stenström, "A survey of Cache Coherence Schemes for Multiprocessors", IEEE Computer, Vol. 23, n° 6, June 90, pp. 12-24.

[9] Gigabit Logic, "10G040, 10G041 Data Sheet", GaAs IC Data Book & Designer's Guide, 1988.

[10] Triquint, "Hot Rod Data Sheet", Triquint Semiconductor, 1990.

[11] Vitesse, "G-Taxi VSC710x Data Sheet", Vitesse Product Data Book 1992.

[12] Hewlett Packard, "Hot-Rod GA 9011, GA 9012 Data Sheet", Optoelectronics Designer' Catalog, 1992.

[13] Bull, "Bull Serial Link Technology ", Bull 1992.

[14] C. Rochange, "M3S : un multiprocesseur à mémoire partagée évolutif - Premiers résultats de simulation", 6° Rencontres Francophones du Parallélisme, Lyon, 7-10 Juin 94, pp. 237-240.

[15] J. Torellas, M. S. Lam, J. L. Hennessy, "False Sharing and Spatial Locality in Multiprocessor Caches", IEEE Trans. on Computers, June 1994, Vol.43, N° 6, pp. 651-663

[16] D. Litaize, A. Mzoughi, C. Rochange, P. Sainrat, "Towards a shared-memory massively parallel multiprocessor" , 19th Annual International Symposium on Computer Architecture, Gold Coast, Australia, 19-21 mai 92, pp. 70-79. ACM SIGARCH Computer Architecture News, Volume 20, number 2, pp. 70-79.

[17] "Fast Silicon Challenges Exotic Technologies for the Infobahn", Electronic Design, July 1994, Vol. 42, n° 15,pp 48-50.

[18] D. Bursky, "Fast DRAMs Can Be Swapped For SRAM Caches", Electronic Design, July 1993, pp. 55-70.

[19] "Special Report on high-speed DRAMs", IEEE Spectrum, October 1992, pp. 34-57.

[20] B. Prince, "Semiconductor Memories", Wiley & Sons, 1991.

[21] B. Prince, "Memory in the Fast Lane", IEEE Spectrum, February 1994, pp. 38-41.

[22] A. Mzoughi, C. Guittenit, D. Litaize, " An integrated Input/Output System for Multiprocessors", Fourth Workshop on Scalable Shared Memory Multiprocessor, ISCA'94 Workshops, Chicago, April 1994.

[23] Norme du bus SCSI

[24] IBM, "Serial Storage Architecture"

[25] "IEEE plans to unveil a low-cost high performance serial interface with speed and flexibility", Electronic Design, June 1994, Vol. 42, n° 12, pp 27.

[26] IEEE, "IEEE-P1596 Draft Document, Scalable Coherent Interface", Draft 2.0, 1992.

[27] Pascal Sainrat, Abdelaziz Mzoughi, Christine Rochange, Daniel Litaize "The Design of the M3S : a Muliported Shared-Memory Multiprocessor", Proceedings SuperComputing '92, pp. 326-335, Nov 1992

[28] D. Litaize, O. Hammami, P. Sainrat, R. Pulou "Les liaisons séries du multiprocesseur M3S, Justification, Problèmes techniques, Solutions", Actes du Deuxième symposium sur les Architectures Nouvelles de Machine, Toulouse 12-14 septembre 1990, pp. 243-265

[29] Shogo Matsui "Dynamic Refresh Method for Dynamic RAMs", Computer Architecture News Vol 20 N° 4 Septembre 1992

[30] Mary K. Vernon and Udi Manber, "Distributed Round-Robin And First Come First Serveand their application to Multiprocessors Bus Arbitration", 15th Int'l Symp Computer Architecture 1988, p. 269-277

[31] Peyman H. Dehkordi and Donald W. Bouldin "Design for Packageability : Early Consideration of Packaging from a VLSI Designer's Viewpoint", Computer Vol 26 N°4, April 1993, page 76-81

[32] "VS8021/VS8022 : 2,5 Gb/s SONET 8-bit Mux/Demux Chip Set", in 1992 Product Data Book, VITESSE Semiconductor Corporation 1992

[33] "MC88110, 2nd Generation RISC Microprocessor User's Manual", Motorola Inc. 1991

[34] John L. Hennessy and David A. Patterson "Computer Architecture : A Quantitative Approach", Morgan Kayfmann Publishers, 1990

ANNEXE 2
Structures de caches, Processeurs tolérant les latences mémoire élevées

Yvon JÉGOU
André SEZNEC

21 février 1996

IRISA, équipe CAPS

Dans cette annexe technique, nous décrivons les travaux entrepris au sein du projet CAPS de l'INRIA concernant les mécanismes matériels visant à tolérer les latences élevées d'accès à la mémoire sur les monoprocesseurs.

Cette annexe décrit sommairement les travaux sur les caches déjà publiés ([SKV92], [LS92], [DS93], [JT93], [Sez93c], [ScB93], [Sez93b], [Sez93a], [DGeAS95], [JT94], [Sez94], [DSW95], [Sez95], [BS95]) et décrit plus largement deux études en cours respectivement sur l'étude de la taille de la ligne de cache pour les processeurs avec exécution dans le désordre et les processeurs multiflots.

Table des matières

Participants au projet

André Seznec, DR INRIA
Yvon Jégou, CR INRIA
Sébastien Hily, thèse
Nathalie Drach, thèse
Maurice Havez, DEA
Erven Rohou, DEA
Fabien Lloansi, Stage fin d'études ingénieur

1 La mémoire

1.1 Traitement matériel de la latence

Le traitement matériel de la latence peut se faire à plusieurs niveau : par une amélioration du comportement de la hiérarchie mémoire, par le déclenchement non bloquant des transactions de latence élevée et par la mise en œuvre du multiflot.

L'amélioration du comportement d'une hiérarchie de mémoires se traduit directement par une diminution des appels aux opérations coûteuses en latence. L'étude des caches monoprocesseurs fait l'objet de la section 2

L'utilisation de requêtes non bloquantes permet de ne pas suspendre l'activé du processeur sur un défaut de cache. Il suffit alors d'ordonnancer les instructions de manière à déclencher les accès mémoire plus tôt pour limiter l'effet des latences. Certains constructeurs implémentent des instructions de chargement préventif des antémémoires (*prefetch*). Non bloquantes et placées dans la séquence d'instructions par le compilateur de manière à être exécutées plusieurs cycles avant les instructions d'accès réel, ces instructions diminuent la pénalité des défauts sur les caches. Mais le placement automatique de ces instructions reste délicat. Le déclenchement automatique de ces chargements préventifs est également possible. Il se base sur une analyse à l'exécution des adresses émises par une même instruction. Lorsqu'une régularité est repérée dans cette séquence (adresses en progression arithmétique), les adresses suivantes peuvent être préchargées spéculativement. Les résultats de simulation ([JT93]) montrent l'efficacité de ce système pour des codes numériques réguliers.

Cependant, aucune de ces techniques ne permet d'absorber les latences de plusieurs centaines de cycles qu'il faut traiter sur les architectures massivement parallèles. Compte tenu de sa capacité à exécuter plusieurs flots d'intructions simultanément, le *multiflot* semble une solution efficace pour absorber ces niveaux d'attente. Si l'exécution de l'un des flot doit être suspendue pendant l'attente d'une donnée, la présence des autres flots permet d'occuper les ressources du processeur. L'étude du multiflot fait l'objet de la section 3.

1.2 Traitement logiciel de la latence

Le traitement logiciel de la latence peut se faire à plusieurs niveaux. Au niveau le plus fin, ce traitement consiste, d'une part à augmenter la localité des accès aux données, d'autre part à ordonnancer les instructions de manière à éloigner les instructions de lecture mémoire des instructions consommant les données lues. La plupart des mécanismes matériels comme le préchargement des données ou les caches non bloquants ne peuvent être exploités efficacement qu'associés à des techniques de génération de code spécifiques.

À un niveau plus grossier et en exploitant le parallélisme d'une application il est possible de masquer une partie des communications en entrelaçant l'exécution de plusieurs flots d'instructions sur chaque processeurs. Pour être efficace, le commutation d'un flot à un autre doit être peu coûteuse, ce qui impose qu'elle ne fasse pas appel au système d'exploitation. La bibliothèque d'exécution *pthread*, [Mue93], permet une telle implémentation de manière portable.

Cependant, en l'absence d'une prise en charge matérielle de ces flots d'instructions, la prise en compte des événements non programmés comme les défauts de cache ou

de pages sur un processeur fait nécessairement appel au système d'exploitation du processeur. Ce coût du changement de contexte limite l'utilisation des processus légers à un grain grossier de parallélisme.

2 Études sur les caches monoprocesseurs

En raison de l'écart grandissant entre temps de cycle du processeur et temps d'accès à la mémoire principale, l'accès aux données est souvent le facteur limitatif des performances. L'ajout de mémoires plus rapides (donc plus chères) et de petites tailles, appelées caches, entre le processeur et la mémoire principale permet de donner l'illusion que le temps moyen d'accès aux données est relativement faible. Ce concept architectural est appelé hiérarchie mémoire [HP90]. Compte tenu de l'évolution technologique des circuits intégrés (augmentation du nombre de transistors sur une puce), il est devenu possible d'avoir sur une même puce l'unité de calcul du microprocesseur et des caches de petites tailles. Cette évolution se concrétise aussi par une augmentation de la vitesse des composants (25 % tous les ans).

Ainsi, une structure à plusieurs niveaux de stockage va permettre d'améliorer les performances des ordinateurs, mais nécessite une bonne gestion des caches et des mécanimes mis en jeu tout en maintenant des temps d'accès à ces caches relativement bas. Dans la suite de cette étude, nous présentons tout d'abord brièvement les directions de recherche visant à améliorer les performances des caches que nous avons explorées au cours des 2 dernières annés en liaison avec le projet ILIAD. Les résultats déjà publiés au niveau international ne sont pas développés.

Nous développons par contre le thème d'une étude initialisée dans le cadre du projet ILIAD et qui devrait aboutir prochainement.

2.1 Le problème des caches

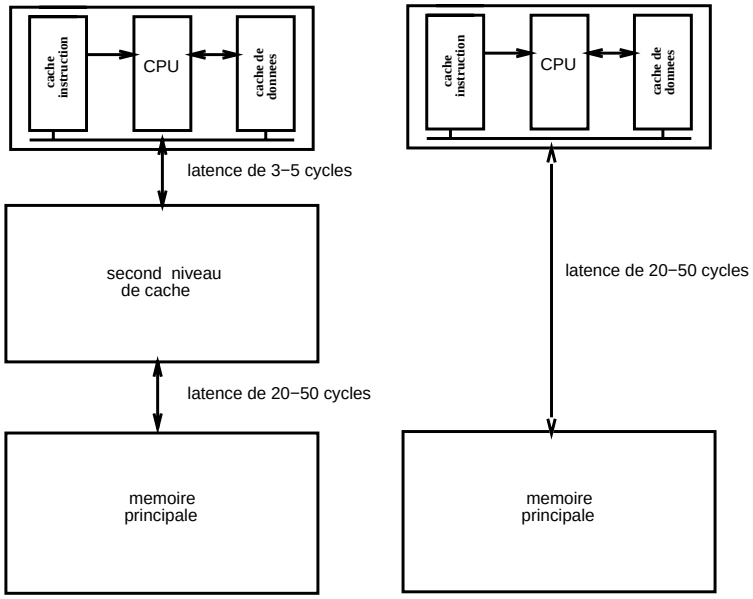
2.1.1 Hiérarchie mémoire

Le rôle d'une hiérarchie mémoire (figure 1) est de diminuer les temps moyens d'accès du processeur aux instructions et données.

Deux principes essentiels sont à l'origine de ce concept: le coût par bit d'une mémoire et les localités spatiale et temporelle exhibées par les programmes. Le coût par bit d'une mémoire croît lorsque son temps d'accès décroît. D'autre part le coût global d'une mémoire est proportionnel à sa taille. Par conséquent il est très coûteux de réaliser une mémoire de grande taille et très rapide. Aussi, plus une mémoire devra être rapide, plus sa taille sera réduite.

La hiérarchie mémoire est donc un agencement de plusieurs mémoires ayant des caractéristiques différentes: temps d'accès, capacités et coûts. Au niveau supérieur, proche du CPU, de petites mémoires très rapides sont utilisées. Elles sont destinées à stocker les éléments récemment référencés. Plus l'éloignement du CPU est grand, plus les mémoires sont grosses et lentes. A l'heure actuelle, le premier niveau de cache est intégré sur le même composant que le processeur.

FIG. 1 – *Hiérachies mémoire à deux niveaux de cache et à un seul niveau de cache on-chip*



2.1.2 Mémoire principale et caches *off-chip*

Lorsque le processeur émet une requête, elle est d'abord interceptée par le premier niveau de cache. Si cette donnée n'appartient pas à ce niveau de cache, la requête est transmise au niveau supérieur (le dernier niveau de cette hiérarchie étant la mémoire principale, indispensable car elle a une forte capacité de stockage à moindre coût). Ainsi le processeur doit attendre un certain délai fonction de la latence des différents niveaux mémoire parcourus, avant que la donnée ne soit utilisable. A noter que la pénalité est supérieure au simple temps de réponse de la mémoire. Par exemple, dans une hiérarchie mémoire à deux niveaux de cache, un cache *on-chip*[1] et un cache secondaire *off-chip* construit en mémoire SRAM dont le temps de réponse serait de 10 ns, la pénalité payée lors de l'accès à une donnée absente du cache *on-chip*, mais présente sur le cache secondaire sera de l'ordre de 40 à 60 ns. Ceci s'explique par les différents traitements effectués durant la requête: recherche de la donnée dans le cache *on-chip*, gel du pipeline, requête à la mémoire principale, envoi au processeur de cette donnée avec copie dans le cache *on-chip*, reprise des calculs au point d'arrêt. Et cette latence est d'autant plus pénalisante que la mémoire répond rapidement (caches secondaires *off-chip*).

Depuis 1985, les performances des processeurs augmentent de 50 à 100% par an alors que celles des DRAM (Dynamic Random Access Memory), componsants des mémoires principales actuelles, est de 7% [HP90]. Cette évolution qui conduit à des processeurs ayant des fréquences d'horloge élevées, induit des pénalités dues aux défauts de cache importantes.

1. Off-chip concerne l'extérieur de la puce du processeur et on-chip, la puce du processeur

2.2 Caches *on-chip*

L'espace de stockage disponible pour les caches *on-chip* est une contrainte très forte et le premier niveau de cache ne peut être que de faible capacité (4 à 32 kilo-octets). Le but est alors de gérer au mieux l'espace disponible afin de réduire au maximum les accès *off-chip* coûteux du fait de la différence entre temps de cycle du processeur et temps d'accès aux mémoires *off-chip*. Mais tout en diminuant le nombre d'accès *off-chip*, il faut essayer de ne pas pénaliser ou éventuellement de réduire le temps d'accès au cache *on-chip*. Les caractéristiques et les problèmes décrits dans ce paragraphe concernent également les autres niveaux de caches *off-chip*.

2.2.1 Accès *off-chip*

L'organisation des caches peut être définie par plusieurs caractéristiques qui influencent les performances et motivent les directions de recherche actuelles.

Répartition physique des caches entre données et instructions

La plupart des microprocesseurs utilisent deux caches distincts pour les données et les instructions: ceci permet de réaliser dans le même cycle un accès à une donnée et à une instruction. Cependant, cette partition statique du volume réservé aux caches sur le composant n'est pas optimale: pour certaines applications, un volume de cache plus important pour les données serait préférable; pour d'autres (ou à une autre phase de la même application) un volume plus important pour les instructions serait désirable. Même les caches unifiés (instructions et données stockées dans un même cache), utilisés dans la plupart des versions du SPARC et récemment dans le Power PC d'IBM/Motorola, et permettant une gestion dynamique de l'espace alloué aux instructions et aux données, présentent des inconvénients; il n'y a pas d'espace minimal préservé pour les instructions et les données et une instruction et une donnée ne peuvent pas être accédées au même cycle.

Compte tenu de ces contraintes, nous avons proposé une nouvelle organisation de caches, les caches *semi-unifiés* [DS93], qui est composée de deux caches distincts, mais où le cache de données peut être utilisé comme cache secondaire pour les instructions (et réciproquement). Cette organisation permet d'allouer dynamiquement l'espace de stockage entre instructions et données et d'accéder dans un même cycle à une instruction et une donnée.

Le choix de la répartition de l'espace de stockage entre instructions et données et les stratégies de placement et remplacement dans les caches sont étroitement liés; ces problèmes ont fait l'objet de nombreuses études [Hil88, MDH89, Smi82].

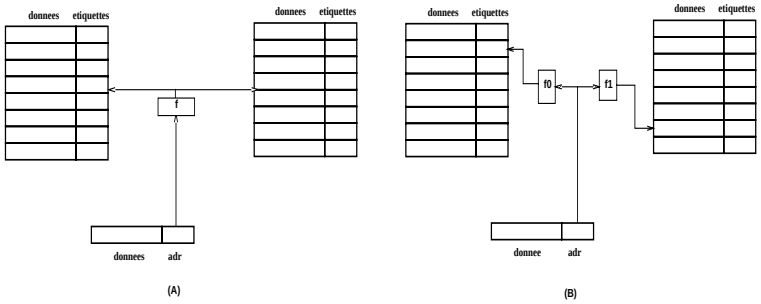
Politique de placement

Un cache est constitué de lignes (une ligne contient plusieurs données) et à chaque ligne est associée une étiquette ou *tag*, qui contient une partie d'une adresse, et des bits d'état. Une ligne, lors de sa copie, dans un cache peut être soit rangée n'importe où dans le cache (cache *fully-associative*), soit à une seule place (cache *direct-mapped*), soit dans un ensemble fixé de n lignes, mais à n'importe quelle place dans cet ensemble (cache n-*way set-associative*). L'associativité d'un cache définie le nombre de places où peut être

stockée une donnée. Une associativité élevée permet de réduire le nombre de défauts de cache, car elle diminue le nombre de conflits entre plusieurs données qui devraient être copiées dans la même ligne du cache. La complexité d'un cache (mécanismes et espace nécessaire à leurs mises en oeuvre) est liée à son associativité. Il est actuellement difficile d'implémenter de gros caches *fully-associative*.

Nous avons proposé une organisation de caches différente de celles utilisées dans les caches *set-associative*, les caches *skewed-associative* [Sez93c, ScB93, BS95]. Pour une associativité classique (voir figure 2.2.1.A), les lignes d'un même ensemble sont distribuées identiquement sur les différents bancs: une donnée peut être copiée dans la ligne 2 du banc 1 ou la ligne 2 du banc 2 (cas d'un cache *two-way set-associative*). Ainsi deux données qui sont en conflit sur une ligne le sont pour tous les bancs. Dans un cache *two-way skewed-associative* (voir figure 2.2.1.B), les fonctions de sélection de la ligne diffèrent pour chaque banc, et des défauts de cache sont ainsi évités. Mais comme nous le verrons par la suite, le choix de l'associativité d'un cache est également lié au temps d'accès à ce cache.

FIG. 2 – *Cache two-way set-associative (A) et cache two-way skewed-associative (B)*



Prefetch

Toujours dans l'optique de diminuer le nombre de défauts de cache *off-chip*, des techniques de préchargement ont été proposées. Ces techniques permettent d'éviter ces défauts de cache en préchargeant les données ou/et instructions à partir de la mémoire [Jou90, HP90]. Elles sont basées soit sur la localité spatiale des données, soit sur la localité temporelle (maintient dans un *buffer* des données récemment rejetées [Jou90]), soit sur une détermination spéculative de la prochaine adresse référencée [JT93].

Plus la place disponible sur la puce pour le stockage des données sera importante, plus le nombre de défauts de cache *off-chip* diminuera. Ainsi il faut essayer d'utiliser le maximum de cet espace pour le stockage des données et limiter la place destinée au contrôle.

Volume du contrôle

Dans un cache, le volume utilisé réellement pour le stockage des données, ne concerne pas tout l'espace de stockage disponible. Le reste de cet espace (étiquettes, bits de validité, bits de cohérence, ...) sert au contrôle et occupe une place importante sur la puce. La taille de l'espace d'adressage augmentant chaque année, le nombre de bits nécessaire pour coder une adresse (étiquette) croît également. Le rapport $\frac{volumeducontrole}{volumedesdonnees}$

est d'autant plus important que la taille de la ligne est petite. Si la taille des lignes de cache est augmentée, les problèmes de pollution augmentent également (beaucoup de données inutiles sont chargées et le taux de défauts de cache risque d'augmenter). De plus, pour un même nombre de défauts, le traffic entre la mémoire et le processeur augmente avec la taille des lignes (la ligne est l'unité de transfert). Enfin, dans un environnement multiprocesseur, l'utilisation de lignes longues accroît les problèmes de cohérences.

Pour réduire le volume des étiquettes sans augmenter la taille de la ligne, le concept de *sectored cache* est utilisé [HP90]. Un secteur consiste en un ensemble de lignes contiguës partageant une même étiquette. L'utilisation de secteurs peut permettre de résoudre les problèmes de lignes longues, mais induit un taux d'échec plus important (plusieurs lignes contiguës sont liées à une seule étiquette, le programme doit donc présenter une grande localité spatiale). Pour résoudre ce problème, nous proposons le *Decoupled sectored cache* [Sez94]. Dans ce cache, une ligne n'est plus liée statiquement à l'étiquette d'un secteur, mais un secteur possède plusieurs emplacements pour une étiquette et le lien entre étiquettes et lignes est décidé dynamiquement au moment du chargement. Ce procédé permet de maintenir un volume d'étiquettes faible tout en conservant un taux d'échecs également faible.

Choix de la longueur de ligne

Les caches exploitent deux types de localités dans les accès à la mémoire : la localité temporelle qui provient de la réutilisation de données et la localité spaciale qui provient de l'accès à des données d'adresses voisines. Une ligne de cache courte favorise l'exploitation de la localité temporelle (en conservant dans le cache plus de données accédées récemment) alors qu'une ligne longue favorise l'exploitation de la localité spaciale par des effets de préchargement. Nous avons montré dans [JT94] qu'il était possible d'exploiter simultanément ces deux types de localité par l'introduction de lignes virtuelles : un défaut de cache entraîne le rechargement d'une ligne virtuelle complète mais les données remplacées sont conservées dans un petit cache secondaire structuré en lignes plus courtes. La présence de ce deuxième cache réduit les interférences dues aux lignes longues.

2.2.2 Temps d'accès *on-chip*

Si réduire le nombre de défauts est important pour les caches *on-chip*, il ne faut pas négliger le second facteur critique de tout cache, le temps d'accès aux données. Ceci d'autant plus que, pour le premier niveau de cache, c'est ce temps d'accès qui détermine la fréquence d'horloge du microprocesseur. On essaie donc de le réduire au maximum. Le temps d'accès au cache est lié actuellement à son associativité. Pour un cache *direct-mapped* [Hil88], la lecture d'une donnée peut être décomposée en deux étapes :

1: lecture de la donnée et de l'étiquette associée,

2: comparaison de l'étiquette avec l'adresse de l'accès.

Pour un cache *n-way set-associative*, la lecture est décomposée en trois étapes :

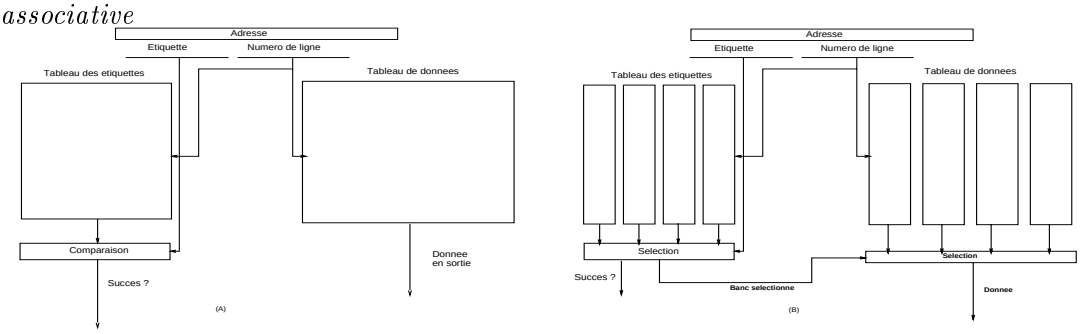
1: lecture d'une donnée pour chaque ligne de l'ensemble et des étiquettes associées,

2: comparaison des étiquettes avec l'adresse de l'accès,

3: sélection de la bonne donnée.

Cette troisième étape, qui est d'autant plus longue que le degré d'associativité est important, explique pourquoi le cache *direct-mapped* est légèrement plus rapide. La différence peut être encore plus significative car, dans un cache *direct-mapped*, la donnée peut être directement émise vers l'unité de calcul, avant la comparaison (voir figure 2.2.2); c'est ce que l'on appelle une exécution optimiste. Si la comparaison n'est pas valide, le cycle d'exécution de la donnée est simplement annulé et aucun cycle n'est perdu.
Une exécution optimiste est également possible pour des caches associatifs, mais il faut être capable de sélectionner rapidement une donnée à émettre. L'organisation *MRU* (Most Recently used) [SR90] permet, grâce à une étiquette associée à chaque ensemble, de choisir la donnée la plus récemment référencée dans l'ensemble.

FIG. 3 – *Complexité matérielle d'un cache direct-mapped et d'un cache four-way set-associative*



C'est en considérant les deux paramètres, défauts de cache et temps d'accès, que les caches *semi-unifiés* et *skewed* ont été proposés. Les caches *semi-unifiés direct-mapped* ont le même comportement que les caches unifiés *two-way set-associative*, tout en permettant de garder le temps d'accès très court propre aux caches *direct-mapped*. Et les caches *two-way skewed-associative* exhibent le même taux de succès que les caches *four-way set-associative*, mais avec la complexité matérielle d'un cache *two-way set-associative*.

Si le temps d'accès au cache reste trop important pour l'obtention d'une fréquence élevée, l'accès à ce cache peut être décomposé en plusieurs étages de pipeline. Certains microprocesseurs, comme le *DEC21064*, ont un accès au cache sur deux étages de pipeline. Faut-il continuer dans cette direction et avoir des pipelines de plus en plus longs (et réduire la fréquence d'horloge) ou au contraire limiter cette longueur (et limiter les pénalités classiques des pipelines)? C'est la question à laquelle, nous avons tenté de répondre dans l'étude [DSW95].

Un autre facteur important conditionnant le temps d'accès est le type d'adressage utilisé. Les programmes utilisent des adresses virtuelles alors que la mémoire principale est adressée physiquement. Il est donc nécessaire de traduire les adresses, ce qui est généralement fait à travers un TLB (*translation lookahead buffer*). Si les adresses sont traduites avant l'accès au cache (adressage physique), la perte de temps se traduit par une augmentation des temps d'accès. Pour maintenir un temps d'accès bas, la traduction devrait être effectuée après l'accès au cache (cache adressé virtuellement). Mais alors, il y a risque de perte de la cohérence mémoire (notemment du fait des alias, c'est-à-dire des adresses virtuelles correspondant à une même adresse physique), c'est pourquoi de nombreux microprocesseurs (*DECalpha*, *superSPARC*, ...) ont un adres-

sage physique.
Nous avons proposé une architecture de cache permettant d'utiliser un adressage virtuel mais avec tous les avantages de l'adressage physique, le *DASC* cache. L'organisation DASC (*Direct-mapped Access Set-associative Check*) [Sez95] repose sur un tableau de données *direct-mapped* et un tableau d'étiquettes associatif. Si le degré d'associativité correspond à la taille minimum d'une page, le cache peut être adressé virtuellement, et donc avoir un temps d'accès court, tout en maintenant automatiquement la cohérence des données. Le temps d'accès est court car l'accès aux données est *direct-mapped*, tandis que la gestion associative des étiquettes apporte un taux d'échecs bas.
De même, pour des caches *skewed-associative*, l'adressage virtuel est possible sans pénaliser le temps d'accès par rapport aux caches *set-associative*, à condition d'imposer une petite contrainte au système d'exploitation.

2.3 Etude en cours : vers de petites lignes pour les caches de premier niveau

2.3.1 Introduction

Beaucoup des processeurs récemment annoncés (HP8000, Intel P6, MIPS R10000, etc), implémentent une architecture superscalaire très complexe qui autorise l'exécution dans le désordre. Sur ces processeurs, l'exécution des instructions continue pendant que les *miss* du cache L1 sont traités à l'aide du cache L2 pipeliné. Comme plus d'un *miss* peut être traité à la fois, cela tend à favoriser l'utilisation de petites lignes de caches.

2.3.2 Pénalité d'un *miss* pour des caches synchrones

Sur la plupart des microprocesseurs sortis en 1992-1993 comme le DEC 21064, le MIPS R4400 ou l'Intel Pentium, un *miss* du cache d'instruction ou du cache de données avait pour résultat un blocage complet du pipeline du processeur.

Sur ces processeurs, le coût de la pénalité pour un miss du cache L1 était simple à modéliser. La pénalité était constituée de la latence L pour accéder au premier mot de la ligne manquante augmenté d'une pénalité supplémentaire r pour chaque mot en plus dans la ligne. Soit K le nombre de mots dans une ligne de cache, la formule 1 représente cette pénalité :

$$L + (K - 1) * r \quad cycles \tag{1}$$

L inclut le délai pour prendre en compte le *miss*, pour accéder au cache L2 externe avec la broche d'adresse, lire le cache L2, rapatrier la donnée dans le processeur, et reprendre l'exécution.

Par exemple, pour un système avec un Pentium 60 Mhz, le minimum pour L est de 5 cycles et le minimum pour r est de 1 cycle.
De nombreuses études indiquent que pour des tailles de cache de 8 KO à 32 KO, pour des latences réalistes L (5-30 cycles) et pour une mémoire (ou un cache L2) avec un temps de lecture r (1-3 cycles), la taille optimale des lignes de caches est 32 ou 64.

2.3.3 Taille des lignes de caches et exécution dans le désordre

Mais maintenant la technologie a changée. Sur les microprocesseurs annoncés récemment, une exécution dans le désordre est implémentée et/ou des caches L2 pipeliné sont utilisés. Nous expliquons ici, pourquoi la durée d'exécution sur ces processeurs est simplement moins affecté par chaque *miss* que dans un processeur synchrone traditionnel et que ceci tend à favoriser l'usage de petites lignes de caches.

FIG. 4 – *traitement des miss sur des données avec un cache L2 pipeliné*

cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
L1	ld A		ld B			Wr A	ld C	Wr B						
L2					ld A		ld B							

Use A Use B & C

(a) un bus d'un mot par ligne de cache

cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
L1	ld A		ld B			Wr A	Wr A	Wr A	Wr A	Wr B	Wr B	Wr B	Wr B	ld C
L2					ld A	ld A	ld A	ld A	ld B	ld B	ld B	ld B		

Use A Use B Use C

(b) quatre mots bus par ligne de cache

2.3.4 Caches L2 à haut débit

Des caches L2 spécifiques associés avec les derniers processeurs annoncés seront capables de délivrer une très grande bande passante : 16 octets par cycles pour le DEC 21164 ou 8 octets par cycles pour l'Intel P6 par exemple. Le temps d'accès au cache L2 est aussi très court : pour un *miss* sur le cache de données, la donnée chargée peut être utilisée 5 cycles après le début du *load* pour le DEC 21164 ou 6 cycles pour l'Intel P6 ou le MIPS R10000.

2.3.5 Cache de données

Un *miss* sur un cache de données ne doit pas bloquer tout le processeur. Pour un *load*, seules les instructions en dépendant postérieurement seront mises en attente pendant que les instructions n'en dépendant pas continuerons leur progression dans le

pipeline. Une technologie agressive de compilation doit pouvoir autoriser de prétraiter les instructions de chargement qui sont suspectées de provoquer un *miss*, par ce moyen la pénalité effective pour un *miss* peut être tout à fait limité pour la plupart des *miss*.

D'ailleurs, sur ces microprocesseurs, l'accès au cache L2 est pipeliné et plus d'un *miss* peut être traité pendant le temps résultant, avec une pénalité moyenne payée significativement plus faible que le temps dépensé pour résoudre le *miss*.

La figure 4 illustre pourquoi cela favorise l'usage de petites lignes de caches. Dans cet exemple, trois instructions de chargement sont considérés : *load A* et *load B* provoque des *miss* pour le cache L1, *load C* donne un *hit* pour le cache L2. Deux cas sont envisagés, (a) une ligne de cache de même largeur que celle du bus et (b) une ligne de cache de largeur quatre fois plus grande que celle du bus. Donnons quelques commentaire sur cet exemple :

disponibilité des données :

- en (a), la donnée B est disponible pour son utilisation au cycle 7
- en (b) la donnée B est disponible pour son utilisation au cycle 9

occupation du cache de données L1

- en (a) un *miss* occupe le cache de donnée L1 pendant deux cycles : premier accès résultant d'un miss et mise à jour de cache L1 avec la ligne
- en (b) le cache de donnée L1 est occupé pendant 5 cycles : premier accès et puis 4 cycles pour la mise à jour du cache L1

Quand le cache de donnée L1 est occupé par une mise à jour, il ne peut traiter d'autre requête, et alors l'exécution des instructions postérieures doit être différée. Par exemple, load C peut être réalisé au cycle 7 en (a), mais doit attendre le cycle 14 en (b).

occupation du cache L2

Le cache L2 est une ressource partagée qui doit résoudre les miss du cache d'instructions et les *miss* du cache de données, il doit aussi assurer un traitement cohérent des transactions dans le système complet. L'utilisation d'une ligne de cache de petite taille pour le cache L1 limite son occupation.

2.3.6 Cache d'instructions

Sur les nouveaux microprocesseurs, les instructions sont lues dans le cache avec une vitesse supérieure à celle de l'exécution. De sorte que lorsqu'il advient un *miss* pour une instruction, l'exécution continue pour les instructions déjà décodées durant la résolution du *miss* du cache. De manière que l'impact sur l'exécution d'un miss sur une instruction peut être très petit dans de nombreux cas.

L'accès aux instructions peut aussi tirer profit d'un cache L2 pipeliné. Les instructions ont généralement une très grande localité spatiale; le *prefetch* systématique de la

FIG. 5 – *traitement des miss sur des instructions avec un cache L2 pipeliné*

cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
L1	I2					W I2	W I3	W I4	W I5	W I6				
L2					I2	I3	I4	I5	I6					

Use I4

(a) un bus d'un mot par ligne de cache

cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
L1	I2					W I2	W I3	W I0	W I0	W I4	W I5	W I6		
L2					I2	I3	I0	I1	I4	I5	I6			

Use I4

(b) quatre mots bus par ligne de cache

prochaine ligne de cache après un *miss* sur une instruction est utilisé sur le DEC 21164. Cette technique permet de ne subir une pénalité effective que lors du premier miss.

On peut aussi noter que la pénalité effectivement payée pour un *miss* du cache d'instructions est généralement plus petite que le temps dépensé pour résoudre le *miss*. Sur tous les nouveaux microprocesseurs, les instructions sont séquencés en avance et mises dans un tampon pour attendre leur exécution. Pendant qu'un *miss* sur une instruction est traité, les instructions déjà lues peuvent être sélectionnées pour l'exécution. Dans certain cas, s'il y a assez d'instructions lues en avance, un *miss* sur une instruction pourrait n'avoir aucun impact sur la durée de l'exécution.

La figure 5 montre comment des lignes plus grande que la largeur du bus peuvent produire une durée d'exécution plus importante, une plus grande occupation du cache L1 et une plus grande occupation du cache L2 comme pour le cache de données. Dans cet exemple, un *miss* du cache d'instructions est assumé sur la ligne I2, I2 étant le troisième mot d'une ligne de quatre mots.

2.3.7 Avancement de l'étude

Nous avons formalisé les problèmes à étudier : l'impact de la taille de la ligne de cache de premier niveau sur les performances des processeurs superscalaires de haut degré.

Un simulateur complet de processeur superscalaire avec exécution dans le désordre est en cours de mise au point.

Des simulations extensives seront menées afin de montrer que :

1. La taille de la ligne du cache de premier niveau doit être petite (c-a-d d'une largeur égale à celle du bus externe)

2. L'exécution dans le désordre sur les processeurs superscalaires permet de tolérer une grande partie de la latence des caches de second niveau

Conclusion sur les études sur les caches

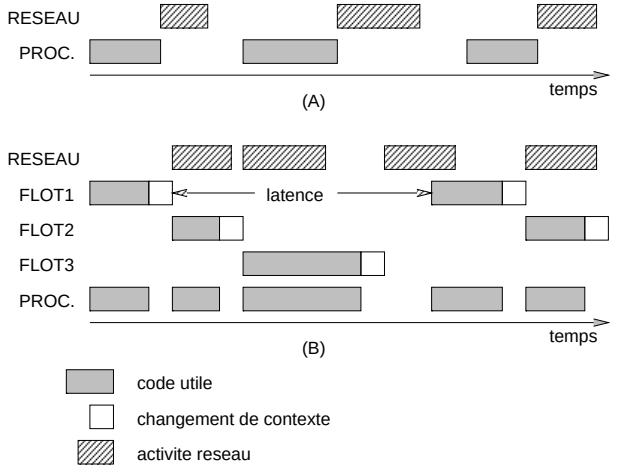
Les problèmes que nous avons étudiés ne constituent pas une liste exhaustive de tous les problèmes rencontrés dans les caches, mais en énumèrent les principaux. Les résultats que nous avons obtenus montrent qu'il est important de continuer les recherches dans ce domaine. De plus l'évolution de la technologie risque de motiver de nouvelles directions de recherche.

3 Le multiflot

3.1 Quelques généralités sur le *multiflot*

Le *multiflot* est l'une des méthodes permettant d'utiliser plus efficacement le processeur. Plûtot que de bloquer toute activité du processeur lorsqu'il effectue une opération distante, cas typique d'un processeur *monoflot* sans mécanisme de tolérance des latences (figure 6 (A)), un processeur *multiflot* avec un mécanisme de changement de contexte rapide peut commuter sur un nouveau flot alors que les autres flots sont bloqués en attente de leurs requêtes externes (cas (B) de la même figure).

FIG. 6 – *Activité processeur/réseau pour une machine monoflot (A) et multiflot (B)*



Une architecture *multiflot* est caractérisée par le nombre de flots qu'elle peut supporter. Déterminer un optimal pour ce nombre est assez difficile. Selon l'efficacité atteinte par un processeur avec un contexte unique, utiliser plusieurs contextes est ou n'est pas payant, et ceci est très dépendant de l'application. Ainsi dans des simulations effectuées par W-D Weber et A. Gupta [WG89], l'efficacité atteinte lors de l'exécution de LocusRoute (un routeur de standard cell dont les tâches sont à gros grain mais dont les données sont partagées avec un grain fin) avec un contexte unique est déjà proche de 90 pourcent et ajouter plus de contextes rapporte peu. Suivant la durée du changement de contexte, le *multiflot* peut même être pénalisant. Par contre, un programme tel que MP3D (un simulateur de particules en 3D, un exemple typique de code scientifique parallèle avec des boucles distribuées et des accès fréquents à des variables globales) peut attendre du *multiflot* des gains en performances substantiels (65 pourcent d'utilisation avec 4 contextes au lieu de 38 pourcent avec un seul).

Considérons une machine pouvant supporter un nombre quelconque n de flots. D'après la figure 6, on peut déduire une équation de l'efficacité du processeur [SBCvE90] [Aga92], c'est à dire la fraction de temps pendant laquelle le processeur exécute du code utile. La table 3.1 présente les paramètres principaux caractérisant l'exécution d'un programme sur une telle architecture. Un flot s'exécute pendant R cycles, prend C cycles pour changer de contexte, puis attend pendant L cycles la résolution de sa requête. Pour des raisons de simplicité et de lisibilité, on considère ces paramètres constants quel que soit le flot.

TAB. 1 –

R	nombre de cycles de code utile
C	nombre de cycles pour un changement de contexte
L	nombre de cycles de latence
n	nombre de flots supporté
E	degré d'utilisation efficace du processeur

Deux cas peuvent alors se présenter. Si le nombre de flots disponible n'est pas suffisant pour que la latence soit recouverte par une exécution de code (ce qui est le

cas dans notre exemple de la figure 6 (B)), soit pour $n-1 < \frac{L}{R+C}$, nous sommes dans une zone linéaire, l'efficacité augmente avec le nombre de flots :

$$E = \frac{n \times R}{R + C + L} \tag{2}$$

Lorsque le nombre de flots devient suffisant pour recouvrir la latence, ce qui correspond à $n-1 \geq \frac{L}{R+C}$, nous sommes alors en zone de saturation et l'utilisation devient limitée par le coût du changement de contexte :

$$E = \frac{R}{R + C} \tag{3}$$

Augmenter le nombre de flots ne permet alors plus d'augmenter l'efficacité, et peut même, comme nous venons de le voir, la diminuer.

En général, l'ordre de grandeur de la latence L est connu et c'est à partir de lui que l'on va définir R, C et le nombre de supports de flots (donc la borne maximale pour n). En fait R, le nombre de cycles de code utile, est déterminé essentiellement par la politique de changement de contexte. Ainsi, si l'on décide de changer de contexte à chaque instruction, R vaut 1 (grain fin). Si on choisit de changer de contexte uniquement lors d'un défaut de cache en lecture ou en écriture, L correspond au temps nécessaire pour satisfaire un défaut de cache et R vaut en moyenne l'inverse du taux de défaut ... La politique de changement de contexte est liée au niveau de parallélisme (grain) que l'on décide d'exploiter et son choix est délicat. En effet, choisir un *multiflot* à grain fin, cycle par cycle, permet d'éviter, dans le pipeline, les bulles dues aux dépendances ou à la latence mémoire. Par contre, du fait de l'entrelacement des flots, il présente l'inconvénient de dégrader les performances *simple-flot*, i.e. scalaires [WW93]. Ainsi, sur la Hep, qui empêche l'exécution consécutive d'instructions d'un même flot, un flot unique ne peut obtenir que un huitième des performances maximales. Hors, typiquement, même dans les applications numériques les plus parallélisables, le parallélisme disponible dans un programme peut varier de plusieurs ordres de magnitude sur une relativement courte période et les portions uniquement séquentielles sont souvent nombreuses, d'où l'importance de pouvoir supporter efficacement le *monoflot*. L'utilisation d'optimisation du pipeline par compilateur, une solution adoptée par exemple par la Tera ou April, ne fait que limiter les pertes en *monoflot*. Le *multiflot* à gros grain, moins coûteux, permet lui de conserver des performances normales lorqu'il n'y a plus que un flot à exécuter mais il a le désavantage d'aggraver la génération des bulles dans le pipeline. Non seulement il n'élimine pas celles induites par les ruptures classiques mais il en introduit de nouvelles à chaque changement de contexte (vidage du pipeline). Nous venons de voir que R dépend en fait d'un choix, guidé ou non, du concepteur. Un autre paramètre important, qui apparaît dans les deux équations précédentes est le coût du changement de contexte. On peut déjà remarquer qu'il détermine la latence minimale qui peut être tolérée de manière efficace. En effet, il n'y a aucun intérêt à avoir un temps de changement de contexte plus long que la latence, car le code exécuté pour commuter n'est pas utile pour le programme. Comme on peut le constater avec l'équation (2), le coût du changement de contexte est surtout le principal facteur limitant l'utilisation maximale du processeur. Pour être efficace, une architecture *multiflot* doit donc avoir un coût de changement de contexte très faible. Malheureusement, on cherche également de bonnes performances en *monoflot* et ces deux propriétés sont difficiles à atteindre simultanément. De bonnes performances en *monoflot* demandent que l'on puisse ordonnancer

plusieurs instructions d'un même flot dans le pipeline, ce qui complexifie beaucoup le matériel et nécessite de maximiser le nombre d'états résidents alors que le contraire est souhaitable pour avoir un changement de contexte rapide. En effet, supporter la présence d'instructions de flots différents dans le pipeline accroît la quantité d'état qui doit être sauvée et rechargée lors d'un changement de contexte et rend plus difficile un arrêt propre du pipeline. Ceci est d'autant plus vrai que le vidage du pipeline est une opération difficile à minimiser [Aga92].

On peut réécrire l'équation (2) en :

$$E = \frac{1}{1 + \frac{C}{R}} \tag{4}$$

ce qui met mieux en évidence que, si l'on dispose de suffisamment de flots, le degré d'utilisation est déterminé par le rapport entre le nombre de cycles du changement de contexte et le nombre de cycles passé en exécution utile. Si l'on augmente la taille de grain (si la fréquence des changements de contexte diminue), on peut donc disposer d'une certaine marge pour le coût du changement de contexte. C'est ce qui a été fait pour l'Alewife qui supporte un grain grossier et peut mettre en œuvre un changement de contexte logiciel en 14 cycles. A moins que le changement de contexte se fasse sans perte de cycle, il est donc essentiel de maximiser la longueur du code utile.

3.2 Architecture multiflot

Si les architectures superscalaires ou superpipelines permettent de tirer le meilleur parti du parallélisme à grain fin (quelques dizaines d'instructions), il n'en va pas de même des parallélismes à grain moyen (quelques centaines d'instructions, temps d'exécution de l'ordre de la dizaine de ms) ou à gros grain (applications entières, tâches de quelques dizaines de secondes). Pour exploiter ces deux derniers niveaux de parallélisme, les concepteurs de calculateurs ont imaginé des architectures contenant plusieurs microprocesseurs. Ces architectures peuvent être classées en deux grandes catégories :

- Les multiprocesseurs à mémoire partagée (*shared memory multiprocessors*); l'espace mémoire global est virtuellement partagé par les différents processeurs mais peut soit correspondre à une seule mémoire commune externe aux nœuds, soit être physiquement distribué dans les différents nœuds de la machine;
- Les multicomputers à échange de messages (*message passing multicomputers*); la mémoire n'est pas partagée et les communications se font exclusivement par échanges de messages entre les différents nœuds.

Ces machines parallèles, fonctionnant de manière asynchrone, offrent des perspectives fabuleuses mais présentent dans la pratique de nombreuses faiblesses. L'une des plus importantes provient des communications entre les processeurs et des accès à la mémoire. Sur de telles machines, les applications sont généralement divisées en tâches qui sont exécutées sur différents nœuds. Ces tâches doivent parfois se synchroniser ou échanger des données. Cela entraine, au niveau du microprocesseur, des latences extrêmement pénalisantes. Prenons le cas du multiprocesseur KSR1 [WBHA91]. Chaque nœud comprend un sous-cache et un cache local. Les nœuds sont connectés en anneau et la machine est un anneau d'anneaux. L'espace mémoire global est distribué sur les

différents nœuds, ce qui fait que le temps d'accès à une donnée varie suivant où est stockée cette donnée (table 2). Lorsque le microprocesseur veut lire une donnée, la latence peut varier entre 2 et 600 cycles. Puisque le processeur est un VLIW de degré 2, l'exécution potentielle de 4 à 1200 instructions est perdue. Pour obtenir des performances

TAB. 2 – *temps d'accès au cache sur la KSR1*

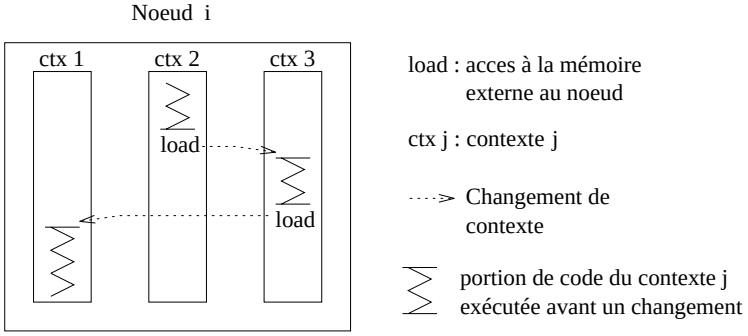
niveau mémoire	nombre de cycles
sous-cache	2
cache local (page existante)	23.4
cache local (nouvelle page)	49.2
cache distant SG:0 (page existante)	135-175
cache distant SG:0 (nouvelle page)	135-175
cache distant SG:1 (page existante)	470-600 (estimation)
cache distant SG:1 (nouvelle page)	470-600 (estimation)

maximales, il est donc très intéressant de faire se recouvrir les calculs et les communications. La technique de l'exécution *multiflot*, ou *multithreading*, a été développée dans ce sens [Hwa93]. Cette technique consiste à maintenir plusieurs flots d'instructions prêts à s'exécuter à tout instant. Lorsqu'un flot doit attendre la fin d'une communication ou d'une transaction avec la mémoire externe, les ressources de calcul sont allouées à un autre flot. Le multiflot peut être implanté à plusieurs niveaux : matériel pour exploiter un grain de parallélisme relativement fin ou logiciel pour un grain plus gros. Un microprocesseur *multiflot* est capable d'exécuter un ou plusieurs flots d'instructions (*thread*) indépendants simultanément. Lorsqu'une instruction d'un flot provoque par exemple un défaut de cache (la donnée à lire ne se trouve pas dans le premier niveau de cache), l'exécution de ce flot est suspendue, son état est sauvegardé pour permettre une reprise ultérieure et un autre flot est exécuté. Pour cela le microprocesseur gère plusieurs contextes simultanément sur la base du changement de contexte (*context-switching*, figure 7). Le contexte est constitué par les valeurs des registres du *CPU* (*Central Processing Unit*, unité centrale de traitement) et les variables d'état et représente lui-même l'état d'exécution d'un flot d'instructions. Clairement, une architecture *multiflot* fusionne l'exécution de flots d'instructions indépendants pour minimiser la fraction de temps passée à attendre. Le microprocesseur n'est alors bloqué que si tous les contextes sont suspendus. Il faut bien évidemment que le coût du changement de contexte soit très faible, ou soit bien moindre que celui de la latence mémoire. On distingue les architectures *multiflot* suivant leur politique de changement de contexte. Les quatre politiques les plus courantes sont :

1. changement sur défaut de cache;
2. changement à chaque accès au cache;
3. changement à chaque instruction;
4. changement à chaque bloc d'instruction.

Un changement sur défaut de cache est la politique la plus immédiate et n'entraine de changement de contexte que lorsque l'on est sûr que le microprocesseur va devoir

FIG. 7 – *Exemple de changement de contexte sur un nœud à 3 contextes*



attendre. Un changement à chaque instruction permet de minimiser les dépendances de données dans le pipeline: seules les instructions prêtes à s'exécuter sont séquencées à condition que l'ensemble des contextes d'exécution soient accessibles en permanence. Un changement à chaque bloc d'instructions permet d'améliorer le taux de succès des accès au cache grâce à une meilleure localité. L'un des premier multiprocesseur *multiflot* a été le Denelcor HEP [AAC+92] conçu en 1978. Cette machine comportait 16 processeurs, chaque processeur pouvant supporter 128 flots d'instructions simultanément. La machine TERA de la société Tera Computer descend directement de ce modèle et devait être disponible en 1993. C'est un multiprocesseur basé sur une architecture von Neumann classique [AAC+92]. Chaque processeur peut également exécuter jusqu'à 128 flots simultanément et le changement de contexte se fait à chaque cycle. Le *multiflot* s'applique naturellement au principe des architectures à *flot de données* (*dataflow*) dans lesquelles l'ordre d'exécution des instructions n'est plus directement dicté par le programme (*control flow*) mais est commandé par la disponibilité des données [Hwa93]. Deux machines hybrides mariant les concepts *flot de données* et von Neumann et intégrant le principe du *multiflot* existent actuellement, l'ETL/EM-4 [SKS+92] mise au point au Japon et la *T [NPA92] dont le prototype a été conjointement réalisé par le MIT et Motorola.

3.3 Multiflot et caches

La gestion de plusieurs flots sur un seul circuit induit un traffic énorme et la bande passante nécessaire pour un haut débit et un faible taux de contention n'est pas disponible du fait du nombre limité de pattes de connexions des circuits intégrés. L'utilisation de mémoires caches est donc indispensable, mais elle pose de nombreux problèmes qu'il est important d'avoir à l'esprit lors de la conception d'une architecture *multiflot*. Le but d'une mémoire cache est de profiter de la localité spatiale et temporelle des données pour réduire le traffic avec la mémoire [HP90]. Un accès n'est effectué vers la mémoire que lorsque le cache est pris en défaut, c'est à dire lorsqu'il ne contient pas déjà la donnée. Dans un processeur *monoflot* classique, on peut rencontrer trois types de défauts de cache :

– les défauts d'initialisation (*start up effects* ou *compulsory misses*), lorsqu'un bloc est chargé dans le cache pour la première fois (exécution initiale du programme),

– les défauts de capacité (*capacity misses*), lorsque la taille de cache n'est pas suffisante pour héberger tout l'espace de travail (*working set*),

– les défauts de conflit, lorsque des blocs d'un même processus se chassent mutuellement.

Lorsque ce processeur est placé dans un environnement multiprocesseur, à ces trois types de défauts de cache s'ajoutent les invalidations induites par le maintient de la cohérence de cache (*invalidation misses*). Enfin, dès que plusieurs programmes se partagent le cache, ce qui est le cas pour la *multiprogrammation* en général et le *multiflot* en particulier, un nouveau type de défaut apparaît, causé par les blocs de différents processus rivalisant pour résider dans le cache. Généralement, on appelle ce dernier, défaut extrinsèque, par opposition aux défauts de capacité et de conflit regroupés sous le terme de défauts intrinsèques. En fait, avoir plusieurs flots travaillant avec un même cache est susceptible de générer plusieurs types d'interférences pouvant avoir des effets très différents. D'un coté l'effet peut être positif car un flot peut charger des données utilisées par un autre flot (il agit alors comme une sorte de préchargement). D'un autre côté, négatif, les données chargées par un flot peuvent chasser celles nécessaires à un autre flot, ce qui augmente le taux d'échecs. À l'extrême, on peut assister à un phénomène désastreux de ping-pong, des blocs se chassant mutuellement de manière constante. Dans la pratique, des simulations [WG89] [SBCvE90] ont montré que les interférences négatives l'emportent sur les positives. Dans le cas d'un changement de contexte sur défaut de cache, un nombre plus élevé de défauts diminue l'exécution utile (R) et, du fait de l'augmentation du traffic, peut accroître la latence mémoire. C'est ce qui explique pourquoi la zone linéaire (1) n'est que pseudo-linéaire et pourquoi l'on ne triple pas l'efficacité en triplant le nombre de flots. Avec un cache, le début du régime de saturation est décalé et au dela de ce point, contrairement à ce qui se produit pour la courbe déterministe où l'on ne tient pas compte des caches, l'efficacité diminue quand le nombre de flots augmente. W.D. Weber et A. Gupta [WG89] ont évalué l'utilité d'avoir plusieurs contextes par processeur (2 ou 4) pour un multiprocesseur dont la cohérence de cache est assurée par un répertoire (*directory-based*). Chaque contexte contient un jeu de registres indépendants pour assurer une commutation rapide. Le changement de contexte se fait sur défaut de cache ou sur une écriture avec succès d'une donnée partagée en lecture. Grâce a ce mécanisme simple, la décision de commuter peut être prise en un cycle, ce qui permet de conserver un changement de contexte très court. Pour le programme MP3D, augmenter le nombre de contextes résulte en des délais sur le réseau nettement plus importants, du fait d'un traffic global plus élevé, et cela induit des latences qui fluctuent au cours de l'exécution (pas de modèle déterministe possible). Des travaux d'Agarwal [Aga92] montrent que, pour un modèle donné (latence de 10 cycles, changement de contexte en 4 cycles, cache de 64 koctets, ...), quand le nombre de flots résidants passe de un à deux, le taux d'utilisation du processeur passe de 0,4 à 0,7, et, du fait du nombre croissant d'interférences dans le cache et du coût du changement de contexte, quand le nombre de flots passe à trois, l'utilisation passe seulement à 0,8. Bien sur ces effets négatifs sont très dépendants des applications et donc difficiles à maîtriser. Simplement, la recherche d'une meilleure utilisation du processeur doit nécessairement passer par la recherche d'un équilibre entre le nombre de flots supporté et le taux de conflits générés sur le réseau par un taux d'échec plus élevé (sans négliger l'impact sur le coût du changement de contexte). Ceci d'autant plus

que l'augmentation du nombre de supports matériels de flots entraîne un coût matériel croissant et peut amener une diminution de la fréquence d'horloge.

Pour limiter les défauts d'invalidation et d'initialisation, une idée immédiate est d'héberger, sur un même noeud, des flots qui partagent des données. Une étude effectuée par R. Thekkath et S.J. Eggers [TE94] montre que, en fait, ça n'est pas efficace. Quel que soit l'algorithme de placement utilisé (pour des applications à gros grain et à grain moyen), le taux de défauts d'invalidation et d'initialisation reste à peu près constant. Au contraire, l'équilibre des charges est apparu comme le facteur critique affectant les performances. Cela vient du fait que la plupart des applications ont des accès séquentiels et uniformes et que le nombre d'instructions nécessitant un accès distant est très réduit par rapport au nombre d'instructions (ces simulations ont été faites sur des programmes tels que ceux de *Splash* ou de *Presto* avec un cache à correspondance directe pour les données et en supposant un temps d'accès d'un cycle aux instructions (pas de conflits avec les données)).

En assignant des caches indépendants pour les contextes, on élimine l'effet des interférences extrinsèques sur le taux de défauts de cache, [DMT95]. Si cela est très simple pour des caches d'instructions, cela devient par contre beaucoup plus compliqué à appliquer pour les données (notamment si l'on veut partager les ressources entre les différents flots). De plus on ne bénéficie plus du phénomène d'adaptation automatique relative de la taille de cache en fonction de la charge, phénomène que l'on trouve dans les caches partagés (partagés pour un même espace de travail, entre instructions et données, ou entre plusieurs espaces de travail différents).

Pour les défauts intrinsèques, ce sont notamment la taille du cache et son associativité qui sont importantes. Lorsque plusieurs flots partagent un cache, chaque flot voit un cache plus petit. Des travaux d'Agarwal [Aga92] indiquent que, même avec de petits caches, le *multiflot* peut être très intéressant pour un processeur pourvu que la bande passante du réseau soit suffisante. De gros caches (au moins 64 koctets) permettent d'atteindre une utilisation complète du processeur avec seulement 2 à 4 flots alors que de petits caches ne peuvent atteindre un haut degré d'utilisation et ce, quel que soit le nombre de flots, à moins que la bande passante du réseau soit augmentée et le coût du changement de contexte faible. Les travaux d'Agarwal rejoignent ainsi ceux de Weber et Gupta en affirmant que, quand la latence est petite (50 cycles), 2 à 4 flots suffisent à avoir une utilisation presque maximale. Sans augmenter le nombre de flots supporté, on peut essayer de diminuer la taille de grain des flots. Pour une application de taille fixe, augmenter le parallélisme permet de réduire la taille de l'espace de travail du flot. De plus, les espaces de travail de plusieurs flots tendent ainsi à moins interférer et la probabilité de partage est augmentée. Cependant, on retrouve alors l'un des inconvénients de n'avoir qu'un faible nombre de flots (courts), à savoir la possibilité d'apparition de famine et donc l'impossibilité de changer de contexte à volonté.

Que la machine mette en œuvre un cache ou non, un autre problème posé par le *multiflot* est le choix d'une politique de gestion des accès distants. Généralement les processeurs utilisent des mécanismes d'attente active (*busy waiting*). Lors d'un accès distant par un noeud N1, la donnée à lire ainsi que ses bits d'état sont rapatriés. Si les bits d'état indiquent que la donnée n'est pas valide, le noeud peut alors retenter la lecture immédiatement ou après un certain délai. Cependant, cela gâche de la bande passante réseau ainsi que des cycles du noeud N1 pour les tests et les réessais. L'attente active est donc souvent utilisée avec un mécanisme de compteur. C'est la solution

mise en œuvre par exemple sur Tera et Alewife en utilisant des trappes plûtot que des tests explicites [NPA92]. Que l'on change immédiatement de contexte lors d'un accès distant ou après un certain nombre d'essais, les messages doivent permettre une continuation [NPA92]. Si après avoir effectué un accès distant dans un premier flot, nous commuttons sur un second flot qui effectue également un accès distant, les réponses peuvent arriver dans le désordre. Ainsi chaque accès distant et sa réponse doivent porter un identifieur du flot initiateur. Dans *T, cet identifieur de flot s'appelle une *continuation*. Le nombre d'identificateurs de continuation supporté matériellement varie beaucoup d'une machine à une autre: 4 dans Alewife, 64 dans la HEP et 1024 dans la Tera.

3.4 Multiflot et superscalaire

Dans un pipeline classique, les aléas de pipeline (*pipeline hazards*) comme la lecture après écriture, les conflits sur l'accès aux registres ou au bus résultat empêchent une nouvelle instruction d'être séquencée à chaque cycle. Pour pallier à ce problème, Farens et Pleszkun [Far91] proposent d'entrelacer un second flot d'instructions dans le pipeline de manière à pouvoir lancer une instruction quand le premier flot ne le peut pas. Les premières instructions de deux flots indépendants sont décodées et la logique de séquencement détermine laquelle lancer, l'autre étant bloquée. Toutes les unités fonctionnelles sont partagées, la seule dépense venant de la duplication du fichier de registres (y compris le bus résultat, pour de meilleures performances, notamment lorsqu'il est partagé) et de la logique de décodage/lancement. Parmis les politiques de sélection examinées, permutation du flot exécuté tous les cycles, avec priorité ou sur blocage, la solution sur blocage est la meilleure. Dans le meilleur des cas, les simulations effectuées ont démontré une amélioration du débit par un facteur variant entre 1,39 et 1,69.

Lorsqu'un flot s'exécute, chaque instruction n'utilise qu'une partie limitée des ressources disponibles. Pour améliorer l'utilisation, on peut également envisager de lancer plusieurs instructions simultanément.

Avec une architecture *superscalaire*, ça n'est plus une unique instruction qui est exécutée en parallèle sur le processeur, mais plusieurs instructions. Cela permet d'alléger la tâche du compilateur et d'extraire dynamiquement le parallélisme. Malheureusement, l'un des points faibles du *superscalaire monoflot* classique vient de ce que son *ILP* (degré de parallélisme des instructions) est limité par la taille de la fenêtre d'exécution des instructions (alors que pour le *VLIW*, il n'est limité que par la taille du bloc de base). En *multiflot*, l'indépendance des instructions entre les flots permet d'étendre avantageusement la taille de la fenêtre d'exécution des instructions et de pallier ainsi à la faiblesse principale de l'extraction dynamique. Le *superscalaire* semble un moyen intéressant pour le partage des ressources entre les flots, cependant, la question du degré à implémenter est ouverte. Le principal obstacle à un haut degré est le coût matériel du contrôle qui s'ajoute à celui du *multiflot*. Hirata et al. ont utilisé le *superscalaire* pour améliorer l'utilisation des unités fonctionnelles dans leur modèle de processeur [HKN+92]. D'après des simulations portant sur l'exécution de deux et quatres flots en parallèle sur un processeur à 9 unités fonctionnelles, ils obtiennent une accélération d'un facteur, respectivement, 2,02 et 3,72 par rapport à un *RISC* conventionnel. Dans le modèle LCM (*Large Context Multithreaded*) proposé par [LGN92], les instructions de plusieurs flots sont groupées pour l'exécution. Une telle approche permet d'exploi-

ter l'*ILP* en même temps qu'un parallélisme à grain plus grossier, sans problème de fenêtre d'instructions. À chaque cycle, l'unité d'ordonnancement sélectionne jusqu'à n instructions disponibles pour l'exécution (avec n le nombre de pipelines d'exécution). L'approche est similaire à celle proposée par Hirata et al., mais l'architecture multi-anneaux LCM tire partie d'un tampon haute-vitesse avec une approche par couche de la synchronisation et des *activations* résidentes de grande taille.

Une troisième approche peut être utilisée pour la conception d'architectures à lancement d'instructions multiple, le *découplé*. Une architecture *découplée* tente d'exploiter la nature indépendante du flot de contrôle, des accès mémoires et des opérations sur les données. En rendant possible l'exécution de ces opérations dans le désordre, le découplage permet d'éviter de nombreux blocages du flot d'instructions du fait de dépendances. G. Tyson et al. ont marié le *découplé* au *multiflot* dans une architecture monolithique appelée MISC (*Multiple Instruction Stream Computer*) [TF93]. Quatres processeurs *découplés* élémentaires (PEs) travaillent de façon asynchrone pour exécuter en parallèle les flots d'un même programme. Les files, utilisées de manière intensive dans toute architecture découplée, et engendrant l'asynchronisme, permettent également l'échange de messages explicite entre PEs (approche *MIMD* sans les limitations dues aux latences inter-processeurs, ni les problèmes de débit). Les latences variables des diverses unités fonctionnelles (principalement la mémoire) sont un obstacle qui complique l'ordonnancement de code sur les architectures *VLIW* et *superscalaires*. Ici, l'asynchronisme des PEs compense la variabilité des latences sans affecter le taux d'exécution des instructions indépendantes. MISC apparaît ainsi comme une machine *MIMD* intégrée, découplée, destinée à supporter et exploiter de façon élégante l'ILP.

3.5 Quelques architectures intéressantes

L'idée de faire gérer plusieurs flots d'instructions par un même processeur n'est pas récente. L'une des premières machine *multiflot* a été le processeur à flot multiple, sans registres et à ressources partagées de Miller en 1974 [AAC+92]. À chaque étage du pipeline de traitement des instructions, des files d'attente contiennent les opérations en attente. À chaque cycle, une opération prête est choisie dans chaque file, exécutée, puis mémorisée dans la file suivante. Ainsi, chaque étage est utilisé au maximum.

Le calculateur **HEP** de la société Denelcor [Smi78], mis au point pour des recherches ballistiques de l'armée américaine, est le premier multiprocesseur *multiflot* commercialisé. Il s'agit d'une machine *MIMD* pouvant contenir jusqu'à 16 processeurs connectés par un réseau de commutateurs à larges bandes. L'architecture est à grain fin, une instruction d'un flot différent étant exécutée à chaque cycle. Les instructions actives dans le pipeline sont indépendantes et les interblocages sont donc rares. La synchronisation s'effectue à travers un bit *plein/vide* (*full/empty bit* [Hwa93]) pour chaque emplacement mémoire et chaque registre. Quand une condition de synchronisation n'est pas respectée (emplacement marqué *plein* pour une lecture, *vide* pour une écriture), une réexécution de l'instruction est tentée au tour suivant du flot. Malgré l'absence de caches et une lattence élevée pour l'accès à la mémoire principale, l'organisation de cette architecture permet d'atteindre de bonnes performances. Mais l'un des gros inconvénients de la HEP qu'elle ne peut allouer plus d'1/8 de son pipeline à un même flot. Pour utiliser totalement le pipeline, il est nécessaire d'avoir un grand nombre de flot, ce qui n'est pas toujours réalisable [WG89][AAC+92].

La machine **Tera** [AAC+92] est un calculateur scientifique multi-utilisateur à mémoire distribuée. Elle est extensible et peut héberger jusqu'à 256 processeurs reliés par un réseau de commutateurs pipelinés disposés en grille creuse à trois dimensions. Elle est l'héritière directe des architectures HEP et Horizon, mais est optimisée pour gérer un parallélisme hétérogène (grain fin, grain moyen et gros grain) et permet à plusieurs instructions d'un même flot de coexister dans le pipeline. Chaque flot possède son propre jeu de registres 64-bits (un registre contenant l'état et le PC, 32 registres généraux, 8 registres cibles de branchements). À chaque cycle d'horloge, un flot prêt à être exécuté est sélectionné pour lancer sa prochaine instruction. Dès que l'exécution d'une instruction est terminée, une nouvelle instruction du même flot (qui en dépendait) peut être sélectionnée.

Le compilateur de Tera est responsable de la détection et de l'ordonnancement du parallélisme à grain très fin (instructions de type LIW, indicateurs d'anticipation pour les accès mémoire). La présence d'indicateurs d'anticipation pour les accès mémoire permet de tolérer la latence mémoire même à l'intérieur d'un flot (jusqu'à huit opérations mémoire en attente). Ce système permet de réduire à 9 le nombre de flots nécessaire pour tolérer la latence de 70 cycles du pipeline mémoire. Le parallélisme à grain fin (par exemple au niveau des boucles ou des blocs de code) est détecté et exploité par le compilateur et le matériel. Tera fournie les instructions *reserve*, *create* et *quit* pour respectivement allouer, activer et désallouer un flot sur un processeur. La programmation parallèle explicite est encouragée via des variables et des déclarations de type *future*. Le parallélisme à gros grain (*tasks*, *scheds* et *teams*) est laissé au système d'exploitation qui exécute de manière concurente des tâches indépendantes.

APRIL [ALKK90] défini l'architecture d'un processeur destiné à tolérer les latences rencontrées dans les multiprocesseurs étendus. Pour cela il met en œuvre un mécanisme de changement de contexte rapide permettant de gérer du *multiflot* à gros grain. APRIL fait partie du projet **Alewife** du MIT, un réseau étendu de processeur sous forme de grille, avec une mémoire partagée distribuée et maintient de la cohérence de cache par répertoires distribués (*distributed-directories based coherence*). Chaque noeud de l'Alewife contient un microprocesseur Sparcle à 33 Mhz, un cache à correspondance directe unifié de 64 koctets, une mémoire principale partagée de 4 Moctets, un coprocesseur flottant, une unité de gestion de la mémoire et un commutateur de routage. Le processeur **Sparcle**, une implémentation RISC d'APRIL basée sur un processeur Sparc, a été réalisé avec LSI Logic et Sun Microsystems et est opérationnel depuis mars 1992. Le coût d'un changement de contexte se situant entre quatre et dix cycles, ce type d'architecture vise un grain moyen de parallélisme. Dès lors, il devient possible de confier une fonctionnalité telle que l'ordonnancement des flots au logiciel exécutif, ce qui permet à la machine de supporter un nombre illimité de flots virtuels dynamiques. Un même flot est exécuté jusqu'à ce qu'il y ait un accès distant ou un échec lors d'une tentative de synchronisation. Le contrôleur de cache détecte un tel évènement et une trappe est alors initiée. Par contre, seul un état d'attente (*wait*) est déclenché pour les transactions courtes comme une requête à la mémoire locale après un défaut de cache. April contient quatre jeux de registres généraux, quatre jeux de registres d'état (*PSR*) et de chaines de pointeurs de programmes (*PC-chain*, pointeur courant et futur pointeur). En maximisant les accès au cache et à la mémoire locale, la nécessité de changement de contexte se réduit à un tout les 50 ou 100 cycles, ce qui permet de tolérer des latences de l'ordre de 150 à 300 cycles pour les accès distants avec 4 contextes.

Les concepteurs d'APRIL ont montré qu'une implémentation basée sur des processeurs Sparc peut atteindre près de 80 pourcent d'utilisation du CPU avec seulement trois flots résidents par processeur dans une machine étendue à base de caches et avec une latence réseau moyenne de 55 cycles.

La ***T** (*starT*) [NPA92] est une machine hybride ayant des caractéristiques à la fois des architectures classiques (*von Neumann*) et des machines à *flot de données* (*data-flow*). *T est également un acronyme de "multi(*)-Threaded", la machine pouvant être vue comme le modèle logique d'un noeud pour une architecture massivement parallèle *multiflot*. Chaque noeud contient trois processeurs asynchrones séparés, dP, sP et RMem. Le processeur principal, ou processeur de données, dP, est optimisé pour l'exécution séquentielle des flots. Le coprocesseur de synchronisation, sP, gère les messages correspondants aux activités courtes (synchronisation, accès distants, ...) et empile les autres messages pour le processeur de données. Le processeur sP prend donc en charge tous les aspects *multiflot* et le processeur dP, qui se limite aux tâches de calcul intensif, peut être un *Risc* normal. Le coprocesseur RMem, quant à lui, est destiné à satisfaire les accès en lecture/écriture des autres noeuds. Il accède à la mémoire locale et traite les requêtes sans perturber le fonctionnement de dP. Chaque flot est exécuté jusqu'à achèvement. Suivant le principe même du *flot de données*, il ne reste aucune trâce dans le processeur de calcul d'un flot qui effectue un accès distant ou une synchronisation. Une requête distante contient une *continuation* pour redémarer le flot quand la réponse est disponible. Par contre, un flot est obligé de sauvegarder ses données en mémoire avant de déclencher une opération distante, ces données étant "relues" par la continuation associée à la requête.

Dans [HKN+92], Hirata et al. proposent une architecture de processeur *multiflot* visant à améliorer le débit sur une machine parallèle orientée vers le calcul numérique intensif. Ils distinguent le multiflot concurent habituel (un seul flot s'exécute à un instant donné) du multiflot parallèle qui autorise le séquencement d'instructions en provenance de plusieurs flot au cours du même cycle. Chaque processeur physique contient une unité de chargement des instructions, plusieurs unités de support de flot (constituées d'une file d'instruction et d'un décodeur), un ensemble d'unités fonctionnelles et un grand fichier de registres divisé en bancs. Chaque unité fonctionnelle possède ses propres stations de réservation et son ordonnanceur. L'association d'un support de flot (*thread slots*) et d'un pointeur de programme (PC) forme un *processeur logique*, les *processeurs logiques* se partageant l'unité de chargement des instructions et les unités fonctionnelles.

Des simulations ont montré qu'en exécutant deux ou quatre flots en parallèle avec neuf unités fonctionnelles, on pouvait atteindre une accélération de 2,02 et 3,72 par rapport à un monoprocesseur Risc conventionnel. Les instructions sont envoyées dans l'ordre aux unités fonctionnelles mais peuvent être exécutées dans le désordre.

Le **MISC** [TFP92] [TF93] est un processeur monolithique *multiflot* (*Multiple Instruction Stream Computer*) capable d'extraire le parallélisme au niveau instruction et proposé par ses concepteurs comme une alternative aux machines *monoflot* séquençant plusieurs instructions par cycles (machines *superscalaire* ou *VLIW*). Le processeur MISC est un descendant direct de l'architecture découplée PIPE. Il est composé de quatres processeurs élémentaires (PE), d'un cache de données et d'un ensemble de *chemins de données* permettant les communications internes. Chaque processeur élémentaire (PE) contient, outre des unités fonctionnelles arithmétiques classiques, un

fichier de 32 registres généraux, et surtout une file de sortie, une file d'entrée pour chaque PE du système (lui même inclu) et deux files mémoires. Chaque chemin de données du processeur correspond à une file de sortie d'un PE et il y a en une file d'entrée par chemin de données. Ces files servent à la mise en œuvre d'un système d'*échange de messages* sans conflits au niveau bas et permettent des communications internes de faibles latences. Ainsi, les PE travaillent de manière asynchrone pour exécuter un programme sous forme de flots parallèles. Le MISC est capable de supporter en parallèle quatre tâches complètement indépendantes mais, comme des mécanismes sont mis en place pour faciliter les échanges entre PE, il est préférable de partionner une unique tâche en plusieurs flots d'instructions coopérants. Le système de files permet également une approche découplée des opérations mémoire. Selon les auteurs, MISC est une bonne voie pour utiliser la densité croissante de transistors disponible en VLSI (ne nécessite pas qu'une horloge synchrone soit distribuée sur toute la puce). De plus, chaque PE est assez simple, ce qui réduit le temps de développement et autorise des fréquences élevées.

Parmi toutes les machines évoquées précédemment, seules la HEP, Alewife et l'ETL EM4 ont fait l'objet du développement d'un prototype ou d'une commercialisation. Un multiprocesseur Alewife de 128 noeuds est en construction (un prototype à 16 noeuds existe déjà). Enfin pour l'ETL EM4, un prototype à 80 processeurs est opérationnel depuis avril 1990. Pour la MASA, la TAM, *T, la machine d'Hirata et la MISC, seules des études papier existent. Un prototype de la *T devait originalement utiliser des processeurs 88110 de Motorola, mais maintenant l'utilisation de PowerPC est envisagée. Un prototype de la machine Tera contenant 16 noeuds est en cours de conception et devrait être terminé avant la fin de l'année ...

3.6 Perspectives du multiflot

Compte tenu de sa capacité à exécuter plusieurs flots d'intructions simultanément, le *multiflot* semble une solution idéale pour gérer les niveaux de parallélisme à partir d'une centaine d'instructions. De plus, sa présence semble nécessaire pour une mise en œuvre efficace du parallélisme logiciel fin. Il n'a pas encore fait l'objet de beaucoup de recherches et nombre de ses potentialités sont encore ignorées mais il semble également très intéressant pour absorber partiellement ou totalement la latence. Cependant une telle architecture amène de nombreuses questions :

- Quels degrés *multiflot* et *superscalaire* choisir ? Le coût matériel d'une architecture *multiflot-superscalaire* est très élevé. Il est donc important de trouver un équilibre entre le parallélisme offert et le parallélisme que l'on peut espérer réellement atteindre. Le but est de maintenir occupées le plus d'unités fonctionnelles possible pendant la plus grande partie du temps. Dans un tel choix, la politique de changement de contexte joue un rôle essentiel, tout comme le type de préchargement des instructions.

- Comment alimenter en données et en instructions une telle architecture ? Quel interfaçage à la mémoire proposer ? Le mariage du *multiflot* et du *superscalaire* nécessite un débit extrêment élevé que le bus et la mémoire doivent être capables de supporter.

– La gestion *multiflot* entraîne des ruptures fréquentes de la localité des instructions/données chargées dans les caches mémoires. L'obtention de performances élevées passe donc par une étude approfondie des intéractions avec la mémoire.

– Comment assurer la cohérence des caches dans le cas d'une mémoire partagée?

– Quelle gestion des interruptions?

3.7 Travaux en cours

Le simulateur *multiflot* SimMul a été construit à partir d'une base de simulation scalaire fournie par SPA. SPA est un ensemble d'outils écrit par Gordon Irlam pour analyser les performances de programmes binaires sur les stations de travail de la gamme Sun supportant le système d'exploitation SunOS 4. Ces outils comprennent notamment un programme (spy) permettant de tracer une commande et un analyseur de trace. SimMul reprend ces programmes et les étend de façon significative pour supporter :

– le traçage simultané de plusieurs applications,

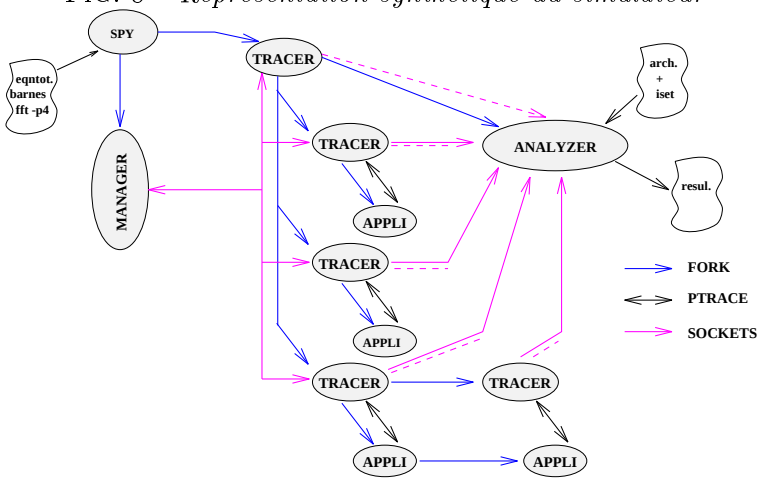
– le traçage d'applications parallèles,

– la simulation d'architectures *superscalaires*,

– la simulation d'architectures *multiflots*.

Le traceur (toujours appelé SPY) prend en entrée un fichier des commandes à tracer et un programme d'analyse vers lequel pipeliner la trace. À l'exécution (confère figure page 8), un processus de gestion (MANAGER) est créé ainsi qu'un premier processus de traçage (TRACER). Ce processus de traçage va d'abord lancer l'exécution de l'ana-

FIG. 8 – *Représentation synthétique du simulateur*

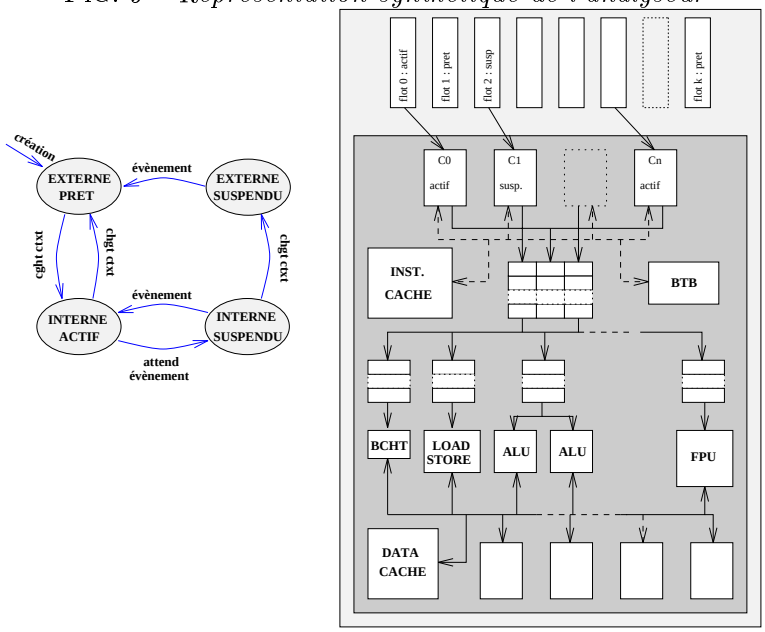


lyseur (ANALYZER) et créer une *socket* de communication vers lui, puis se dupliquer par des forks successifs, chaque fils prenant en charge l'exécution d'un programme correspondant à une commande. Cette exécution se fait en mode Ptrace, ce qui permet un contrôle total. Une *socket* pour l'émission de traces est créée vers l'analyseur et un

code assembleur (Agent, non symbolisé sur la figure) est chargé au début de l'application. L'agent exécute pas à pas le programme et en enregistre une trace. Lorsque son buffer est complet ou lorqu'il rencontre certaines instructions (*fork, wait, sync, ...*), l'agent émet une trappe vers le processus de traçage. Ce dernier, conjointement avec le MANAGER va gérer l'interruption et éventuellement envoyer la trace à l'analyseur. La rencontre d'une instruction *fork* provoque, outre la duplication de l'application, une duplication du processus de traçage associé et le chargement d'un agent dans l'application. Un message de création est également émis sur la *socket* de communication pour permettre à l'analyseur la création d'une nouvelle *socket* de trace. Pour chaque processus s'exécutant (application, fils d'une application, ...), une *socket* de trace est donc créée vers l'analyseur.

La gestion et la synchronisation des processus engendrés par l'exécution des applications sont assurées par les processus TRACER et par le processus MANAGER. Nous avons modifié le MANAGER pour qu'il soit notamment capable de traiter les macros PARMACS. Les macros PARMACS sont des macro-instructions développées aux laboratoires nationnaux d'Argonne permettant d'écrire des applications parallèles portables. Elles ont été utilisées pour coder les applications de SPLASH (Stanford Parallel Applications for SHared memory) qui nous servent de benchmark parallèle. Elles nous offrent les outils indispensables au contrôle du paralélisme, à savoir les verrous, les barrières, ... ou la déclaration de zones de mémoire partagée.

FIG. 9 – *Représentation synthétique de l'analyseur*



L'analyseur est un programme souple configurable à partir d'un fichier de description de l'architecture à simuler et d'un fichier de description du jeux d'instructions. Il permet de définir notamment le nombre et le type d'unités fonctionnelles, le nombre de registres, le degré superscalaire, la configuration de la hiérarchie mémoire, la prédiction de branchement et le nombre de supports de flots (confère figure 9). Il permet de

simuler l'exécution des différents flots d'instructions (fournis par les *socket*s de trace), d'évaluer les performances de l'architecture et d'extraire des informations détaillées sur certains points importants comme le nombre de défauts de cache, le nombre et le type des ruptures de contrôle, le nombre et le type des conflits de données, etc.

Grâce à la plateforme de simulation SimMul, nous disposons d'un outils à la fois souple et puissant. Nous allons ainsi pouvoir étudier très finement les potentialités des architectures *multiflots* en mettant en évidence l'impact de certaines caractéristiques comme le nombre de flots supporté, le nombre de registres, la hiérarchie mémoire, la politique de changement de contexte, etc. Une comparaison avec des architectures plus classiques telle que les architectures *superscalaires* devrait également nous permettre de valider l'intérêt du *multiflot* pour la conception des futurs microprocesseurs.

Références

[AAC+92] G.. Alverson, R. Alverson, D. Callahan, B. Koblenz, A. Porterfield, and B. Smith. Exploiting heterogeneous parallelism on a multithreaded multiprocessor. In ACM, editor, *Supercomputing*, pages 188–197, 1992.

[Aga92] A. Agarwal. Performance tradeoffs in multithreaded processors. *IEEE transactions on parallel and distributed systems*, 3(5):525–539, September 1992.

[ALKK90] A. Agarwal, B.H. Lim, D. Kranz, and J. Kubiatowicz. April: A processor architecture for multiprocessing. In *17th Int. Symp. on Computer Architecture*, June 1990.

[BS95] F. Bodin and A. Seznec. Skewed-associavity enhances performance predictability. In *Proceedings of the 22th International Symposium on Computer Architecture (IEEE-ACM)*, Santa-Margharita, June 1995.

[DGeAS93] Nathalie Drach, Alain Gefflaut, and Philippe Joubert eand André Seznec. About cache associativity in low-cost shared memory multimicroprocessors. Rapport de recherche 760, IRISA, October 1993.

[DGeAS95] Nathalie Drach, Alain Gefflaut, and Philippe Joubert eand André Seznec. About cache associativity in low-cost shared memory multimicroprocessors. *À paraître dans Parallel Processing Letters*, 1995.

[DMT95] H. M. Levy D. M. Tullsen, S. J. Eggers. Simultaneous multithreading: Maximizing on-chip parallelism. In *Preceedings 22nd Annual International Symposium on Computer Architecture (ISCA)*, pages 392–403, Santa Margherita Ligure, Italy, June 1995.

[DS93] Nathalie Drach and André Seznec. Semi-unified caches. In *Proceedings of the International Conference on Parallel Processing*, St Charles, Illinois, August 1993.

[DSW95] N. Drach, A. Seznec, and D. Windheiser. Direct-mapped versus set-associative pipelined caches. In *Proceedings of PACT'95 (Parallel Architectures and Compiler Techniques)*, Chypre, June 1995.

[Far91] M. K. Farrens. Strategies for achieving improved processor throughput. In ACM, editor, *Inter. Symposium on Computer Architecture*, pages 362–369, May 1991.

[FTP94] M. Farrens, G. Tyson, and A.R. Pleszkun. A study of single-chip processor/cache organizations for large numbers of transistors. In IEEE, editor, *21th Int. Symp. on Computer Architecture*, pages 338–347, April 1994.

[Gwe94] L. Gwennap. Microprocessors head toward mp on a chip. *Microprocessor Report*, 8(6):18–21, May 1994.

[Hil88] M. D. Hill. A case for direct-mapped caches. *IEEE computer*, December 1988.

[HKN+92] H. Hirata, K. Kimura, S. Nagamine, Y. Mochizuki, A. Nishimura, Y. Nakase, and T. Nishizawa. An elementary processor architecture with simultaneous instruction issuing from multiple threads. In ACM, editor, *19th Int. Symp. Computer Arch.*, pages 136–145, May 1992.

[HKT93] Y. Hidaka, H. Koike, and H. Tanaka. Multiple threads in cyclic register windows. In *20th Int. Symp. on Computer Architecture*, May 1993.

[HP90] J. Hennessy and D. Patterson. *Computer architecture : a quantitative approach.* Morgan Kaufmann publishers, 1990.

[Hwa93] K. Hwang. *Advanced computer architecture : parallelism, scalability, programmability.* McGraw-Hill, 1993.

[JÝ3] Yvon Jégou. Out-of-order execution of interruptible codes. Technical Report 2139, INRIA, November 1993.

[Jou90] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th International Symposium on Computer Architecture*, June 1990.

[JT93] Yvon Jégou and Olivier Temam. Speculative prefetching. In *Proceedings of International Conference on Supercomputing*, Tokyo, July 1993. ACM.

[JT94] Yvon Jégou and Olivier Temam. Using virtual lines to enhance locality exploitation. In *Proceedings of International Conference on Supercomputing*, Manchester, July 1994. ACM.

[LGN92] P. Lenir, R. Govindarajan, and S.S. Nemawarkar. Exploiting instruction-level parallelism: The multithreaded approach. In *MICRO 25*, pages 189–192, December 1992.

[LS92] P. Laporte and A. Seznec. Une étude comparative des microprocesseurs mips r3000, sparc version 7 et ibm power. Publication interne, IRISA, February 1992.

[MDH89] A. J. Smith M. D. Hill. Evaluating associativity in cpu caches. *IEEE transactions*, December 1989.

[Mue93] F. Mueller. A library implementation of posix threads under unix. In *Proceedings of 1993 USENIX Winter Conference*, San Diego, 1993.

[ND91] P. Nuth and W. Dally. A mechanism for efficient context switching. In *IEEE conf. on Computer Design*, October 1991.

[NO94] B.A. Nayfeh and K. Olukotun. Exploring the design space for shared-cache multiprocessor. In IEEE, editor, *21th Int. Symp. on Computer Architecture*, pages 166–175, April 1994.

[NPA92] R.S. Nikhil, G.M. Papadopoulos, and Arvind. *t : A multithreaded massively parallel architecture. In ACM, editor, *19th Int. Symp. Computer Arch.*, pages 156–167, May 1992.

[PC90] G.M. Papadopoulos and D.E. Culler. Monsoon: an explicit token-store architecture. *17th Inter. Symp. on Computer Architecture*, pages 82–91, 1990.

[SBCvE90] R.H. Saavedra-Barrera, D.E. Culler, and T. von Eicken. Analysis of multithreaded architectures for parallel computing. In *2nd Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 169–178, July 1990.

[ScB93] André Seznec and François Bodin. Skewed-associative caches. In *Proceedings of PARLE'93*, Munich, June 1993.

[Sez93a] A. Seznec. Interleaved sectored caches : reconciling low tag volume and low miss ratio. Rapport de recherche 759, IRISA, October 1993.

[Sez93b] André Seznec. About set and skewed associativity on second level caches. In IEEE, editor, *Proceedings of the International Conference on Computer Design (IEEE)*, Boston, October 1993.

[Sez93c] André Seznec. A case for two-way skewed-associative cache. In ACM, editor, *Proceedings of the 20th International Symposium on Computer Architecture(IEEE-ACM)*, San Diego, May 1993.

[Sez94] André Seznec. Decoupled sectored caches: reconciling low tag volume and low miss ratio. In *Proceedings of the 21st International Conference on Computer Architecture (ACM/IEEE)*, April 1994.

[Sez95] A. Seznec. Dasc cache. In *Proceedings of the 1st IEEE High Performance Computer Architecture Symposium*, 1995.

[SKS+92] M. Sato, Y. Kodama, S. Sakai, Y. Yamaguchi, and Y. Koumura. Thread-based programming for the em-4 hybrid dataflow machine. In ACM, editor, *19th Int. Symp. Computer Arch.*, pages 146–155, May 1992.

[SKV92] A. Seznec, A-M Kermarrec, and T. Vauleon. étude comparée des architectures des microprocesseurs mips r4000, dec 21064 et t.i. supersparc. Publication interne 692, IRISA, December 1992.

[Smi78] B. J. Smith. A pipelined, shared resource mimd computer. In *Int. Conf. on Parallel Processing*, 1978.

[Smi82] A. J. Smith. Cache memories. *ACM Computing Surveus*, September 1982.

[SR90] K. So and R. Rechtschaffen. Cache operation by mru change. *IEEE Transactions on Computers*, June 1990.

[TE94] R. Thekkath and S.J. Eggers. Impact of sharing-based thread placement on multithreaded architectures. In *21th Intern. Symp. on Computer Architecture*, pages 176–186, April 1994.

[TF93] G. Tyson and M. Farrens. Techniques for extracting instruction level parallelism on mimd architectures. In *MICRO 26*, 1993.

[TFP92] G. Tyson, M. Farrens, and A.R. Pleszkun. Misc: A multiple instruction stream computer. In *MICRO 25*, pages 193–196, December 1992.

[WBHA91] D. Windheiser, E.L. Boyd, E. Hao, and G. Abraham. Ksr1 multiprocessor: analysis of latency hiding techniques in a sparse solver. Technical report, ACAL Departement of electrical engineering and computer science. University of Michigan, Ann Arbor, MI 48109-2122, 1991.

[WG89] W.D. Weber and A. Gupta. Exploring the benefits of multiple hardware contexts in a multiprocessor architecture: Preliminary results. In *16th Int. Symp. on Computer Architecture*, 1989.

[WW93] C.A. Waldspurger and W. E. Weihl. Register relocation: Flexible contexts for multithreading. In *20th Int. Symp. on Computer Architecture*, May 1993.

ANNEXE 3
Étude des liaisons point à point à haut débit électroniques et optiques

Patrick Garda, Eric Belhaire
Institut d'Électronique Fondamentale

Philippe Lalanne, Pierre Chavel
Institut d'Optique Théorique et Appliquée

26 février 1996

Les liaisons point-à-point à haut débit sont actuellement l'objet de nombreux développements et de recherches actives. Elles diffèrent par leurs origines et leurs caractéristiques. Ce document fait le point des technologies actuellement utilisées et des recherches et développement en cours, et présente une comparaison des technologies optiques et électroniques pour la réalisation d'interconnexions au sein d'une machine parallèle.

Table des matières

1 Introduction

Les transmissions de données par liens optiques sont utilisées depuis de nombreuses années. Les premières transmissions mises en places commercialement l'ont été pour des distances très longues (liaisons transatlantiques et transpacifiques). Dans ces transmissions, on cherchait à utiliser les fibres optiques plutôt que les guides d'ondes à conducteurs (tels que les câbles coaxiaux) du fait de leurs pertes plus faibles. Après avoir avoir été utilisées pour des transmissions à des distances de plus en plus courtes (réseaux inter-urbains, réseaux urbains), les liens optiques sont, de nos jours, utilisés commercialement dans les réseaux locaux informatiques (FDDI, SONET par exemple). La question que nous nous sommes posée est la suivante: *Les progrés technologiques des composants optoélectroniques permettent-ils d'envisager l'utilisation des liens optiques dans les ordinateurs parallèles?*

2 État de l'art des liaisons à haut débit

2.1 Liaisons pour les télécommunications

Plusieurs normes de liaisons point-à-point à haut débit ont été définies pour les besoins des réseaux de télécommunications à longue distance (WAN), puis envisagées pour les réseaux locaux (LAN).

Ainsi SDH/SONET/STM sont les couches physiques de ATM en particulier. La base en est la transmission à 51,84 Mb/s, appelée STS-1, et dont tous les débits sont des multiples. STS-3c est à 155.5 Mb/s, STS-12 à 622 Mb/s et STS-48 à 2.4 Gb/s. STS-3c est le débit minimum de transmission pour ATM.

L'offre industrielle autour de ATM est abondante, et inclut des commutateurs, des adaptateurs avec des réseaux locaux et des cartes d'interface pour des ordinateurs personnels à des débits de 155 et 622 Mb/s, sur cuivre ou fibre optique.

2.2 Liaisons pour les entrées-sorties

Deux normes de liaisons à haut débit ont été définies pour les transferts de données entre des ordinateurs à haute performance et des périphériques d'entrées-sorties par le groupe de normalisation X3T11 du comité ANSI: HIPPI et Fibre Channel.

2.2.1 HIPPI

La norme HIPPI (High Performance Parallel Interface) a été définie initialement pour l'interconnexion de supercalculateurs, de machines parallèles et d'ordinateurs centraux, entre eux ou avec des périphériques de stockage. Elle permet aussi la réalisation de réseaux pour des centres de calculs, de passerelles entre réseaux ou de grappes de stations de travail.

Le temps d'établissement typique d'une connexion HIPPI est de 1 μs, et la latence de traversée d'un noeud est en moyenne de 160 ns.

HIPPI-PH spécifie la couche physique du modèle ISO pour une liaison point-à-point à 800 Mb/s. La liaison HIPPI est monodirectionnelle, et un échange bidirectionnel entre deux systèmes nécessite deux liaisons HIPPI.

Le support physique standard de la liaison est un câble blindé parallèle composé de 50 paires torsadées (conducteurs en cuivre), qui véhiculent les signaux de contrôle et de données d'un bus de 32 bits à la fréquence de 25 MHz. La portée maximale d'un câble est de 25 m.

La norme Serial HIPPI, en cours d'élaboration, spécifie la sérialisation de HIPPI sur d'autres supports physiques pour les transmissions au delà de cette portée :

- câbles coaxiaux jusqu'à 300 m ;
- fibres optiques multi-modes jusqu'à 1 km ;
- fibres optiques mono-modes jusqu'à 10 km.

Le transport de HIPPI sur des réseaux ATM et SONET a été normalisé et expérimenté pour les transmissions à longue distance (783 Mb/s à 2000 km sur SONET par exemple).

Des normes complémentaires spécifient un protocole de transfert (HIPPI-FP), un protocole de lecture et d'écriture distants en mémoire (HIPPI-MI), un protocole de contrôle de commutateurs (HIPPI-SC), et le plongement de IP et de IPI-3 (HIPPI-IPI, norme d'interface avec des disques durs et des lecteurs de bandes magnétiques) sur HIPPI. Il existe aussi une version 1600 Mb/sde HIPPI, moins développée que la version 800 Mb/s. Elle s'appuie sur un bus de 64 bits, transporté par deux câbles.

HIPPI est exploitée depuis 1988. L'offre industrielle autour de HIPPI inclut des canaux pour supercalculateurs et ordinateurs centraux, des commutateurs, des routeurs, des interfaces pour des périphériques de stockage et des cartes de visualisation, des adaptateurs pourstations de travail et PC (bus PCI). De nombreuses expériences d'inter-opérabilité ont été effectuées.

2.2.2 Fibre Channel

La norme Fibre Channel a été définie pour l'interconnexion d'ordinateurs entre eux ou avec des périphériques de stockage et de visualisation. Elle couvre les besoins de transferts rapides de grandes quantités de données. Elle prévoit 4 débits de transmission, 3 types de supports physiques, 4 types de transmetteurs, 3 catégories de distance, 3 classes de services, et 3 types de commutateurs.

FC-0 spécifie la couche physique du modèle ISO pour Fiber Channel. La table ci-dessous résume les différents supports physiques, avec les débits et distances maximales de transfert correspondants.

Débit (Mb/s)	Débit (Mo/s)	Distance maximale de transfert (m)				
		Paire torsadée	Mini coaxial	Coaxial	Fibre multimode	Fibre monomode
132,81	12,5	100	40	100	1000	-
265,62	25	50	30	75	1500	2000 ou 10000
531,25	50	-	20	50	1000	2000 ou 10000
1062,5	100	-	10	25	500	2000 ou 10000

D'autres normes spécifient un protocole de transfert (FC-1, FC-2 et FC-3), les commutateurs (FC-FG, FC-SW et FC-GS), les boucles arbitrées (FC-AL), et le plongement

de SCSI (FCP), de IPI-3 (FC-I3), de HIPPI (FC-FP), de ATM (FC-ATM) et de IP (FC-LE). Il existe aussi un plongement de Fibre Channel sur HIPPI.

Fibre Channel est une norme récente, adoptée par le comité ANSI à partir de 1994. Les industriels impliqués dans FibreChannel comptent en particulier HP, IBM, Seagate, SUN. L'offre industrielle autour de Fibre Channel inclut des circuits d'interface, des périphériques de stockage et des commutateurs. Par exemple, Fibre Channel est utilisée comme interface entre d'une part les stations de travail ou les serveurs multiprocesseurs SUN et d'autre part les serveurs de stockage RAID à base de disques SCSI SPARCstorage Array (depuis septembre 1995). L'interconnexion est réalisée par une fibre optique multimode à 265,62 Mbaud, à des distances maximales de 2 km. SUN a développé un circuit d'interface entre SCSI et les modules optiques. Fibre Channel est aussi utilisée comme interface entre les hypernoeuds et avec les périphériques de stockage des serveurs de calcul parallèle Enterprise de HP (depuis septembre 1995). Un serveur est composé de 2 à 32 hypernoeuds, chacun étant lui-même un serveur multiprocesseur comprenant de 2 à 12 processeurs PA-7200, soit au maximum 384 processeurs en tout. HP a développé le circuit Tachyon pour les interfaces avec Fibre Channel à 265,62, 531,25 et 1062,5 Mb/s (depuis juillet 1995).

2.3 SCI

2.3.1 La norme

La norme ANSI/IEEE SCI (Scalable Coherent Interface) définit une liaison à très haut débit qui peut être utilisée comme l'un des bus d'un ordinateur monoprocesseur, comme la base du réseau d'interconnexion d'un multiprocesseur ou comme un réseau local. Il s'agit d'une liaison point-à-point monodirectionnelle, dont deux copies de sens opposé sont nécessaires entre deux noeuds. Les topologies d'anneau ou de réseaux basés sur des commutateurs sont prévues par la norme. La norme définit une liaison qui peut servir dans une large gamme d'architectures (en terme de coût et de performances) et de supports physiques. Elle vise la simplicité de réalisation, la réduction de la latence et l'augmentation du débit. La latence des transmissions et la cohérence des données mémorisées dans des caches sont explicitement prises en compte dans la conception de la liaison et des protocoles. La cohérence des caches est basée sur des répertoires.

Un module SCI standard définit des signaux, un connecteur et une puissance permettant une transmission à 1 Go/s. Un connecteur de câble standard définit l'utilisation de ces signaux dans des applications pour lesquelles le module standard n'est pas approprié, sur des distances courtes de quelques mètres. Une interface standard avec des fibres optiques définit les transmissions série à des distances de quelques kilomètres à un débit de 1,25 Gb/s. D'autres débits et protocoles sont en cours de normalisation

La norme a été adopté par ANSI et IEEE en 1992. Plusieurs réalisations ont été effectuées, parmi lesquelles:

- le circuit AsGa "Data Pump" de Unisys et Vitesse, réalise une interface entre une liaison SCI à 1 Go/s, dont la resynchronisation est réalisée par une PLL, et un bus 64 bits à horloge externe ;
- la division AS/400 de IBM a réalisé un circuit BiCMOS d'interface avec SCI qui réalise simultanément des échanges entre une liaison SCI bidirectionnelle à un débit de 1 Go/s et des interfaces externes ;

- LSI Logic a réalisé le circuit CMOS NodeChip, qui fait l'interface entre deux liaisons SCI à 1 Gb/s et une liaison TTL ; il permet des débits de 125 Mo/s ;
- Dolphin a réalisé plusieurs circuits pour la mise en oeuvre SCI, dont le circuit CMOS LinkControler qui réalise la couche de transport de SCI ; il inclut deux liaisons à 1,6 Gb/s, et deux liaisons TTL à 400 Mo/s ; il peut fonctionner avec des paires torsadées ou des fibres optiques ; la consommation est de 3 W ;
- HP a réalisé un circuit qui permet la transmission série de SCI sur une fibre optique unique à 125 Mo/s ; il a été utilisé dans une liaison expérimentale réalisée au CERN par Dolphin. Des débits bidirectionnels de 200 Mo/s ont été mesurés entre deux stations SUN.

Par ailleurs, des cartes d'interface ont été réalisées. Ainsi Dolphin produit une interface entre SBus et SCI pour les grappes de SPARC. Elle peut fonctionner dans des réseaux en anneau ou avec un commutateur avec des liens à 1 Gb/s, sur paires torsadées jusqu'à 20 m. Elle supporte des SBus à 25 et 32 MHz. Elle procure des interfaces logicielles de type mémoire partagée et passage de messages. Le débit soutenu est de 10 Mo/s. La latence de traversée d'un noeud dans un anneau SCI est de 0,25 μs. La latence d'un ping-pong entre deux processus utilisateurs a été mesurée à 8μs.

Dolphin a aussi réalisé un commutateur pour SCI. Il comprend quatre ports à 1.6 Gb/s et supporte les câbles de paires torsadées. La bande passante interne est de 400 Mo/s, et la latence de port à port estde 1 μs. Il peut relier des noeuds isolés ou des anneaux. Un commutateur à hautes performances est étudié dans le cadre du projet européen TOPSCI (Participants: Dolphin, Thomson TCM, Thomson TMS, Sintef , DELAB, CERN RD24).

L'interface entre SCI et ATM est étudiée actuellement par Dolphin et par l'Université de Oslo. Dolphin étudie en particulier l'utilisation de ATM comme artère d'interconnexion entre des grappes reliées par SCI, en collaboration avec Norwegian Telecom et Alcatel.

Plusieurs ordinateurs parallèles récemment industrialisés exploitent SCI, chez HP / Convex et Sequent.

2.3.2 Convex

Les systèmes Exemplar SPP1000/XA et SPP1200/XA de HP/Convex sont des machines parallèles à mémoire partagée de type CC-NUMA. L'architecture comprend trois niveaux principaux. Les unités fonctionnelles comprennent deux processeurs HP PA 7200 à 200 MHz (PA7100 à 100 MHz dans le SPP1000), ayant chacun des caches externes séparés d'instruction et de données de 1 Mo, deux bancs mémoire(de 64 à 256 Mo chacun) et une interface. Le hypernoeuds comprennent jusqu'à quatre unités fonctionnelles et une unité d'entrées-sorties. Ils sont reliés par un crossbar à 5 ports. Un système comprendjusqu'à 16 hypernoeuds, reliés par 4 anneaux parallèles exploitant SCI. Au total un système peut comprendre jusqu'à 128 processeurs, soit une performance crête de 30 GLOPS. Une partie de chaque mémoire est partagée et peut être accédée par n'importe quel processeur, avec un support matériel pour les load et store, et un mécanisme de cohérence de caches basé sur des répertoires. Les mesures effectuées avec PVM donnent une durée de ping-pong de 65 μs pour une taille de message inférieure à 4 Ko.

2.3.3 Sequent

Sequent a introduit récemment une nouvelle architecture de serveurs multiprocesseurs, NUMA-Q, qui repose sur SCI. Les noeuds incluent quatre processeurs Intel Pentium Pro (P6) reliés par un bus à 500 Mo/s et disposant de deux bus d'entrées-sorties PCI à 133 Mo/s. Les noeuds sont reliés par un anneau de liaisons IQ, qui ont une interface avec le bus système des noeuds. Ces liaisons implémentent SCI, et permettent un modèle de mémoire partagée avec cohérence de cache (CC-NUMA) ou de passage de message. Les liaisons ont un débit de 1 Go/s, grâce au circuit AsGa "Data Pump" de Vitesse et à des ASIC de Sequent. Les accès mémoire distants ont une latence de 2 μs. Les interfaces d'entrées-sorties avec les disques et les bandes sont basées sur Fibre Channel. Un système peut contenir jusqu'à 252 processeurs, ce qui est très supérieur au nombre maximal de processeurs accessible sur la génération précédente de machines Sequent, les Symmetry 5000.

2.4 Liaisons série à haut débit

De nombreuses recherches actuelles visent à montrer la faisabilité de liaisons série à haut débit avec des technologies électroniques avancées mais standards. Elles peuvent être appliquées dans les domaines des télécommunications et des machines parallèles. En France des études sont menées par le LRI, le MASI et l'IRIT.

La technologie industrielle de Bull (Bullit de Marbot) montre que des cellules d'interface électronique avec des liaisons électriques sur cuivre ou sur câbles coaxiaux peuvent être conçues et réalisées dans des technologies CMOS 0,5 μm avec des fréquences de fonctionnement de 1 GHz et des débits de 0,8 Gb/s. La consommation d'une cellule est d'environ 1 W sous 5 V. La distance maximale de connexion est de quelques mètres. La faisabilité de liaisons à 3 Gb/s en BiCMOS a été démontrée.

Cette technologie est la base des normes IEEE 1355 et ISO/IEC 14575. Dans le cadre des projets ESPRIT OMI/HIC (Project Number 7252) puis OMI/Macrame (Project Number 8603), cette technologie est utilisée comme couche physique pour SCI, ATM et Fibre Channel, ainsi que comme support de bus rapides (VME et Futurebus+).

Différentes technologies de transmission sont prévues :

Technologie	Débit Bits/s	Débit max. bidirectionnel Mo/s	Support	Distance max. m
DS-SE-02	1.333M-200M	38	4 fils	1
TS-FO-02	250M	39.4	2 fibres multimode	300
DS-DE-02	1.333M-200M	38	8 fils	10
HS-SE-10	700M-1000M	160	2 câbles coaxiaux	8
HS-FO-10	700M-1000M	160	2 fibres multimode 62,5μm	100
			2 fibres multimode 50 μm	1000
			2 fibres monomode	3000

Un autre exemple de liaison série à haut débit a été réalisé par LSI Logic. Cette liaison a été exploitée dans le prototype S3.mp de SUN. La liaison est réalisée sous la forme du module SeriaLink, qui peut être intégré dans des ASIC conçus dans la technologie CMOS 0,5 μm de LSI logic. Le module comprend un sérialiseur 16-1 et un désérialiseur 1-16. Il supporte des transmissions jusqu'à 1.1 Gb/s en full duplex. Cette

liaison est la base d'un circuit de routage S-Connect développé par SUN. Ce routeur inclut deux ports parallèles 16 bit à 66 MHz avec un bus mémoire d'une part, un crossbar 6x6 et quatre liaisons série d'autre part. Les messages sont transmis avec des latences de 200 ns à 5 ms. Un noeud de S3.mp peut servir de carte additionnelle MBUS dans une SparcStation 10 ou 20 ou comme un noeud de calcul autonome. La distance entre noeuds peut atteindre 10 m directement, ou quelques centaines de mètres grâce à des liaisons par fibre. Ceci permet de réaliser un réseau de stations de travail par exemple. S3.mp réalise un modèle de mémoire partagée avec cohérence de cache basée sur des répertoires CC-NUMA, dans lequelles accès distants sont réalisés par envoi de messages.

2.5 État de l'art : liens optiques dans ordinateurs parallèles industriels

2.5.1 Liens à faible débit

Les liaisons optiques sont utilisées depuis plusieurs années dans des machines parallèles auxquelles doit être assuré un fonctionnement distribué. Il s'agit de situations dans lesquelles deux parties de la machine se trouvent à une distance trop grande pour permettre le fonctionnement de liaisons électroniques prévues en standard. Des exemples sont les machines parallèles MIMD à mémoire distribuée à base de transputers et de C40. Tous les grands fabricants de modules standards pour les machines (TRAM ou TIM40 respectivement) proposent des modules de communication basés sur des fibres optiques. Des exemples pour le C40 sont les modules deH unt Engineering et de LSI.

2.5.2 Optobus

Optobus est une liaison optique point-à-point full duplex industrialisée par Motorola. Elle comporte 10 canaux en parallèle dans chaque direction. Chaque canal est capable de transmettre des données à 150 Mb/sec, soit au total 1,5 Gb/s dans chaque direction. La liaison comprend deux transmetteurs identiques et un câble de 10 fibres optiques. La distance maximale de transmission est limitée à 30 m. Le débit et la vitesse maximale de transmission devraient augmenter rapidement. Chaque module est alimenté sous +5.0 V et il dissipe 1.7 W. Les modulateurs optiques sont des "Vertical Cavity Surface Emitting Lasers" (VCSELs) émettant à 850 nm. Le taux d'erreur est de 10^{-13}.

Deux expérimentations de SCI sur Optobus ont été réalisées, l'une au sein du groupe "Advanced Technology Group" de la société Apple, et l'autre au sein du Département de Physique de l'Université de Oslo, en collaboration avec la société Dolphin.

Le groupe ATG de Apple a réalisé une interface Optobus pour SCI utilisant deux modules Optobus de 10 bits chacun. Les 20 lignes sont utilisées pour transmettre les données SCI (16 bits), deux signaux d'horloge et un indicateur. Des débits de 140 Mo/s ont été obtenus.

Dolphin a construit une carte d'interface SCI avec transmission sur Optobus. Les 18 signaux de SCI (16 data + flag + clock) sont multiplexés sur les 10 canaux de Optobus. Le circuit de multiplexage 16-8/démultiplexage 8-16 est réalisé avec des composants discrets ECL 100k. En boucle fermée, les latences à travers l'interface Optobus ont été mesurées à 80 and 74ns pour des horloges à 62.5 et 100 MHz respectivement. La latence due à la fibre optique elle-même représente environ 45 ns. La durée totale d'une

séquence (requête, écho de la requête, réponse avec 16 octets de données, et écho de la réponse) est de 1.27 μs.

3 Bilan énergétique des liaisons séries

3.1 Introduction

Une liaison complète comprend (voir aussi la Fig. 1) :

- Une source (ou driver de ligne) qui peut, dans le cas d'une liaison optique, se décomposé en un amplificateur de signal et un transducteur électrons-photons.
- Une ligne, voire deux lignes dans le cas d'une liaison différentielle.
- un récepteur qui peut, dans le cas d'une liaison optique, se décomposer en un transducteur photons-électrons et un amplificateur de signal.

FIG. 1 – *composition d'une ligne série*

Chacune de ces parties est caractérisée par une consommation d'énergie ou des pertes propres et un bilan énergétique complet est donc nécessaire pour comparer une ligne par rapport à une autre. Dans cette section, nous présenterons un bilan des liaisons électroniques et des liaisons optique dans le but de pouvoir les comparer. Nous choisirons pour cela une modélisation simplifiée dont nous avons pu vérifier la validité sur plusieurs exemples de produits commercialisés ou publiés.

3.2 Les liaisons électroniques

3.2.1 Propriétés

La vitesse de propagation d'un signal électrique sur une ligne est limitée par la vitesse de la lumière. Un calcul rapide montre donc qu'un signal ne peut parcourir plus de 30 centimètre par nano-seconde. On voit donc que pour un transfert à 1 Giga-bit par seconde (soit 1 bit par nano-seconde), dès que la longueur de la liaison atteint quelques centimètres celle-ci ne peut plus être considérée comme une équipotentielle. Le problème doit alors être considéré au regard des équations de Maxwell et des équations de propagation des ondes. L'information se propage entre les connecteurs sous forme d'une onde électromagnétique.

Chaque liaison peut alors être considérée comme un guide d'onde électromagnétique caractérisé par :

- Sa structure (ligne coaxiale, ligne microstrip, ...).
- Son impédance caractéristique exprimée en Ohm ;
- Des pertes en lignes exprimées en m^{-1};

L'impédance de tout guide d'onde est limitée par l'impédance du vide qui est de 377 Ω. De plus, du fait de la limitation de la section du guide pour des raisons d'encombrement, l'impédance d'un guide d'onde est en pratique limitée à $R_0 = 100\,\Omega$.

Pour que la liaison soit fiable, il faut alors que les impédances de la source et du récepteur soient adaptées à celle du guide d'onde utilisé. Ainsi évite-t-on les réflexions des ondes aux extrémités du guide d'onde et donc que des informations transmisses à des instants différents ne viennent se superposer. L'adaptation d'impédance est donc une condition nécessaire pour qu'un transfert d'information à haut débit se fasse sans erreurs.

Pour les liaisons électroniques de signaux numériques, pour des raisons de compatibilité et d'immunité au bruit, l'amplitude des signaux est choisie compatible au niveau ECL, c'est à dire une amplitude d'environ 1 Volt entre le niveau 1 logique et le 0 logique.

3.2.2 Description et bilan d'énergie

Consommation dans la source ou driver de ligne Une ligne chargée sous une tension V_0 emmagasine une énergie :

$$E_{LL} = \frac{1}{2} C V_0^2$$

où C est la capacité de la ligne qui est chargée. Si la ligne d'impédance caractéristique Z_0 est chargée par une impulsion de tension de longueur τ, cette énergie peut s'exprimer sous la forme :

$$E_{LL} = \frac{\tau V_0^2}{2 Z_0} \tag{1}$$

Cependant, pour charger une capacité à partir d'une source de tension fixe V_{dd}, une énergie égale à E_{LL} doit être dissipée dans le driver de ligne. Aussi, pour transmettre une impulsion de longueur τ, on doit dissiper une énergie totale :

$$E_t = 2\, E_{LL} = \frac{\tau V_0^2}{Z_0}$$

Si, par exemple $V_0 = 1$ V, $\tau = 1$ ns, $Z_0 = 50\Omega$, on obtient une énergie totale $E_t = 20$ pJ par bit transmis.

Cette valeur doit être comparée à l'énergie nécessaire pour faire commuter une porte CMOS. On peut en effet, très bien avoir une telle porte en bout de liens. Si l'on suppose que la porte est alimentée sous 3,3 V et que sa capacité d'entrée est de l'ordre de 4 fF, il faut alors une énergie $E_c = \frac{1}{2} C V^2 \approx 20$ fJ. Soit donc une énergie 1000 fois inférieure.

Pour une transmission à 1 Gbit/s, en supposant que l'on a en moyenne autant de bits à 1 qu'à 0, on ne peut donc espérer avoir une consommation par ligne inférieure à $P = (1/2) \times (20\,\text{pJ}) \times (10^9\,\text{s}^{-1}) = 10$ mW.

En pratique, les différents circuits spécialisés dans des transmissions à 1 Gbit/s présentent des consommations très supérieures à celle calculée ci-dessus. En effet :

- les drivers de ligne ne peuvent être alimentés par une tension de 1 V et le courant chargeant les lignes est fourni par une alimentation V_{dd} de tension supérieure à 1 V et de l'ordre de 5 V. Ceci, conduit à une consommation $V_{dd}/1 \approx 5$ fois supérieure.

– Les drivers de lignes ne peuvent pour des questions de rapidité être conçus pour avoir une consommation nulle dans le cas de transmission d'un 0 logique et égale à E_t dans le cas de transmission d'un 1 logique. La consommation dans le deux cas est généralement la même. Le driver est dit polarisé à courant constant. La consommation est donc augmentée d'un facteur 2.

– Pour des raisons d'immunité aux différences de tensions entre les masses de la source et du récepteur, les transmissions sur les liens électroniques sont souvent doublées pour un deuxième guide d'onde transmettant une donnée complémentaire à celle transmisse sur le premier lien (liens différentiels, voir la Fig. 3). Ceci augmente la consommation par un facteur 2.

En pratique, on a donc une consommation globale de l'ordre de 20 fois supérieure à la consommation minimum calculée, c'est à dire une consommation de 200 mW.

Le câble Les guides d'ondes électroniques que ce soit des câbles ou des lignes microstrips, présentent des pertes qui augmentent avec la fréquence. La Fig. 2 représente l'atténuation d'un câble coaxial miniature en fonction de la fréquence d'utilisation. Pour une fréquence de 500 MHz (correspondant à une transmission à 1 Gbit/s), l'atténuation est de 5 dB pour 10 m de câbles. Aussi, 22 % de la puissance transmise est dissipée dans le câble après 10 m.

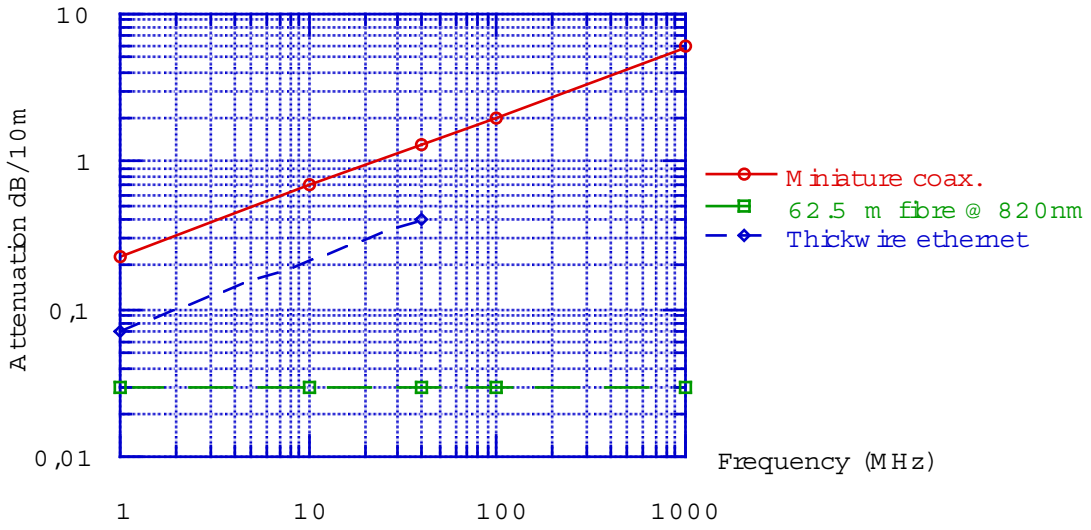


FIG. 2 – *Atténuation de câbles en fonction de la fréquence*

Dans une machine parallèle, les distances de transmission sont malgré tout assez courtes et les répercutions sur le bilan énergétique de cette dissipation sont souvent négligeables en comparaison à l'énergie dissipée dans le driver de ligne.

Un autre problème des transmissions électroniques est la dispersion. Les impulsions de tension représentants les données se déforment en se propageant rendant difficile la distinction des données entre-elles. Nous ne parlerons pas particulièrement de ce problème ici, les distances considérées rendant ce phénomène négligeable.

Cependant, l'atténuation et la dispersion des câbles électroniques montrent que ce média de transmission ne peut être utilisé à haute fréquence que pour des longueur assez courtes. Pour des distances moyennes, il faut avoir recours soit à des câbles de section plus grosse et plus performants (voir sur la Fig. 2 l'atténuation des câbles ethernet épais) ou bien à des transmissions optiques (voir l'atténuation des fibres optiques sur la figure).

Le récepteur On a pu voir que la liaison série électronique est généralement différentielle pour des raisons d'immunité aux bruits sur les masses du récepteur et de l'émetteur. Aussi, le récepteur sera généralement composé d'une paire différentielle détectant la différence des signaux entre les deux lignes. Une telle paire différentielle est polarisée à courant constant et a donc une consommation constante quelques soient les données en entrée.

L'impédance d'entrée du récepteur doit être égale à l'impédance caractéristique de la ligne guide d'onde. Ainsi, toute la puissance transmise dans la ligne est dissipée dans l'impédance d'entrée et il n'y a pas d'onde réfléchie sur le câble. Pour obtenir cette impédance d'entrée, on a généralement recours à des résistances en parallèle avec la paire différentielle servant à la détection des données.

Pour un lien série à 1 Gbit/s, le courant de polarisation sera typiquement de 100 μA, ce qui fait avec une tension d'alimentation de l'ordre de 5 V, une consommation de 0,5 mW. Cette consommation est de toute façon négligeable devant la consommation du driver. Nous ne ferons donc pas une analyse détaillée de ce récepteur du point de vue de sa consommation.

3.2.3 Conclusion

Les transmissions de données sur lignes à support électrique (coaxial ou microstrip) présente une consommation importante dès lors que sa longueur est de l'ordre de la longueur d'onde de l'onde transmisse (par exemple à partir de $\lambda/10$). En effet, avant que l'onde qui se propage sur la ligne ait fait un aller-retour, l'impédance d'entrée de la ligne vaut l'impédance caractéristique de la ligne qui est en pratique de l'ordre de 50 ou 100 Ω. Si, le débit de la transmission de donnée doit être élevé, et que l'on ne peut attendre ce temps d'aller et retour, il faut :

- empêcher l'onde de revenir en adaptant le bout de ligne à l'impédance caractéristique de la ligne ;
- avoir un driver de faible impédance de sortie (de l'ordre de l'impédance caractéristique de la ligne).

Pour une transmission à une fréquence f=500 MHz, correspondant à un débit de 1 Gbit/s, la propagation se faisant entre deux conducteurs séparés par de la silice (S_iO_2) de permittivité relative ϵ_r=3,9, on peut calculer que la longueur d'onde est :

$$\lambda = \frac{c}{\sqrt{\epsilon_r} f} = 30 \text{ cm}$$

. La longueur de la ligne au dessus de laquelle, une adaptation d'impédance du driver est nécessaire est donc $L_c = \lambda/10 = 3$ cm.

Pour des raisons d'immunité aux bruits de masse, les transmissions se font en mode différentiel sur deux câbles. Deux autres câbles sont nécessaires dans l'autre sens de transmission (liens dits full-duplex), voir la Fig. 3.

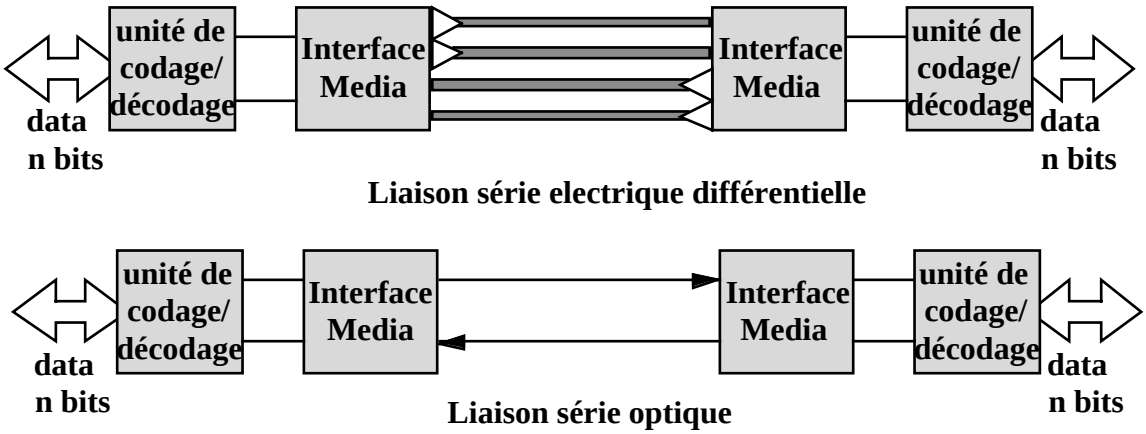


FIG. 3 – *liaison full-duplex électrique et optique*

3.3 Les liaisons séries optiques

3.3.1 Propriétés des liaisons optiques

Remplacer simplement les liaisons séries électroniques par des séries optique apporte-t-il quelque chose et pourquoi ? Nous allons tenter de répondre à cette question dans cette section. La vitesse de propagation d'un signal optique sur un guide d'onde n'est pas significativement différente de celle d'un signal électrique sur un cable. Aussi, l'optique ne peut pas représenter un avantage en terme de latence mais nous allons voir qu'elle peut conduire à des consommations plus faibles que les liaisons électroniques.

Lors de l'étude des liens électroniques, on a pu voir que la consommation importante était due à la nécessité de disposer d'une impédance du driver de ligne égale à l'impédance caractéristique de la ligne. Dans le même temps il faut que les signaux au niveau du récepteur soient d'une amplitude de l'ordre du volt pour des raisons d'immunité aux bruits.

La solution pour pouvoir s'affranchir de cette contrainte est de disposer de convertisseurs d'impédance aux deux extrémités de la ligne (voir la Fig. 4). Deux transformateurs à enroulement classiques pourraient jouer ce rôle, mais ils sont volumineux et très difficilement intégrables dans les technologies actuelles. De plus, ceci conduirait à une faible immunité aux bruits au niveau du cable.

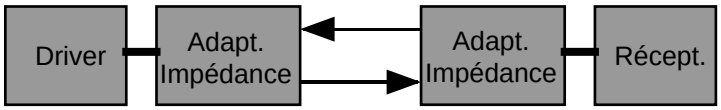


FIG. 4 – *Adaptation d'impédance*

Une solution plus avantageuse pour la réalisation de cette adaptation d'impédance est de recourir à des transmissions optiques. En effet, nous allons montrer que la trans-

duction électrons↔photons agit comme une adaptation d'impédance et permet pour cette raison une réduction de la consommation du liens de communication de données.

3.3.2 Conversion d'impédance

Pour illustrer cette conversion d'impédance, reprenons ici un exemple utilisé dans un article de D. Miller [3]. Pour cela considérons une photodiode chargée par une résistance de 1 MΩ et éclairée par un puissance lumineuse de 2 μW (voir la Fig. 5).

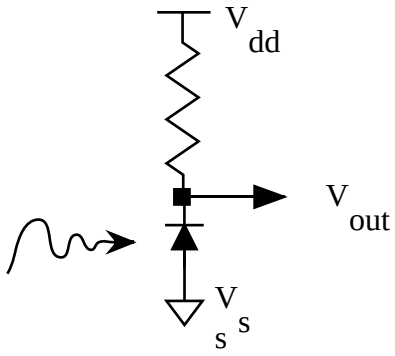


FIG. 5 – *Photodiode avec résistance de lecture*

On peut obtenir avec cette puissance, un courant de l'ordre de 1 μA dans la photodiode, qui dans la résistance de 1 MΩ réalisera la tension de 1 V recherchée. Sur un ligne électrique d'impédance caractéristique 100 Ω, cette puissance (E_{LL}/τ) correspondrait, d'après l'équation 1, à une tension de 20 mV. On peut donc voir clairement illustré ici, le principe de la conversion d'impédance opéré par la transformation photons→électrons.

3.3.3 Bilan d'énergie

Consommation dans la source ou driver de ligne La consommation dans le driver de diode laser est la source principale de consommation d'énergie de la liaison optique. Les diodes laser ont une caractéristique typique semblable à celle représentée sur la Fig. 6. Sur cette caractéristique, on voit clairement que la diode laser présente un seuil et ne génère de puissance optique qu'au dela de ce seuil.

La consommation dans le driver est essentiellement due à deux facteurs :

- la consommation due au courant de seuil est la consommation principale des liaisons optiques que nous avons considérées. Ce courant de seuil diminue avec l'évolution des diodes laser et il n'y a guère de raisons pour qu'il ne diminue pas encore dans le futur.

- le facteur d'efficacité différentielle représente le rapport entre la puissance optique et la puissance électrique consommée par la diode laser lorsque celle-ci est modulée autour d'un point de fonctionnement. Les diodes laser modernes présentent des facteurs d'efficacité différentielle qui peuvent atteindre 80%.

La diode laser qui a les caractéristiques de la Fig. 6 peut-être utilisée pour des liaisons optiques si elle est parcourue par un courant de l'ordre de 4 mA. Si, l'on suppose qu'aucun courant ne parcourt la diode laser lors de la transmission d'un niveau logique

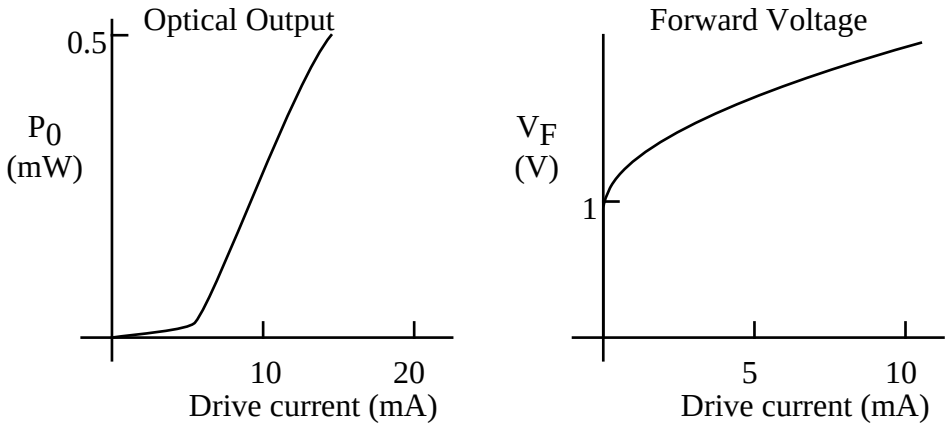


FIG. 6 – *Caractéristique typique d'une diode laser*

'0' et que le driver de cette diode est alimenté avec une tension V_{dd} de 5 V, on peut estimé que la consommation dans le driver sera au minimum :

$$P_d = 0.5 V_{dd} I \approx 10\ mW$$

Le média de communication Le média de communication que ce soit l'air (pour les liaisons en espace libre) ou la fibre optique présente des atténuations qui sont négligeables. En effet, les distances de communication restent faibles dans les machines parallèles et cette partie de la consommation est négligeable.

Le récepteur Les récepteurs étant à haute impédance, la consommation dans ceux-ci est là encore négligeable et nous ne présenterons pas en détail ici la consommation qui est engendrée dans le récepteur. Pour une analyse détaillée, on peut se référer à [1].

3.3.4 Conclusion

L'essentiel de la consommation de la liaison optique se trouve dans le driver de diode laser et l'on a pu voir qu'elle est de l'ordre de 10 mW et tend à diminuer avec l'évolution des technologies. Ce chiffre n'est pas du à une limite physique comme dans le cas des liaisons électronique. On peut donc, ici conclure que l'utilisation de liaisons optiques pour les connexions entre cartes est avantageuse en terme de consommation dès que le couple débit/distance oblige d'avoir recours à des liaisons électroniques adaptées en impédance.

4 Conclusion

La contribution des laboratoires IEF et IOTA dans ce projet ILIAD a porté sur les liens série à haut débit qui relient les processeurs aux bancs mémoires à l'intérieur des groupes locaux. L'objectif est de comparer les technologies optiques et électroniques pour la réalisation de ces liaisons série. Les travaux ont abordé deux points principaux. Tout d'abord, une étude bibliographique a été menée. Elle a porté d'une part sur les liaisons optiques et d'autre part sur les contraintes de réalisation des interconnexions

dans les architectures multiprocesseurs. Cette étude bibliographique a été complétée par un état de l'art des réalisations industrielles d'interconnexions optiques et de liens série à haut débit. Une attention particulière a été accordée à la technologie BULLIT de la Société BULL, qui a été présentée en détail par R. Marbot [2]. Ensuite, une analyse des transmissions électroniques et optiques a été effectuée. Elle a permis de dégager les points critiques pour leur comparaison. Les transmissions optiques bénéficient d'un bilan énergétique meilleur que celui des transmissions électroniques au-dessus d'une longueur critique de transmission L_c. Cet avantage résulte des propriétés suivantes :

- les transformations électron-photon et photon-électron agissent comme des adaptations d'impédance. Les transmissions optiques permettent ainsi la conception d'interfaces électroniques avec des impédances d'entrée ou de sortie de grande valeur et donc de consommation très faible ;
- la bande passante des fibres optiques est intrinsèquement plus grande que celle des lignes à deux conducteurs, ceci conduit à des pertes en ligne très faibles ;
- les transmissions optiques profitent des progrès récents des technologies de réalisation de diodes laser.

Enfin les potentialités offertes par les technologies classiques et émergentes pour les interconnexions optiques ont été approfondies. Pour des liaisons séries avec des débits de l'ordre du Gigabit, la longueur critique L_c peut être évaluée à quelques centimètres. Cette valeur théorique montre que des liaisons optiques entre circuits intégrés (au sein d'un circuit imprimé) seraient très avantageuses, cependant les technologies nécessaires ne sont pas encore disponibles. Nous avons donc dans le projet ILIAD étudié plus spécifiquement les liaisons entre cartes et entre racks.

Références

[1] M. R. Feldman, S. C. Esener, C. C. Guest, and S. H. Lee. Comparaison between optical and electrical interconnects based on power and speed considerations. *Applied Optics*, 27:9, 1988.

[2] R. Marbot, A. Cofler, J.-C. Lebihan, and R. Nezamzadeh. Integration of multiple didirectional point-to-point serial links in the gigabits per second range. In *Hot Interconnects Symposium*, 1993.

[3] D. A. B. Miller. Optics for low-energy communications inside digital processors: quantum detectors, sources, and modulators as efficient impedance converters. *Optics Letters*, 14:2, 1989.

ANNEXE 4
ATM et calcul massivement parallèle

Gilles BERGER SABBATEL
Olivier JACQUIOT
Olivier ONDOA

26 février 1996

LGI - GRAM

L'ATM est un protocole de communication développé pour les réseaux de télécommunications à haut débit. Ses performances amènent à envisager son utilisation dans des domaines éloignés de ses objectifs initiaux. Dans ce document, on effectue une revue des travaux effectués dans ce domaine, et plus particulièrement ceux qui concernent le calcul parallèle. Ceci nous amènera à proposer l'utilisation de l'ATM comme moyen d'interconnecter des éléments de traitement pour réaliser des machines parallèles. Les performances permises par la technologie actuelle limitent cette approche aux machines à communication par passage de message et à la mémoire virtuelle distribuée, avec un grain qui reste moyen à gros. L'obtention de performances plus élevées demande des implémentations de réseaux ATM à faible latence respectant autant que possible les propriétés de l'ATM.

Table des matières

1 INTRODUCTION

L'ATM (Asynchronous Transfer Mode, ou mode de transfert asynchrone [1][2][3]) a été défini par les opérateurs de télécommunications pour le transfert à grande distance et à haut débit de sons, images et données. Ses performances annoncées le mettent directement en concurrence avec des technologies de transmission plutôt destinées aux réseaux locaux, telles que Fast Ethernet, ou FDDI. Un certain nombre de travaux ont donc été effectués pour définir des mécanismes standard d'émulation des fonctionalités nécessaires au support efficace de protocoles existants, tels que TCP/IP, dans le cadre des réseaux locaux [4][5][6].

Avec les technologies de transmission optiques, l'ATM permet d'obtenir des débits du même ordre de grandeur que ceux qui sont couramment rencontrés à l'intérieur des systèmes informatiques (entrées sorties, bus de fonds de panier, réseaux d'interconnexion des machines massivement parallèles). Ceci a ainsi amené un certain nombre de chercheurs à s'interroger sur les limites des possibilités de l'utilisation de l'ATM dans tous les domaines de la communication informatique, c'est à dire non seulement entre systèmes, mais aussi à l'intérieur des systèmes.

En particulier, l'échec commercial de certaines machines massivement parallèles a amené à rechercher d'autres solutions pour le calcul à hautes performances, en s'appuyant sur des technologies préexistantes, telles que les stations de travail et les réseaux à haut débit.

Dans la suite de ce document, après une rapide présentation de l'ATM, nous passerons en revue les principaux travaux de recherches visant à l'utiliser dans ces domaines. Ceci nous amènera à proposer une architecture de machine parallèle basée sur un réseau ATM. Nous discuterons des principaux problèmes posés par une telle architecture, et proposerons des solutions pour l'implémentation de réseaux ATM dédiés au calcul parallèle.

2 L'ATM

Les réseaux de télécommunications procèdent généralement à un multiplexage temporel synchrone de plusieurs voies logiques sur une seule ligne physique : la ligne est allouée successivement à chaque voie, pour un intervalle de temps de longueur fixe. Ce système est simple, et bien adapté aux communications à bande passante fixe, comme la téléphonie ou les liaisons spécialisées. Cependant, il est assez rigide, et mal adapté aux transmissions de données, parce que la bande passante non utilisée par une voie ne peut pas être utilisée par une autre voie.

Pour les transmissions de données, on préfère donc le multiplexage asynchrone, dans lequel il n'y a plus de relation entre le numéro d'une voie et l'intervalle de temps de transmission alloué à cette voie : la voie doit donc être identifiée par un numéro inséré dans les paquets de données.

L'ATM a pour objectif de proposer un protocole de transmission unique pour le transport à haut débit de la téléphonie, de la vidéo, et des données. Ceci a amené au choix du multiplexage temporel asynchrone de très courtes unités de données de taille fixe : les cellules. La taille fixe des cellules a pour but de simplifier l'allocation des mémoires tampon pour l'implémentation en matériel de commutateurs à haut débit. Leur faible taille permet d'une part de réduire le délai de remplissage des cellules pour

la téléphonie, et d'autre part d'éviter qu'un paquet de données de haute priorité puisse être retenu par l'attente de transmission d'un long paquet de priorité inférieure.

2.1 Cellules ATM

Une cellule ATM est composée d'un en-tête de 5 octets et de 48 octets de données utiles (charge utile). La structure d'une cellule ATM est présentée sur la figure 1. L'en-tête comporte un identificateur de connexion de 28 bits, quelques bits de contrôle, et un code de contrôle d'erreur (HEC). Il n'y a pas de contrôle d'erreur pour les données au niveau ATM : le contrôle et la récupération d'erreurs sont reportés sur des couches de protocoles supérieures.

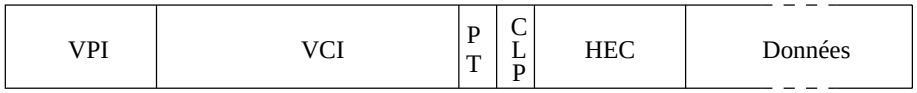


FIG. 1 – *Format d'une cellule ATM*

L'identificateur de connexion permet de définir des canaux virtuels (VC) et des chemins virtuels (VP), qui permettent d'introduire deux niveaux de multiplexage, comme indiqué sur la figure 2. Il est donc composé de deux champs : un identificateur de canal virtuel sur 16 bits (VCI), et un identificateur de chemin virtuel sur 12 bits (VPI), cependant, au niveau des extrêmités, le VPI est réduit à 8 bits, 4 bits étant réservés au contrôle de flux (ces bits sont actuellement inutilisés). Un indentificateur de connexion est local à un lien, et sa valeur est normalement réaffectée pour chaque lien traversé durant la transmission d'une cellule.

Les bits de contrôle comportent trois bits de type de charge utile (PT), utilisés pour distinguer les cellules de contrôle et les cellules utilisateur, et un bit de priorité de perte de cellule, permettant d'indiquer si une cellule peut être rejetée en cas de congestion.

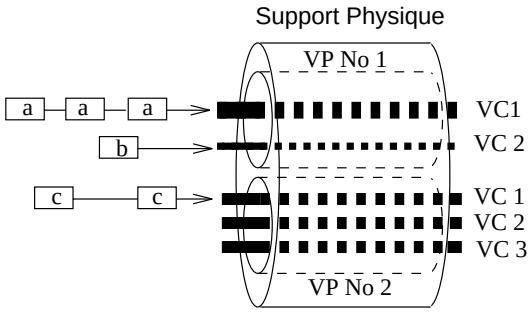


FIG. 2 – *Multiplexage des VP et VC*

2.2 Réseaux ATM

Un réseau ATM consiste en un ensemble d'hôtes et de commutateurs reliés par des liaisons série en point à point. Les connexions sont normalement ouvertes par l'intermédiaires de serveurs de connexions. Les communications en mode non connecté peuvent être émulées par l'intermédiaire de serveurs sans connexions et de connections permanentes établies entre eux.

Pour chaque cellule entrante, le commutateur doit déterminer le lien de sortie, les nouvelles valeurs des VPI et VCI à insérer dans l'en-tête de la cellule réémise, et éventuellement les caractéristiques de la connexion. Ceci est fait normalement par recherche dans une table de connexions.

Les commutateurs disponibles actuellement sur le marché ont des délais qui vont d'une dizaine à une cinquantaine de microsecondes. Les débits normalisés et généralement implémentés pour la transmission sont de 155 et 622 Mbits par seconde. L'implémentation de débits supérieurs (2,5 et 10 Mbits/s) fait l'objet de travaux actifs de recherche et développement.

2.3 Qualité de service

Pour pouvoir répondre à des besoins de communications très différents, tels que voix, données et images, l'ATM définit un certain nombre de paramètres en ce qui concerne la qualité de service :

- débits minimum, moyen et de crête,
- délai de transfert, et variation de ce délai (gigue),
- taux de pertes de cellules.

Quatre classes de services sont définies. Deux d'entre elles sont prévues pour la transmission de sons et d'images compressés ou non, et demandent que le débit et le taux de pertes de cellules soient spécifiés : CBR (Constant Bit Rate) et VBR (Variable Bit Rate). Les deux autres sont prévues pour la transmission de données, et correspondent à une qualité de service du type "best effort" (meilleur effort) : ABR (Available Bit Rate) et UBR (Unspecified Bit Rate).

Les classes de service ABR et UBR permettent de transmettre tant que la bande passante est disponible, mais en cas de congestion, il existe un risque de perte de cellules. Avec la classe de service ABR, le taux de pertes de cellules, le débit de crête et le débit moyen sont spécifiés, et un contrôle de flux par régulation de débit à la source est implémenté pour minimiser le risque de pertes de cellules en cas de congestion. Avec la classe de service UBR, il n'existe aucun mécanisme de régulation, et seul le débit de crête est spécifié : en cas de congestion, il y a donc un risque important de pertes de cellules.

2.4 La couche d'adaptation à l'ATM (AAL)

Quatre couches d'adaptation à l'ATM (ATM Adaptation Layer) sont définies au dessus de l'ATM. Leurs fonctionalités sont prévues pour être implémentables en matériel au niveau de l'interface ATM. Deux d'entre elles (l'AAL1 et l'AAL2) sont plutôt prévues pour des services tels que la transmission du son ou de la vidéo, ou encore l'émulation de lignes spécialisées. Les deux autres (l'AAL3/4 et l'AAL5) sont prévues pour la transmission de données.

L'AAL 3/4 est en fait la fusion de deux AALs initialement définis pour le traffic en mode connecté et en mode non connecté. Avec cet AAL, on ajoute aux messages un en-tête de 4 octets, et une terminaison de 4 octets. Pour la transmission en mode non

connecté, on ajoute à l'ensemble une vingtaine d'octets d'informations de contrôle comprenant autre autres les adresses source et destination. L'ensemble est ensuite segmenté en tronçons de 44 octets, auxquels on ajoute 4 octets de données de contrôle pour former la charge utile d'une cellule ATM. Ces données de contrôle ont pour but essentiel de permettre le multiplexage de messages sur une même connexion : ceci facilite en effet l'implémentation du mode non connecté par le biais des serveurs sans connexion, en évitant à ces derniers de devoir réassembler les messages avant retransmission. L'AAL 3/4 permet optionnellement la détection et la reprise d'erreurs de transmission.

L'AAL 5 est une couche d'adaptation simplifiée ne permettant pas le mode non connecté. Avec cet AAL, on ajoute en queue des messages 8 octets de données de contrôle, et l'ensemble est segmenté en tronçons de 48 octets formant la charge utile des cellules. La fin du message est marquée par une cellule comportant un type de charge utile spécifique (PT).

2.5 Les développements de l'ATM

L'ATM a été défini à l'origine pour être utilisé dans les réseaux de télécommunications. Un certain nombre d'expérimentations sont en cours pour l'utilisation de l'ATM pour du transport de données à grande distance, en particulier dans le cadre de l'interconnexion de réseaux locaux.

Cependant, la possibilité d'utiliser l'ATM dans le cadre des réseaux locaux a rapidement paru importante, de sorte que des produits ATM destinés aux réseaux locaux sont maintenant disponibles chez de nombreux fournisseurs. Ces produits sont souvent destinés à la transmission de données seulement : certains d'entre eux implémentent uniquement les AAL 3/4 et 5, et n'offrent aucune garantie de qualité de service. L'ATM se pose cependant en concurrent sérieux de technologies telles que l'Ethernet rapide ou FDDI, orientées plus spécifiquement vers le transfert de données dans les réseaux locaux : la possibilité d'utiliser un protocole unique pour les réseaux locaux et à grande distance est en effet un atout considérable de l'ATM.

Les caractéristiques de l'ATM ont, par ailleurs, amené à envisager son utilisation dans d'autres domaines que les réseaux d'ordinateurs (locaux ou à grande distance).

Ainsi, Sachs, Leff et Sevigny [7] remarquent que, avec les nouvelles technologies des réseaux à haut débit (dont l'ATM fait partie), les différences entre les communications entre les ordinateurs et leur périphérie (entrées/sorties) et entre ordinateurs (réseaux) tendent à s'estomper, de sorte qu'une seule technologie de transmission pourrait, à terme, couvrir l'ensemble de ces besoins.

A Cambridge, Mark Hayter et al. vont plus loin, en proposant une architecture de station de travail dont le bus fond de panier serait remplacé par un réseau ATM : le Desk Area Network [8][9]. On assemble ainsi les différents éléments de la station de travail autour du commutateur ATM : le processeur et son cache, la mémoire, l'interface réseau local, l'écran, le clavier, une caméra ATM, et un périphérique audio (micro + haut parleur).

Un des point important à noter est que le cache externe associé au processeur communique avec la mémoire par le biais du réseau ATM. Une des motivations de cette architecture est que les flots de données multimédia temps réel sont généralement amenés en mémoire par le biais du bus, pour être ensuite amenés dans le cache externe du processeur (toujours via le bus) afin d'être traités. Le cache externe est donc im-

plémenté de façon à permettre d'y amener directement (sans passer par la mémoire) les données provenant d'un flot de données multi-média. Le problème posé par cette architecture est cependant la latence excessive des défauts de cache, qui reste bien au dessus de ce qui serait compatible avec la rapidité des processeurs actuels. La réduction de ces latences demanderait de réduire d'au moins un ordre de grandeur le délai du commutateur ATM.

Un autre domaine d'application qui fait l'objet de recherches actives est le calcul parallèle. Ce domaine fera l'objet du reste de cet article.

3 CALCUL PARALLÈLE SUR ATM : ÉTAT DE L'ART

De par les performances qu'ils promettent, les réseaux ATM ont fait l'objet d'un assez large intérêt dans le domaine du calcul parallèle. On peut ainsi envisager d'améliorer fortement les performances des applications distribuées tournant au dessus de réseaux locaux conventionnels du type Ethernet. En poursuivant le raisonnement, on peut aller jusqu'à envisager le remplacement de machines massivement parallèles par des réseaux locaux de stations de travail, voire concevoir des machines massivement parallèles dont le réseau d'interconnexion soit un réseau ATM.

Nous classerons en trois catégories les travaux effectués jusqu'ici dans le domaine du calcul parallèle sur ATM :

- Les implémentations sur des interfaces matériels et logiciels standard. Ces travaux sont basés sur un niveau de communication ATM (généralement les API ATM de Fore), implémenté au dessus du système d'exploitation. Les travaux de cette catégorie sont les seuls qui puissent donner une indication sur les performances effectivement possibles avec les matériels et logiciels standard du marché.

- Les implémentations basées sur des interfaces logiciels de bas niveau, intégrés finement dans le système d'exploitation. Ces travaux permettent de se faire une idée des performances qui peuvent être obtenues par des moyens purement logiciels.

- Les travaux basés sur des évaluations, de nature plus spéculative.

3.1 Interface standard

3.1.1 Opérations collectives sur ATM (Huang et al.)

Le problème de la communication sur réseau ATM pour le calcul parallèle a été étudié par Huang, McKinley et Kasten, de l'Université du Michigan[10][11][12]. Leurs travaux sont basés sur l'implémentation d'opérations de communication en point à point ou collectives au dessus d'un réseau ATM. La plate-forme utilisée est basée sur des stations de travail Sun SparcStation-10, équipées d'adaptateurs ATM Fore SBA-200, et interconnectées par l'intermédiaire d'un commutateur ATM Fore ASX 100. Le réseau ATM peut être utilisé soit par l'intermédiaire de l'interface socket standard au dessus de TCP/IP, soit directement par l'intermédiaire d'un interface applicatif spécifique (API[1]), d'un niveau équivalent aux sockets.

1. Les informations publiées ne spécifient pas s'il s'agit des API de Fore Systems, ou d'autres API.

Les évaluations de temps d'aller retour d'un message montrent que l'on gagne un rapport 2,7 en passant d'Ethernet à l'ATM sous TCP/IP, et un rapport 2,8 en passant de TCP/IP aux API sous ATM.

Cependant, les auteurs se sont plus particulièrement intéressés à l'implémentation des opérations collectives, et ont développé dans ce but plusieurs algorithmes de diffusion. La réduction des latences par rapport à l'implémentation de ces opérations sur Ethernet est considérable. La comparaison met en particulier en évidence l'intérêt d'utiliser un réseau maillé, dans lequel la contention sur le support est supprimée, ou fortement réduite par rapport à Ethernet.

3.1.2 PVM sur ATM (Lin et al.)

Lin et al., de l'Université du Minnesota, se sont intéressés à l'implémentation de PVM au dessus d'un réseau ATM [13]. PVM [14] est un interface de communication par passage de messages qui a été défini pour permettre le calcul parallèle sur un réseau de stations de travail interconnectées par un réseau local de type Ethernet. Il a connu un grand succès et est maintenant considéré comme un quasi-standard de fait pour la programmation parallèle sur différents types d'architectures parallèles, depuis des machines massivement parallèles telles que la CM5 ou la Paragon, jusqu'aux réseaux locaux de stations de travail.

La configuration utilisée est un réseau de stations de travail Sun interconnectées par des adaptateurs ATM Fore et un commutateur Fore ASX-100. L'implémentation permet d'utiliser le réseau soit à travers l'interface socket classique, au dessus de TCP/IP, soit au dessus d'un interface applicatif spécifique, les API ATM de Fore. Celles-ci présentent des fonctionalités similaires à celles des socket.

L'ATM utilisé à travers les API Fore propose un niveau de communication non fiable. Une couche implémentant le contrôle de flux et la reprise d'erreurs de transmission a donc été implémentée au dessus. De plus, les fonctions de diffusion de PVM ont été réimplémentées pour permettre d'utiliser les possibilités de diffusion offerte par l'ATM.

Plusieurs niveaux de communication logiciels sur l'ATM ont été comparés :

- Les API ATM de Fore, avec les AAL 3/4 ou 5,
- Les sockets en mode stream (TCP),
- PVM (version 3.3.2) avec le mode de communication direct, dans lequel la communication de processus à processus se fait par l'intermédiaire d'une connexion TCP directe entre processus (donc sans passer par le démon PVM, ce qui est le mode de communication normal),
- Les appels de procédure à distance de Sun, avec la transmission en représentation externe (RPC/XDR).

Sans rentrer dans des détails superflus, on peut dire que PVM et les RPC atteignent des performances très proches. L'utilisation des API Fore montre des performances qui dépendent très peu de l'AAL utilisé, et sont pratiquement le double de celles de PVM ou des RPC, que ce soit en termes de latences de communication ou en termes de débit effectif. L'utilisation des sockets se place à un niveau intermédiaire entre les deux.

L'utilisation de différents modes de communication dans PVM a ensuite été comparée, en utilisant toujours le mode direct :

- Les API ATM de Fore, avec les AAL 3/4 ou 5,
- TCP/IP sur l'ATM,
- TCP/IP sur Ethernet.

La comparaison des débits obtenus avec ces trois mode de communication montre un net avantage de l'ATM sur Ethernet, avec des débits de crête respectifs de 27,2 (avec les API Fore) et de 8,3 Mbits/s. L'avantage des API de Fore sur TCP/IP est plus modéré (20,8 Mbits/s pour TCP/IP). Il n'y a pas de différence de performances vraiment significative entre les deux AAL.

Par contre, la comparaison des latences ne montre pas d'avantage décisif de l'ATM sur Ethernet (1,9ms contre 1,5ms), et la différence entre les API Fore et TCP/IP est marginale.

Par ailleurs, la comparaison de performances entre les applications sur ces différents modes de communication ne montre aucun avantage pour l'ATM.

Ces résultats décevants peuvent s'expliquer par plusieurs facteurs. Les latences de communication d'application à application sont largement dominées par les temps de traitement logiciels : l'avantage d'un système de communication plus rapide est donc rendu négligeable. Certaines limitations des API de Fore (taille maximum des messages, fiabilité) ont par ailleurs obligé à implémenter au dessus une couche d'adaptation compensant les fonctionalités manquantes (segmentation des messages, reprise sur erreur). Le traitement résultant est alors d'une complexité équivalente à TCP/IP, sans avoir la même efficacité au niveau de l'implémentation.

On remarque cependant que ces résultats ont été obtenus sur des stations de travail assez anciennes, en utilisant la transmission en représentation externe. L'utilisation de stations de travail plus récentes, en utilisant la représentation native lorsque c'est possible (configurations homogènes), permettrait de réduire les latences logicielles de manière substantielle, et probablement de tirer plus d'avantages de l'ATM.

Par ailleurs, l'utilisation des possibilités de diffusion de l'ATM permet un gain de performances très important pour ces opérations : la diffusion vers N destinataires se fait en un temps pratiquement indépendant du nombre de destinataires, et légèrement meilleur que la "diffusion" vers un seul destinataire en utilisant la méthode originale de PVM.

3.1.3 PVM sur ATM (ORNL)

A l'Oak-Ridge National Laboratory, l'équipe de développement de PVM s'est également intéressé à l'implémentation au dessus de l'ATM[15], dans le cadre du développement des futures versions de PVM. A l'instar des travaux de Lin et al., cette implémentation expérimentale est basée sur les API ATM de Fore, et elle implémente des fonctions de contrôle de flux et de reprise d'erreurs de transmission.

Les performances ont été comparées d'une part avec l'utilisation de l'ATM via TCP/IP, et d'autre part avec l'utilisation d'Ethernet.

En ce qui concerne la bande passante, l'AAL 5 présente un avantage significatif sur l'AAL 3/4, avec respectivement 26,6 et 17,8 Mbits/s (ces chiffres sont du même

ordre que ceux de Lin et al). L'utilisation de TCP/IP sur l'ATM est un peu moins performante (16,8 Mbits/s), et Ethernet permet d'atteindre 8,6 Mbits/s.

Par contre, en ce qui concerne la latence, l'utilisation des API Fore ne présente aucun avantage sur Ethernet, alors que l'utilisation de TCP/IP permet un petit gain de performances (1,2 ms, contre 1,7 ms). Aucune amélioration des performances des applications n'a été mise en évidence.

Les résultats obtenus par cette équipe confirment donc ceux obtenus par Lin et al., et montrent que, dans l'état actuel de la technologie, l'utilisation de l'ATM, et particulièrement l'utilisation d'un niveau de communication tel que les API Fore ne permet pas un gain de performances très significatif au niveau des applications. Cependant, l'ATM reste considéré comme une technologie prometteuse, et les travaux dans ce domaine se poursuivent pour tenter d'améliorer les performances.

3.2 Interface de bas niveau

3.2.1 Accès mémoire à distance (Thekkath et al)

Thekkath, Levy et Lazowska, de l'université du Washington, se sont intéressés à l'émulation d'accès mémoire à distance au dessus d'un réseau ATM [16]. Les accès distants sont implémentés sous forme d'instructions co-processeur émulées en logiciel par le biais de codes opérations invalides trappés par le système. Une seule instruction permet de lire ou d'écrire jusqu'à 40 octets, transférés entre l'espace mémoire local et l'espace mémoire distant. La protection entre tâches est assurée, grâce à la notion de segment partagé.

Ceci a été implémenté sur des stations de travail DEC 5000 (processeurs MIPS R3000, cadencés à 25 MHz) interconnectées par le biais d'interfaces Fore et d'un commutateur Fore ASX-100. L'implémentation, à bas niveau dans le noyau du système d'exploitation, a pris grand soin de minimiser l'overhead, en minimisant le volume d'informations à sauvegarder à chaque appel. Compte tenu des informations publiées, il semble cependant peu probable que le contrôle de flux et la récupération des erreurs de transmission soit assurés.

L'écriture à distance de 40 octets prend 209 instructions, et s'exécute en 30μs. Une lecture de 40 octets prend 375 instructions, et s'exécute en 45μs (en effet, la lecture demande un aller-retour). Le débit effectif obtenu est de 35,2 Mbits/s.

3.2.2 Messages actifs (Von Eicken)

Les messages actifs sont un mécanisme de communication de bas niveau proposé par Thorsten Von Eicken [17] pour permettre une communication rapide en recouvrant les communications et le traitement. Tout message comporte en tête l'adresse d'une procédure de niveau utilisateur qui sera exécutée à l'arrivée du message avec le corps du message passé en argument.

Les messages actifs ont été portés sur un réseau composé de deux SparcStations (une SS20, et une SS1+) équipées de cartes ATM Fore, directement connectées sans commutateurs [18]. L'implémentation comporte deux niveau communiquant par des traps : un pilote d'entrées sorties, et une librairie assurant l'interface avec les applications. L'implémentation assure le contrôle de flux de bout en bout, et le contrôle et la

récupération d'erreurs de transmission. La protection se base sur la notion de connexion pour assurer une communication de processus à processus.

Les performances des messages actifs ont été comparées aux accès mémoire à distance de Thekkath et. al. pour une fonctionalité équivallente (lecture ou écriture de 40 octets) : la lecture prend 32 μs, et l'écriture en prend 22, pour un débit de 44 Mbits/s. Rappelons cependant que, contrairement à celles de Thekkath et al., ces mesures ne comportent pas de délais de commutation. Par ailleurs, l'implémentation des messages actifs sur la CM5 prend 12 μs, pour un débit de 80 Mbits/s.

3.2.3 PVM sur ATM (Hausner et al)

Contrairement aux implémentations de PVM précédemment décrites, celle de Hausner et al. se base sur un interface matériel spécifique, et l'intégration de fonctions dans les couches basses du système d'exploitation[19].

Le réseau utilisé n'est pas basé sur l'ATM standard, mais sur l'AN2 [20], un réseau ATM développé par DEC, comportant un mécanisme de contrôle de flux par crédits d'émission, ce qui permet de garantir l'absence de pertes de cellules en cas d'engorgement du réseau.

Les machines utilisées sont à base de processeurs Alpha cadencés à 175 MHz, et connectées au réseau par l'intermédiaire d'un contrôleur spécifique: OTTO. Celui-ci est équipé d'un dispositif de transfert de mémoire à mémoire (scatter-gather DMA) permettant l'échange de données non contiguës entre l'espace mémoire de l'utilisateur et l'interface réseau, et il permet d'aiguiller rapidement les cellules entrantes vers le traitement qui correspond à la connexion. Les processus utilisateur ont, par ailleurs, directement accès aux structures de données contrôlant l'interface.

Cette implémentation permet d'obtenir une latence d'application à application de 78 μs (incluant 10 μs de délai de commutation). Ce temps est inférieur d'un rapport 24 à celui obtenu par Lin est al., mais il faut d'abord remarquer que le matériel utilisé n'est pas comparable: la station Alpha est considérablement plus puissante que le Sun 4 de Lin et al., ce qui peut expliquer une bonne part du rapport de performances.

Aucune reprise d'erreur n'est implémentée : le contrôle de flux garantissant qu'aucune cellule n'est perdue en cas d'engorgement, le réseau est supposé fiable. Ce choix nous paraît discutable: en effet, avec les technologies de transmission optique à haut débit disponibles actuellement, le taux d'erreur par bit (bit error-rate) est estimé à 10^{-9} (avec les progrès de la technologie, on estime qu'il peut descendre à 10^{-14}). Avec un réseau débitant à 155 Mbits/s utilisé à 1%, cela fait en moyenne sur chaque lien une erreur toutes les 11 minutes. Pour un réseau de 20 stations (soit 40 liens unidirectionnels), avec un débit de 622 Mbits/s et un taux d'utilisation de 20%, on atteint en moyenne 12 erreurs par seconde. Même si l'on peut espérer une amélioration de la fiabilité des liens (ou plus exactement des transducteurs électro-optiques), l'augmentation des débits et du nombre de stations connectables, et le développement d'applications utilisant le réseau de manière plus intensive rendent les taux d'erreurs inacceptables. Un système de reprise d'erreurs nous semble donc indispensable.

Par ailleurs, les données de contrôle de la communication ne sont pas protégées en écriture, de telle sorte qu'une tâche peut perturber les communications d'une autre tâche. Ceci n'est évidemment pas acceptable, et sera corrigé dans l'avenir.

On peut s'attendre à ce que la correction des deux points ci-dessus entraîne une

certaine pénalisation des performances, dont l'importance est difficile à estimer.

3.3 Évaluations

3.3.1 Mémoire distribuée partagée (Dwarkadas et al)

Dwarkadas et al., de Rice University, se sont intéressés aux protocoles de maintient de cohérence pour l'implémentation logicielle de la mémoire distribuée partagée [21]. Constatant que la bande passante des réseaux locaux de type Ethernet constitue un goulot d'étranglement pour ce type de système, ils ont choisi de se baser sur l'ATM pour évaluer par simulation l'efficacité de ce type de système.

Les simulations supposent des processeurs RISC cadencés à 40 MHz avec 64 Koctets de cache et une latence mémoire de 12 cycles, des pages de 4 Koctets, et une mémoire locale infinie. Le réseau Ethernet est modélisé comme un bus à 10 Mbits/s, et le réseau ATM est modélisé comme un commutateur cross-bar d'un débit de 100 Mbits/s. La latence de communication prend en considération à la fois les délais de transmission, la contention sur le réseau, et l'overhead logiciel. Quatre programmes, extraits de la suite SPLASH [22], ont été simulés : deux d'entre eux (Jacobi et TSP) sont des programmes à gros grain de parallélisme, avec un fort ratio calcul/synchronisation. Un autre (Water) est un programme à grain moyen, et le dernier (Choleski) est un programme à grain fin.

On ne s'intéressera ici qu'à la comparaison entre Ethernet et l'ATM, puisque le reste du papier (comparaison entre les protocoles de cohérence) ne relève pas du domaine de cet article. Les résultats de simulation montrent un fort avantage de l'ATM pour Ethernet : pour Jacobi, le meilleur speedup obtenu sur Ethernet est de 5 pour 8 processeurs, alors que sur ATM, il atteint 14 pour 16 processeurs (la plus grosse configuration simulée). Des speedups intéressants sont également obtenus pour TSP (10 pour 16 processeurs) et pour Water (8 pour 16 processeurs. Par contre, pour Choleski, le meilleur speedup obtenu est d'à peine 1,3,. en raison du fort taux de synchronisations de ce programme.

Les auteurs ont également évalué l'incidence du débit du réseau sur les performances des applications. Ainsi, à 10 Mbits/s, les performances sont évidemment très dégradées, mais restent très supérieures à celles obtenues sur Ethernet. Par contre, le passage à 1Gbits/s n'apporte qu'un gain de performances assez marginal.

3.3.2 Réseaux de stations de travail (projet NoW)

Le projet NOW [23], de Berkeley, se base sur diverses considérations économiques.

Les auteurs ont d'abord comparé les coûts de diverses configurations de machines parallèles et de réseaux, comportant 128 processeurs SuperSPARC à 40 MHz, 128 postes de travail utilisateurs (consoles de station de travail, ou terminaux X), 128×32 Mega octets de mémoire RAM, 128 Giga octets de disque, et un réseau d'interconnexion scalable. Il ressort de cette comparaison qu'un réseau de 32 SparcStations 10 quadri processeurs accompagné de 96 terminaux X interconnectés par un réseau ATM coûte à peu près deux fois moins cher qu'une CM5 à 128 processeurs accompagnée de 128 terminaux X.

En fait, une machine parallèle implique des coûts de développement élevés qui doivent être amortis sur un marché réduit, alors que les réseaux de stations de tra-

vail utilisent des technologies préexistantes dont le développement est amorti sur un large marché.

Par ailleurs, l'activité des stations de travail d'un réseau opérationnel a été tracée et analysée : on s'aperçoit ainsi que, même durant les heures ouvrables, plus de 60% des processeurs restent totalement inactifs. Ceci amène l'idée de récupérer cette énorme puissance de traitement latente gratuite pour pouvoir effectuer des travaux qui relevaient, jusqu'ici, de super-calculateurs ou de machines parallèles. L'idée est donc de mettre au point un système "non intrusif", permettant de récupérer la puissance de traitement inutilisée, sans pénaliser les utilisations interactives des machines : pour cela, un processus de calcul parallèle peut migrer sur une autre station lorsque l'utilisateur commence à utiliser sa machine.

L'efficacité de différentes solutions pour le calcul à hautes performances a été comparée, à partir d'une application : celle-ci permet de modéliser la dissémination de la pollution atmosphérique dans la région de Los-Angeles. Le temps d'exécution de cette application a été évalué sur un Cray C-90 à 16 noeuds, une Paragon à 256 noeuds, et un réseau de 256 stations de travail RS 6000. Pour ce dernier, quatre configurations ont été considérées : la première utilise un réseau Ethernet, PVM et un système de fichiers standard; la seconde utilise un réseau ATM à la place d'Ethernet; la troisième utilise un système de fichier parallèle, et la quatrième utilise un niveau de communication à faible latence à la place de PVM. Le tableau 1 résume les performances et le coût des diverses solutions.

Machine	temps de traitement	coût
C-90	27s	30M$
Paragon	46s	10M$
RS-6000	27374s	4M$
id + ATM	2211s	5M$
id + FS parallele	205s	5M$
id + messages rapides	21s	5M$

TAB. 1 – *Performances et coûts de quelques solutions pour le calcul parallèle*

L'utilisation d'un réseau ATM pour interconnecter les stations de travail augmente donc le coût de 25%, mais améliore les performances d'un ordre de grandeur. L'utilisation d'un système de fichiers parallèle permet encore de gagner un ordre de grandeur. Enfin, un troisième ordre de grandeur peut être gagné en utilisant un interface à passage de messages à faible overhead. On arrive alors à des performances meilleures que celles du Cray C-90, pour un coût 6 fois moindre.

3.4 Discussion

Les travaux présentés ci-dessus ont essentiellement pour but d'obtenir de meilleures performances que sur les réseaux locaux actuels, et de fournir une alternative aux machines massivement parallèles, au moins pour certains types d'applications.

Ces travaux montrent que les performances de l'ATM ne peuvent pas être pleinement exploitées sans une intégration fine du logiciel de communication au système d'exploitation. Les latences de communication obtenues avec des logiciels d'usage géné-

ral (API Fore ou TCP/IP) sont peu inférieures à celles d'Ethernet, les seuls avantage de l'ATM étant la bande passante disponible et l'absence de contention.

A ce jour, et à notre connaissance, le délai de communication minimal sur un commutateur ATM du commerce est d'une dizaine de microsecondes. Ce délai est à comparer, par exemple, aux latences de bout en bout de 500 à 800 ns annoncées pour l'IBM SP2 avec des configurations allant jusqu'à 256 processeurs. Cependant, on considère généralement que la latence des commutateurs ATM peut être amenée bien en dessous de celles des implémentations actuelles, limitées par la demande du marché, l'évolution des standards qui implique des solutions logicielles, et la nécessité d'assurer la compatibilité avec l'existant.

La latence matérielle du réseau n'est cependant qu'un élément parmi d'autres. En pratique, les délais dûs aux traitements logiciels sont souvent bien plus importants que les délais matériels, et il convient de considérer la latence de transmission d'un message d'application à application. La table 2 résume les latences rapportées dans les publications citées ci-dessus.

Système	latence A/R	débit crête
CM5 + active msg	12 μs	10 MO/s
ATM + active msg	32 μs	5.6 MO/s
Paragon + NX	44 μs	73 MO/s
SP1 + MPL/p	56 μs	8.3 MO/s
API Fore AAL5	1700 μs	4 MO/s
AAL5 + TCP/IP	3900 μs	2 MO/s

TAB. 2 – *performances de quelques interfaces de communication*

On voit donc que, si une implémentation bien intégrée au système d'exploitation comme les active messages peut atteindre des performances proches de celles des machines parallèles, il est loin d'en être de même dès qu'on passe par des interfaces de programmation plus standard.

4 RÉSEAUX ATM POUR LE CALCUL PARALLÈLE

La contribution du LGI au projet ILIAD consiste à étudier la possibilité du calcul parallèle sur réseau ATM, en cherchant à évaluer les limites d'une telle utilisation. Ainsi, à l'extrême, l'ATM pourrait être utilisé dans les réseaux d'interconnexion de machines massivement parallèles, avec, par exemple une architecture analogue à celle des machines IBM SP1/SP2 [24] , le commutateur étant remplacé par un commutateur ATM.

Les avantages d'une telle conception seraient les suivants :

- Interconnexion facile avec les réseaux ATM, ce qui permet l'extension de la machine par interconnexion avec d'autres machines éventuellement éloignées.

- Traitement possible de données multi-média (vidéo, sons), ou de données en temps réel. Ceci ouvre des possibilités d'utilisation dans le domaine de l'informatique industrielle, du traitement d'images animées ou de sons en temps réel (robotique,

amélioration d'images bruitées...), de la synthèse d'images animées ou de parole (présentation des résultats...)...

- Système de communication standardisé, pouvant même devenir un standard d'interconnexion avec la périphérie, ce qui peut entraîner une réduction des coûts.

Une première solution est d'utiliser des éléments du marché : cartes processeurs, mémoires et périphériques de stations de travail ou de micro-ordinateurs, adaptateurs et commutateurs ATM du marché, logiciels standards : ceci peut constituer une solution économique pour aborder le calcul parallèle.

Nous avons vu cependant qu'une telle approche ne permet pas d'obtenir des performances très élevées. Trois étapes successives permettent d'améliorer les performances de plusieurs ordres de grandeur:

- optimisation des logiciels, intégration dans le système d'exploitation.
- conception d'adaptateurs ATM ad-hoc : de tels adaptateurs pourraient permettre d'intégrer certaines fonctions de communication, et surtout alléger l'interface programme utilisateur - réseau, en minimisant les interventions du système.
- conception d'un réseau ATM spécifique.

Ces trois étapes doivent permettre de passer d'un réseau de stations de travail (ou de micro-ordinateurs) performant à une machine parallèle de performances équivalentes à celles des meilleures machines actuelles. Une telle machine peut en outre être connectée à d'autres machines similaires, au niveau d'un réseau local ou à grande distance, pour répondre à des besoins spécifiques : besoin ponctuel d'une grande puissance de calcul, besoin d'interconnecter des composants développés sur des machines différentes, utilisation de données en mémoire secondaire, ou de périphériques situés sur des sites différents.

Cette démarche pose plusieurs problèmes :

- Vérifier que les caractéristiques de l'ATM correspondent bien aux besoins du calcul parallèle.
- Déterminer les fonctions qui pourraient être implémentées au niveau d'un interface de communication (logiciel/matériel). Ceci passe par une étude approfondie de la manière dont les applications communiquent.
- Déterminer si l'architecture des réseaux ATM correspond bien au type de traffic généré par le calcul parallèle.

Ces trois points feront l'objet des prochaines sections de ce document.

5 ÉTUDE DES CARACTÉRISTIQUES DE L'ATM

Un certain nombre de caractéristiques de l'ATM ont retenu notre attention comme semblant peu adaptées au calcul parallèle. Certains des problèmes soulevés ont été résolus par l'évolution du standard ATM : c'est le cas du problème de l'allocation de

la bande passante, avec la définition des classes de service ABR et UBR qui rendent facultative cette allocation. Cependant, ces classes de service reposent sur un contrôle de flux dont l'efficacité reste à démontrer pour certains types de traffic, dont celui généré par les applications de calcul parallèle. Par ailleurs, l'utilisation du mode connecté et la taille des cellules demandent plus de développement.

5.1 Le mode connecté

Les modèles de communication couramment considérés dans le domaine du calcul parallèle se basent sur une communication en mode non-connecté, alors que l'ATM fonctionne en mode connecté. Les questions que l'on peut se poser sont donc: est ce que cela pose un problème, et si oui, comment peut on implémenter le mode non connecté?

5.1.1 Le mode non connecté est il indispensable?

Le mode connecté suppose que, préalablement à toute communication, les processus communiquants auront établi entre eux un chemin virtuel, en réservant les ressources nécessaires aux communications ultérieures.

Pratiquement, un modèle de communication en mode non connecté (tel que MPI) peut être implémenté au dessus d'un réseau fonctionnant en mode connecté de plusieurs façons:

- en établissant statiquement une connexion entre chaque paire de processus (ou de noeuds de traitement) pouvant communiquer. Ceci suppose que le nombre d'interlocuteurs potentiels ne soit pas trop élevé. Dans MPI, la notion de groupes de processus et de topologies virtuelles peut d'ailleurs permettre d'optimiser le choix des connexions à établir.

- en ouvrant des connexions à la demande. Si les ressources disponibles ne permettent pas l'ouverture d'une nouvelle connexion, une connexion inactive est choisie pour être fermée de façon à libérer les ressources nécessaires.

Ces solutions demandent donc, soit que chaque processus n'ait qu'un nombre limité d'interlocuteurs, soit que les communications présentent une bonne localité temporelle, c'est à dire que la destination d'un message ait une forte probabilité d'être à nouveau destination d'un message dans un laps de temps limité.

Ces propriétés ont été étudiées en particulier par Hsu et Banerjee [25], en se basant sur des mesures effectuées à partir d'applications de CAO et de calcul numérique. Pour ces applications, une pile des dernières destinations est maintenue. Si un transfert de message a pour destination l'une des destinations contenue dans la pile, il y a succès. Sinon, il y a échec, et une destination contenue dans la pile est choisie (selon la politique LRU) pour être remplacée par la nouvelle destination. Les mesures montrent qu'un comportement très correct est obtenu dès que 3 destinations au moins sont cachées, avec un taux de succès compris entre 86% et 99,96%, selon les applications. Des résultats similaires sont obtenus en ce qui concerne la longueur des messages. Forts de ces résultats, Hsu et Banerjee proposent une architecture de processeur de communication qui exploite ces propriétés [26][27].

Blumrich et al. ont également analysé le comportement d'applications parallèles [28]. Des traces de communication ont été collectées et analysées pour 6 applications tournant sur un iPSC/860 de 16 noeuds. Pour chaque buffer d'émission, on définit la destination dominante comme étant la destination du plus grand nombre de messages émis à partir de ce buffer. Pour quatre applications, tous les messages sont envoyés à la destination dominante. Pour une application, 51% des messages, représentant 99,8% du volume des échanges, sont envoyés à la destination dominante. Pour la dernière application, 25% des messages, soit 53% des données, sont envoyés à la destination dominante. Blumrich et al. proposent donc un modèle de communication dans lequel les transmission vers la destination dominante sont facilitées (projet SHRIMP).

Ces résultats montrent donc que la plupart des applications devraient pouvoir tirer parti d'un système de communication fonctionnant en mode connecté. Cependant, si l'établissement des connexions demande un temps important, certaines applications risquent d'être pénalisées. À nouveau, l'évaluation de ce risque demanderait à être plus poussée, sur la base d'applications réelles, en envisageant la possibilité de définir des procédures rapides pour l'établissement des connexions. Il est cependant aussi possible d'envisager certaines solutions au cas ou un mode non connecté efficace s'avère indispensable.

5.1.2 Comment implémenter le mode non connecté?

L'implémentation standard du mode non-connecté dans l'ATM (voir section 2) ne saurait répondre aux besoins de performances du calcul parallèle. D'autres solutions pour émuler le mode non-connecté sont également définies pour le domaine des réseaux locaux, et s'apparentent aux solutions mentionnées précédemment. Si les performances du mode non connecté s'avèrent réellement critiques, il est alors nécessaire de rechercher une solution au niveau ATM, implémentée en matériel dans les commutateurs.

Des solutions spécifiques sont possibles dans un environnement "propriétaire", sans remettre en cause la facilité d'interconnexion avec l'ATM standard. Ces solutions consistent généralement à réserver certaines plages de valeur des VPI/VCI pour encoder des adresses destination qui seront interprétées par les commutateurs. Les autres valeurs restent donc disponibles pour le mode connecté, ce qui permet d'assurer la transmission correcte des cellules ATM "standard", sachant que la valeur des VPI/VCI est locale à chaque lien. Il est par contre plus délicat (mais pas forcément impossible) de permettre le transport sur un réseau ATM de cellules en mode non connecté propriétaire.

Une solution possible est celle des "adresses courtes", mentionnée par Truong et al. [6]. Dans cette solution, les cellules transmises en mode non connecté empruntent toutes un même chemin virtuel, correspondant à une valeur de VPI prédéfinie. L'identificateur de voie virtuelle (VCI) encode l'adresse destination sur deux octets, sous la forme N.Y, où N identifie le commutateur sur lequel la station destinataire est raccordée, et Y identifie le port de ce commutateur sur lequel elle est raccordée.

Lorsque la cellule est reçue par le commutateur auquel la station émettrice est raccordée, celui-ci reconnaît (d'après le VPI) que la cellule est en mode non connecté. Il utilise alors le premier octet du VCI pour identifier le commutateur destinataire (N), ce qui lui permet de déterminer l'identificateur de chemin virtuel à utiliser pour atteindre ce commutateur (VPI). Il remplace ensuite le premier octet du VCI (N) par le numéro du port d'où provient la cellule (X), et retransmet donc la cellule avec pour VCI

la valeur X.Y. Le commutateur destinataire peut déterminer, d'après le VPI utilisé, le commutateur d'où elle provient. Son identification (M) est alors placée dans le VCI, qui devient M.X, et contient donc l'adresse de la source du message. Le VPI est lui aussi modifié, pour spécifier qu'il s'agit du mode non connecté.

Cette solution n'a pas été retenue par l'ATM Forum, parce qu'elle n'est pas implémentable sur les commutateurs ATM standard. Cependant, l'échanges de cellules en mode connecté avec un réseau ATM standard ne pose pas de problèmes. On remarque aussi que le chemin virtuel reliant deux commutateurs peut comporter des commutateurs ATM quelconques, dans la mesure où ils permettent d'établir un chemin virtuel (identifié par le seul VPI, et non seulement des voies virtuelles identifiées par VPI/VCI), sur lequel la valeur des VCI est préservée.

Christian Paetz, de l'université de Chemnitz, s'est aussi intéressé à l'implémentation de réseaux ATM adaptés au calcul parallèle [29]. Il propose deux techniques d'adressage, selon la taille des systèmes. Dans les deux cas, un bit du VPI est utilisé pour distinguer un VPI normal (appartenant à une connexion) d'un VPI spécifiant une adresse en mode non-connecté. Pour des systèmes de petite taille (moins de 2^{11}noeuds), l'adresse destination et l'adresse source sont spécifiées dans le reste du VPI/VCI. Pour des systèmes plus gros, seule l'adresse destination est spécifiée dans l'en-tête de la cellule, l'adresse source étant spécifiée dans la partie utile de la cellule. En fonction du type d'adresse, les commutateurs traiteront le VPI/VCI de la manière standard, ou appliqueront un algorithme de routage en fonction de la topologie du réseau. Un tel système doit permettre l'échange des cellules en mode connecté avec un réseau ATM standard, mais, contrairement à la solution précédente, il semble difficile d'assurer le transfert des cellules en mode non connecté sur un réseau standard.

5.2 La taille des cellules

La taille des cellules est une caractéristique fondamentale de l'ATM, qui semble à peu près incontournable. Dans le cadre du calcul parallèle, elle pose un problème particulier, dans la mesure où des échanges très courts peuvent être fréquents, et conduire à une mauvaise utilisation de la bande passante du réseau.

Pour évaluer l'importance de ce problème, nous avons recherché les publications présentant des résultats de mesures sur des applications parallèles. Deux publications fournissent des statistiques intéressantes sur la longueur des messages.

Les mesures publiées par Cypher et al. [30] indiquent la distribution des longueurs de messages pour 8 applications parallèles. Une seule de ces applications génère des messages dont la distribution des longueurs pénalise les communications de façon importante (58% de messages de taille inférieure ou égale à 10 octets). Les autres applications présentent des tailles de messages compatibles avec une utilisation correcte d'un réseau ATM.

Les mesures publiées par Horie et al. [31] portent uniquement sur la longueur moyenne des messages générés par 15 applications. 3 d'entre elles génèrent des messages d'une taille moyenne inférieure à 48 octets (respectivement 8 et 12 octets). Les autres programmes génèrent tous des messages d'une taille moyenne de plus d'une centaine d'octets.

On voit donc qu'il existe une large gamme d'applications pour lesquelles la taille des cellules ATM pose peu de problèmes. Des interfaces à passage de messages tels que

PVM ou MPI [32][33] permettent par ailleurs de regrouper les échanges en spécifiant en un seul transfert de message l'émission de données qui ne sont pas contiguës en mémoire, telles que des blocs de matrices, voire des structures de données arbitraires (notion de type dérivé dans MPI).

L'évolution de la puissance des processeurs intégrés entraîne par ailleurs une augmentation de la puissance des noeuds de traitement, de telle sorte que le grain de parallélisme efficace est plus gros, ce qui implique des échanges de messages plus gros. L'augmentation des débits de transmission, par ailleurs, donne un poids de plus en plus important au temps d'initialisation des communications, de telle sorte que les messages courts sont, de toutes façons, pénalisés.

On admettra donc, pour l'instant, que ce problème reste marginal, mais ceci reste une conjecture, et demande à être vérifié par des évaluations plus nombreuses et plus précises, prenant en compte l'évolution de la technologie et des utilisations.

5.3 La gestion du trafic

Lorsque l'ATM a été défini, il était prévu que chaque ouverture de connexion devait spécifier de manière assez précise les caractéristiques du trafic qu'elle allait engendrer, et en particulier la bande passante (moyenne, de crête, minimum). Or, avec les applications de transfert de données, la bande passante nécessaire n'est pas forcément prévisible: elle peut dépendre de l'application, des données mises en jeux, ou du comportement de l'utilisateur. De plus, la plupart des applications de transfert de données peuvent se contenter d'une bande passante réduite (au prix de temps de réponse augmentés), mais peuvent aussi bénéficier d'une bande passante disponible très élevée.

Pour des applications parallèles, la prévision de la bande passante nécessaire n'est pas forcément impossible dans tous les cas, mais on s'attend à ce qu'elle soit souvent difficile ou imprécise. En outre, la bande passante peut évoluer en fonction du déroulement de l'application. La définition de la classe de service ABR (Available Bit Rate) répond à ces problèmes, en permettant aux applications d'utiliser la bande passante disponible sur le réseau, au prix d'un mécanisme de contrôle de flux permettant de minimiser (ou supprimer) les risques de pertes de cellules par congestion.

Le système de contrôle de flux adopté par l'ATM Forum est basé sur la régulation du débit à la source [34]. Dans ce système, les commutateurs observent l'évolution du remplissage des files d'attente d'émission. En fonction de ce remplissage, les cellules peuvent être marquées (dans le champs "Payload Type" de l'en-tête) si une congestion imminente est détectée. A la réception de ces cellules, la destination peut alors renvoyer à la source une cellule de régulation du trafic demandant de réduire le débit dans une certaine proportion.

Un tel mécanisme a été préféré au contrôle de flux de proche en proche ("hop by hop") par crédits d'émission [35], parce qu'il apparaissait plus simple et ne demandait pas une gestion de mémoires tampon par chemin virtuel. Il a été validé par des évaluations et des simulations, essentiellement orientées vers les réseaux à grande distance. Ces simulations modèlisent le trafic en supposant une distribution uniforme des émissions et des destinations des cellules, et en prenant éventuellement en compte le caractère sporadique de certaines sources (émission par rafales, ou "burst").

Ce type de modèle de trafic est supposé correspondre assez bien à la réalité dans le cadre des réseaux de communication: dans un tel réseau, en effet, chaque commutateur

voit se superposer des trafics provenant d'un grand nombre de sources différentes et indépendantes, de telle sorte que le comportement spécifique de certaines sources peut être négligé. Cependant, il rend mal compte du comportement de certaines applications, telles que celles basées sur le client serveur, dans lesquelles les requêtes d'un grand nombre de stations clientes convergent vers un nombre limité de serveurs [36].

Avec le calcul parallèle, on introduit une difficulté supplémentaire: les différentes sources peuvent être synchronisées, et, par conséquent, déclencher toutes une émission d'un gros volume de données dans un intervalle de temps réduit. Si, en raison du type de communication (convergence des données vers un noeud), ou de la topologie du réseau, un tel trafic converge sur un nombre limité de liens, on s'attend à ce que des problèmes d'engorgement apparaissent, et qu'ils soient mal pris en compte par le mécanisme de contrôle de flux standard.

Ici encore, on n'en est qu'au stade des conjectures, et il apparaît nécessaire de modéliser ce type de comportement, pour évaluer les limites de l'utilisation des réseaux ATM standard pour le calcul parallèle. Ces limites peuvent porter sur le nombre de noeuds interconnectables, en fonction du comportement des applications, et il s'agit probablement, à notre avis, d'un des problèmes majeurs qui devraient être pris en compte dans le domaine du calcul parallèle sur réseau ATM. A notre connaissance, aucun résultat n'a encore été publié dans ce domaine, mais une étude est en cours dans notre laboratoire.

Il est évidemment prématuré de proposer des solutions à ce problème, mais on peut envisager quelques directions: adapter la topologie du réseau, déterminer convenablement les paramètres du contrôle de flux, ordonnancer les traitements en fonction de la charge du réseau, ou enfin, utiliser des solutions propriétaires (non standard) pour le contrôle de flux. De telles solutions ont déjà été proposées [35], et ne remettent pas forcément en question la compatibilité avec l'ATM standard: l'interopérabilité avec l'ATM standard est garantie si les extrémités implémentent correctement le protocole de régulation du trafic standard (ou si celui-ci est implémenté au niveau de passerelles), et le contrôle de flux par crédits d'émission peut également traverser un réseau standard (au prix d'une diminution de la qualité de service), dans la mesure où les signaux de contrôle de flux sont transportés par des cellules ATM.

6 L'INTERFACE DE COMMUNICATION

Comme nous l'avons vu, l'utilisation de l'ATM à travers les couches logicielles traditionnelles effectuant l'interface avec le réseau amène des latences de communication d'application à application qui ne permettent pas l'exploitation des performances du réseau. Il est possible de chercher à définir des interfaces de communication logiciels légers, dont l'interface avec le système soit beaucoup moins pénalisant: c'est le cas en particulier des messages actifs de Von Eicken.

Cependant, il nous semble que, pour assurer une transmission sûre avec un délai minimal, des solutions matérielles soient nécessaires, de façon à:

- permettre le transfert aussi direct que possible des données entre l'espace mémoire utilisateur et le réseau,

- permettre de minimiser les interventions du système d'exploitation,

– assurer les protections entre tâches, ainsi que la détection et la reprise d'erreurs de transmission.

Par ailleurs, les fonctions de l'interface doivent être adaptées à l'utilisation qui en est faite, et exploiter au mieux les propriétés de l'ATM. Ceci nous a amené à étudier les différents modèles de communication utilisés dans le calcul parallèle. Ceux-ci se répartissent entre deux catégories : la communication par mémoire partagée (éventuellement émulée), et la communication par messages.

6.1 La mémoire partagée

Dans le contexte du projet ILIAD, l'ATM était vu comme un second niveau de communication permettant d'interconnecter des multiprocesseurs à mémoire commune. L'émulation de mémoire partagée est donc un modèle de communication intéressant, d'une part parce qu'il est généralement considéré comme le plus simple à utiliser, et d'autre part parce qu'il fournit un modèle de communication homogène sur l'ensemble de la machine.

On distingue deux manières d'émuler une mémoire partagée sur un système de communication par messages : la cohérence de caches, et la mémoire virtuelle distribuée.

6.1.1 La cohérence de caches

La cohérence de caches a pour unité de cohérence la ligne de caches, soit généralement 4 mots : ceci est adapté à un parallélisme à grain assez fin. Un certain nombre d'implémentation ont été proposées, parmis lesquelles DASH [37] est probablement la plus connue.

L'implémentation de la cohérence caches au dessus d'un réseau ATM se heurte cependant à plusieurs difficultés. Le première est que les systèmes implémentés jusqu'ici fonctionnaient au dessus de réseaux dont la latence de communication était de quelques centaines de nanosecondes, alors que celle des réseaux ATM se compte au minimum en dizaines de microsecondes. Il faut donc soit supposer possible un gain de deux ordres de grandeur dans les latences du réseau ATM, soit trouver des stratégies de maintient de cohérence qui rendent de telles latences acceptables - au moins pour certains types d'applications.

La seconde difficulté est liée à la gestion des connexions, qui peut être complexe dans certaines configurations de mémoire partagée, ce qui peut pénaliser les performances.

Enfin, la trosième difficulté est liée à la taille des cellules, peu compatible avec la petite taille des messages échangés - et en particulier les messages d'invalidation. En ce qui concerne les transferts de données, l'objectif de bon remplissage des cellules peut imposer le choix de tailles de lignes de caches. Ainsi, une ligne de 32 octets peut tenir dans une cellule avec une perte de 16 octets, soit 33ligne de 64 octets demandera deux cellules, avec une perte de 32 octets, soit toujours 33qu'on peut obtenir un meilleur remplissage : 3 cellules sont requises, soit 144 octets, avec, donc, une perte de 16 octets, soit 11

Il est possible que des stratégies particulières de gestion de la cohérence apportent des solutions à ces problèmes. Par ailleurs, il est possible que les latences de communication des commutateurs puissent être assez fortement diminuées - bien que cela

n'apparaisse pas actuellement comme l;objectif prioritaire des constructeurs. La résolution des ces problèmes demanderait donc un effort de recherche à long terme, qui sort du cadre du projet ILIAD, et la cohérence de cache a donc été abandonnée.

6.1.2 La mémoire virtuelle distribuée

La mémoire virtuelle distribuée a pour unité de cohérence la page de mémoire virtuelle, soit généralement 4K octets. Une telle unité est plus compatible avec l'utilisation efficace d'un réseau ATM, et avec une implémentation logicielle des opérations de cohérence. Par contre, la grande taille des pages augmente le risque du faux partage, phénomène qui se produit lorsque deux processus accèdent en lecture/écriture à deux variables différentes placées dans la même page. Dans un tel cas, le système de gestion de la mémoire virtuelle devra gérer des mises à jour, alors qu'il n'y a pas réellement de partage. Pour cette raison, la mémoire virtuelle distribuée est souvent réservée à des applications présentant un grain de parallélisme assez grossier.

Si les problèmes mentionnés pour la cohérence de caches n'apparaissent plus avec la mémoire virtuelle distribuée, celle-ci présente tout de même un grand nombre de variantes d'implémentation, tenant au modèle de cohérence retenu (cohérence séquentielle, faible, au relachement, etc...), à la manière de propager les mises à jour, etc... Bien évidemment, l'efficacité des différentes solutions peut dépendre fortement des performances du réseau utilisé, et l'architecture du réseau peut aussi être plus ou moins bien adaptée au type de traffic généré.

Pour approfondir l'étude de ces problèmes, nous avons décidé de nous appuyer sur un simulateur de machines à mémoire virtuelle distribuée, permettant de faire varier les divers paramètres et d'en évaluer l'impact sur les performances. Le simulateur a été implémenté sur un PC tournant sous OSF/1 au dessus d'un micro-noyau MACH, et en exploitant les notions de tâches, threads et paginateur externe fournies par ce système.

Le principe du simulateur est de laisser au système hôte l'exécution des tâches correspondant à l'application, et de n'exécuter que les opérations pertinentes en fonction des objectifs recherchés. Ainsi, la notion de paginateur externe de Mach permet de traiter les défauts de page en dehors du système, et ainsi d'expérimenter diverses stratégies de remplacement, ou de simuler le transfert par un réseau. L'élément principal du simulateur est un ordonanceur, qui a pour rôle de simuler le parallélisme, en choisissant l'ordre d'exécution des différents processus client.

6.2 La communication par messages

Bien que moins "intuitive" que la communication par mémoire partagée, la communication par messages est assez couremment utilisée, en particulier parce qu'elle reste encore la plus simple à implémenter et la plus efficace. Elle peut être utilisée directement par le biais d'interfaces logiciels tels que PVM [14] ou MPI [33], ou indirectement, à travers des langages de plus haut niveau comme HPF.

Nous avons choisi de restreindre notre étude à PVM et MPI. PVM semble être le plus populaire à l'heure actuelle, il peut être porté sur de nombreuses machines, depuis le réseau de stations de travail communiquant par TCP/IP sous Ethernet jusqu'aux machines parallèles communiquant par le biais d'un réseau spécialisé. MPI a été défini plus récemment, mais a pour objectif d'être un interface standard de communication par messages. Une étude plus approfondie de MPI peut être trouvée dans [32].

MPI présente un certain nombre de caractéristiques intéressantes de notre point de vue : les notions de groupes de processus, de contextes de communication, de types de données dérivés, et de topologie virtuelle. Les opérations de communications possèdent de nombreuses variantes (par exemple: émissions non bloquantes, localement bloquantes, ou globalement bloquantes), et il existe de nombreuses opérations globales. Du point de vue de l'implémentation au dessus de l'ATM, les notions de topologie virtuelle, et d'objets de communication persistants nous semblent intéressantes, parce qu'il est possible de les exploiter pour déterminer les connexions à établir, statiquement ou dynamiquement. La notion de types de données dérivés permet de définir des ensembles de données de structure arbitraire, éventuellement stockés en mémoire de façon non contiguë, et dont l'émission ou la réception peuvent être spécifiés en une seule opération : ceci permet donc d'assurer un meilleur remplissage des cellules ATM.

PVM est beaucoup plus simple, mais il nous semble constituer un sous ensemble intéressant des fonctionalités de MPI. À l'époque de notre étude, ses implémentations avaient beaucoup plus de maturité que celles de MPI, il en existait une version tournant au dessus de l'ATM, et il était largement utilisé pour des applications réelles. Nous avons donc choisi de nous appuyer sur PVM pour la suite de l'étude.

La version de PVM tournant au dessus de l'ATM a été récupérée, et interfacée avec un simulateur de réseau ATM. L'objectif de cette expérimentation est double : d'une part, permettre d'évaluer le comportement de réseaux ATM avec la charge générée par des applications de calcul parallèle, et d'autre part, fournir un moyen d'évaluation du gain de performances que permet l'implémentation en matériel de certaines primitives de communication.

Une première version du simulateur a été mise au point, mais ses performances ne permettent pas encore l'exécution de programmes PVM de taille suffisante. Il reste donc à corriger ces problèmes : ceux-ci semblent en particulier dus à une simulation de trop bas niveau sur certains points, qui entraîne des commutations de processus (au sens Unix) trop fréquentes. Le portage sur une machine plus puissante (configuration multi-processeurs sous Unix, ou machine parallèle type IBM SP1) pourra aussi être envisagé. Lorsque des performances suffisantes auront été obtenues, il restera à exploiter le simulateur par une campagne de mesures systématiques.

6.3 Fonctions de l'interface de communication

Les fonctionnalités offertes par l'interface de communication constituent un niveau de communication. Celui-ci doit correspondre aux modèles de communication de plus haut niveau proposés au programmeur.

En ce qui concerne la mémoire virtuelle distribuée, seules des fonctionalités de base de type transfert de messages sont requises, à moins d'envisager des stratégies de mise à jour plus sophistiquées, par envoi des seules différences entre deux versions d'une même page..

En ce qui concerne le transfert de messages, nous avons déja évoqué PVM et MPI, qui semblent être, à l'heure actuelle, les deux interfaces de communication par messages les plus utilisés. D'autres modèles de communication existent, à des niveaux plus ou moins élevés, comme Linda [38], un modèle de haut niveau, ou les messages actifs [17], un modèle de plus bas niveau.

Dans le domaine des implémentations matérielles, SHRIMP [28] propose un modèle

de communication original : des zones mémoires (source et destination) de deux processus sont mises en correspondance par le biais de l'adressage virtuel. Les mises à jour de la zone source peuvent être propagées dans la zone destination, soit automatiquement, soit par une commande explicite. Les commandes de l'interface de comunication sont par ailleurs mappées dans l'espace d'adressage virtuel, de telle sorte que le processus utilisateur peut émettre ces commandes sans passer par le système d'exploitation. Ce modèle de communication apparaît séduisant, et se prêterait sans doute assez bien à une implémentation matérielle au dessus de l'ATM. Il s'agit cependant d'une approche particulière de la programmation parallèle, dont il n'est pas prouvé qu'elle pourra s'imposer aux programmeurs, étant donné le poids des standards. Un tel modèle devrait probablement être utilisé à travers un niveau de communication plus élevé (par exemple PVM ou MPI), et il n'est pas évident qu'il soit optimal pour cela.

Nous préférons donc baser la suite de notre étude sur des interfaces de communication existants et largement implémentés et utilisés, PVM et MPI. Ces deux interfaces de communication permettent des communications en point à point, ou des communications collectives sur un réseau homogène ou hétérogène. Dans ce qui suit, on ne s'intéressera qu'aux communications en point à point. L'opportunité d'un support matériel pour les opérations collectives fera l'objet de travaux ultérieurs.

6.3.1 Types de données

Du fait de la possible hétérogénéité du réseau, les informations transmises par PVM et MPI sont typées, de façon à permettre des conversions lors des communications entre machines où les types ont des représentations différentes.

Dans PVM, les données sont traduites par l'émetteur dans un format standard (XDR), et retraduites par le destinataire dans le format local. Si les formats d'origine et de destination sont les mêmes, on a alors deux traductions inutiles. Or, avec la tendance à la standardisation des formats et à la réduction du nombre de types de processeurs différents disponibles sur le marché, il est de plus en plus probable que les différents types de données auront une même représentation entre deux machines.

Cet inconvénient peut être contourné en spécifiant un transfert en représentation native, sans conversion (PvmDataRaw). Cependant, les futures versions de PVM devraient implémenter un autre système, dans lequel les données sont émises dans la représentation de l'émetteur, et converties par le récepteur seulement si nécessaire [15]. L'avantage est alors que seules les conversions nécessaires sont effectuées. L'inconvénient est que le récepteur doit disposer de multiples fonctions de conversion adaptées à chaque format susceptible d'être reçu.

Par contre, le standard MPI ne spécifie pas de méthode particulière pour la conversion entre les représentation de types, et laisse le choix à l'implémenteur.

Dans PVM, les conversions de type sont spécifiées explicitement : chaque élément de données est transmis dans le buffer d'émission à l'aide d'une primitive correspondant au type. Un autre effet de ces primitives est de permettre de regrouper en un seul envoi de message des données de types différents, disposées en mémoire de façon non contigué.

Dans MPI, le même effet est obtenu par l'intermédiaire de la notion de type dérivé : un type dérivé permet de décrire des ensembles de données de types er de structures arbitraires, éventuellement non contigus en mémoire, et dont la transmission peut être spécifiée en une seule opération.

Au niveau de l'implémentation matérielle d'un niveau de communication, il ne semble pas opportun de prendre en compte les conversions de types. La conversion dans un format d'échange commun alourdit les échanges, et est potentiellement inutile, et l'implémentation matérielle de la conversion à la réception semble très complexe, vue la multiplicité des cas possibles. On préfère donc ne pas implémenter de conversions en matériel : les conversions éventuellement nécessaires peuvent être implémentées en matériel. Par ailleurs, pour des hautes performances, on préfèrera privilégier le cas où le réseau est homogène.

Par contre, on retiendra de la notion de type dérivé dans MPI la possibilité de décrire des données dispersées en mémoire, et transmissibles en une seule opération.

6.3.2 Émission et réception de messages

Dans PVM, l'émission d'un message se fait en plusieurs opérations :

- Un buffer d'émission est crée et initialisé par la primitive **pvm_mkbuf(***encoding***)**. Le paramètre *encoding* spécifie le codage des données : conversion en format XDR, envoi en représentation native, ou données laissées en place. Dans le dernier cas, le buffer ne contient que des pointeurs sur les éléments de données à transférer. Cette option permet d'éviter une recopie des données, et simplifie l'émission de valeurs successives d'une même donnée

- Les données à transmettre sont ensuite placées dans le buffer d'émission après conversion éventuelle par une série de primitives correspondant à chaque type de données manipulé. Un exemple est la primitive **pvm_pkint(****np,nitem,stride***)** : elle permet de spécifier l'émission de *nitem* entiers à partir de l'adresse spécifiée par *np*. Le paramètre *stride* indique la distance séparant deux éléments successifs à transmettre. Dans le cas où les données sont laissées en place, les données ne devront pas être modifiées avant l'envoi du buffer.

- L'envoi du buffer d'émission est enfin déclenché par la primitive **pvm_send(***dest, tag***)**. Le paramètre *dest* spécifie la tâche destinataire, et le paramètre *tag* est une étiquette qui doit correspondre avec celle qui sera spécifiée par l'opération de réception.

Il existe par ailleurs la primitive **pvm_advise(***route***)**, qui permet de spécifier une transmission directe de tâche à tâche par l'intermédiaire d'une connection TCP.

La réception d'un message est spécifiée par la primitive **pvm_recv(***source, tag***)**, la valeur des paramètres *source* et *msgtag* devant correpondre respectivement à l'origine du message et à son étiquette, à moins que la valeur -1 soit spécifiée, auquel cas n'importe quelle valeur est acceptée. Cette fonction retourne un pointeur sur le buffer de réception, où les données peuvent être récupérées par une série de primitives correspondant à chaque type de données manipulé. La primitive **pvm_nrecv** est analogue à **pvm_recv**, mais elle est non bloquante : elle retourne un pointeur nul si le message n'est pas arrivé.

Dans MPI, l'émission bloquante se fait par la primitive **mpi_send**(*buf, nitem, type, dest, tag, comm*). Le paramètre *buf* est l'adresse du premier élément de données à transmettre, *nitem* est le nombre d'éléments, et *type* est l'adresse d'un descripteur de type (un type pouvant être un type dérivé, spécifiant une structure de données

complexe). La destination est spécifiée par *dest*, et *tag* a le même rôle que dans PVM. Enfin, le paramètre *comm* spécifie un communicateur, c'est à dire la spécification d'un groupe de processus et d'un contexte de communication.

La réception bloquante se fait par la primitive **mpi_recv**(*buf, nitem, type, source, tag, comm, status*). Les paramètres sont les mêmes que pour l'émission, avec une sémantique analogue à **pvm_recv**.

Il existe également des versions non bloquantes des primitives d'émission et de réception de messages.

On voit que PVM et MPI ont en commun la possibilité de transmettre dans un seul message des structures de données arbitraires, disposées en mémoire de façon non contigué. Il s'agit d'une caractéristique importante pour un système de communication tel que l'ATM, puisqu'il permet de regrouper les données à transmettre en un seul message, et donc d'assurer un meilleur remplissage des cellules ATM.

PVM spécifie un buffer d'assemblage des messages - bien que ce buffer puisse être optionnellement supprimé. MPI ne spécifie pas de tel buffer, ce qui pourrait permettre une transmission directe des données entre la mémoire du processus utilisateur et l'interface du réseau.

Cependant, avec un système de communication non fiable, comme l'ATM, il est nécessaire d'implémenter une couche de communication fiable. Pour cela, il faut conserver une copie des messages transmis, pour pouvoir les retransmettre en cas d'erreur de transmission. Cette copie peut être conservée sous forme d'un message assemblé, ou sous forme d'une suite de cellules ATM. Dans le second cas, la couche fiable est implémentée au dessous de la couche segmentation de l'AAL, ce qui ne correspond pas aux fonctionalités normales de l'AAL. La création et la mémorisation du message assemblé semblent donc inévitable. L'architecture du processeur de communication peut cependant permettre d'effectuer ces opérations en parallèle avec la segmentation et la mise en émission des cellules ATM : ceci est possible avec l'AAL 5, dans lequel les données de contrôle sont insérées à la fin du message.

Au niveau de la réception, il serait souhaitable de pouvoir transférer le message reçu directement dans la mémoire du processus utilisateur, si celui-ci a déja émis l'opération de réception. Sinon, le message devrait être stocké dans une mémoire tampon, après réassemblage. Pour cela, le modèle de communication de MPI semble plus intéressant que celui de PVM, dans la mesure où les informations concernant la destination du message sont disponibles lorsque l'opération de réception est lancée.

La conclusion que l'on peut tirer de ces considérations est qu'on a le choix entre deux philosophies assez différentes :

- la philosophie basée sur MPI : on a un descripteur qui indique la disposition en mémoire des données à émettre ou à recevoir. L'émission ou la réception de ces données sont ensuite déclenchées en une seule opération, sans imposer de mémorisation du message.

- la philosophie basée sur PVM : le message est assemblé élément par élément avant émission. A la réception, il est réassemblé dans un buffer, avant d'être placé dans la mémoire du processus.

La philosophie de PVM semble plus primitive. Elle a cependant l'inconvénient d'imposer un niveau de mémorisation des messages, ce qui pénalise les performances. Un

autre inconvénient majeur de PVM - en particulier pour une implémentation matérielle - est que le buffer d'émission est alloué et initialisé avant même qu'on en connaisse le contenu, et donc la longueur.

Il est par ailleurs probable qu'une grande partie des applications de PVM pourraient tourner sur des environnements homogènes en laissant les données en place, ce qui correspond à peu près à la philosophie de MPI. La philosophie de MPI semble donc préférable, mais le choix entre les deux demanderait un étude d'implémentation plus poussée, de façon à vérifier la possibilité de minimiser effectivement les mémorisations intermédiaires des messages émis.

6.3.3 Gestion des connexions

La gestion des connexions, c'est à dire, en l'occurence, la manière d'associer une conexion avec une requête d'émission ou réception de message, est un problème spécifique de l'ATM. Il existe trois catégories de solutions, correspondant à trois niveaux de multiplexage des messages sur les connexions :

- avoir une seule connexion par destination physique. Ceci signifie que les communications des différentes tâches sont multiplexées sur une seule connexion. Ceci permet de minimiser le nombre de connexions ouvertes. Par contre, on se prive ainsi de la possibilité de protections entre tâches qu'offre la notion de connexion. La tâche destinataire et la tâche source devront être indiquées dans le corps du message, et le démultiplexage devra être effectué par le processeur de communication, en plus du décodage des VPI/VCI. Cette solution ne nous semble pas devoir être explorée plus avant.

- avoir une seule connexion par destination logique (destination physique + tâche). Ceci signifie que les différents contextes de communication d'une même tâche sont multiplexés sur une seule connexion. Le contexte de communication devra donc être spécifié dans le corps du message, et le démultiplexage entre contextes sera effectué par le processeur de communication.

- avoir une connexion par destination logique et par contexte. La notion de connexion va donc encapsuler à la fois la destination logique et le contexte, ce qui est susceptible de simplifier le démultiplexage à la réception, et favorise la séparation entre les communications des différents contextes. Par contre, un nombre plus important de connexions sera nécessaire, et la correspondance entre la destination logique et le numéro de connexion devra être établie par une table locale à chaque contexte.

A ce point, il convient de rappeler que l'ATM comporte deux niveaux de multiplexage des connexions : les canaux virtuels (VC) et les chemins virtuels (VP). On peut envisager de faire correspondre la notion de chemin virtuel avec la notion de destination physique ou logique, mais il faut aussi se rappeler que 8 bits seulement sont disponibles aux extrêmités pour identifier les chemins virtuels, soit 256 destinations (ou sources) possibles au maximum. Par contre, les 16 bits disponibles pour l'identification des canaux virtuels semblent dépasser largement les besoins. Il y a donc un choix à effectuer au niveau de l'implémentation, en fonction des ambitions du système visé : en pratique,

sur un réseau ATM standard, 256 noeuds devrait déja constituer une configuration importante.

On remarque par ailleurs que MPI possède un certain nombre de caractéristiques qui peuvent faciliter la gestion des connexions :

- La notion de groupes de processus, qui permet de partitionner l'espace de communication, et donc de définir les ensembles de processus entre lesquels des communications pourront être établies;
- La notion d'objets de communication persistants, permettant de spécifier que plusieurs communications se feront avec les mêmes paramètres, ce qui peut se traduire par l'ouverture explicite d'une connexion, et son maintient pour la durée de vie de l'objet;
- La notion de topologie virtuelle, qui permet de déterminer des relations privilégiées entre processus, et peut donner lieu à l'établissement d'un réseau de connexions.

6.3.4 Commandes de l'interface de communication

Dans le but de réduire les délais de traitement logiciels, on cherche à permettre le contrôle direct de l'interface de communication par les processus utilisateurs. Le problème est donc de permettre ce contrôle tout en assurant les protections nécessaires.

En pratique, plusieurs cas sont à prévoir :

- Les commandes concernant la gestion des connexions, ou l'utilisation directe des AAL doivent être contrôlées par le système d'exploitation. Les informations concernant les connexions ouvertes peuvent être accédées par l'utilisateur, mais doivent être protégées en écriture.
- Les informations concernant la description des données à transmettre ou à recevoir peuvent être définies et modifées directement par l'utilisateur, à condition qu'elles soient en adresses virtuelles.
- L'émission ou la réception des messages doivent pouvoir être déclenchées directement par les processus utilisateurs, dans la mesure où seules les connexions autorisées sont accessibles.

La commande de l'interface de communication peut s'inspirer de SHRIMP : les commandes sont émises sous la forme d'écritures à une adresse virtuelle particulière. Cette adresse est traduite, par l'intermédiaire de la table des pages, en une adresse physique reconnue par l'interface de communication. L'adresse et la donnée écrite fournissent certains paramètres de l'opération, et en particulier l'adresse d'un descripteur des données. Le numéro de connexion à utiliser peut être déduit de la destination, et éventuellement du contexte.

Un tel mode de commande permet de déclencher une émission en une simple instruction d'écriture en mémoire. La protection est assurée par la translation des adresses : seules les adresses physiques correspondant aux opérations et aux structures de données valides sont accessibles.

7 L'ARCHITECTURE DU RÉSEAU

L'idée initialement suivie dans le projet était de définir des implémentations de l'ATM qui permettent d'obtenir des performances du même ordre que celles des réseaux d'interconnexion des machines parallèles. Ceci implique:

- une forte augmentation des débits, pour atteindre des débits de l'ordre du gigabits/s. De tels débits sont à l'étude dans le cadre du développement de réseaux ATM multi-gigabits (2,5 et 10 Gigabits par secondes).

- une forte réduction des délais de commutation, qui devraient passer d'une dizaine de microsecondes actuellement à moins d'une microseconde. En fait, cette réduction devrait être un corollaire de l'augmentation des débits, puisque une contrainte constante est que les différents étages de commutation ne demandent pas plus d'un temps de cellule, soit 169ns pour un débit de 2,5 Gigabits/s.

L'ATM a été défini pour les réseaux publics de communication à grande distance. Or, les communications à distance moyenne ou grandes dans les réseaux publics ou locaux présentent des contraintes et des propriétés très différentes de celles des réseaux d'interconnexion des machines parallèles.

Dans les réseaux locaux ou à grande distance, l'application type est le transfert de fichiers (transfert au coup par coup de gros volumes de données). En tout point du réseau, le traffic résulte de la superposition de traffics de nombreuses sources indépendantes et de comportements différents. L'élément essentiel du coût est composé par les liens de communication, dont il faut optimiser l'utilisation. La longueur de ces liens est telle que de grandes quantités d'information peuvent se trouver en transit sur le médium (pour une liaison transatlantique à 155 Mbits par seconde, cela peut représenter de l'orde de 500 K octets).

Dans les machines parallèles, les échanges peuvent être courts, avec des synchronisations fréquentes des processus. Le délai de transmission influe directement sur les performances des applications. Les circuits de routage sont l'élément essentiel du coût du réseau, et leur coût ne doit pas être excessif comparé à celui des éléments de traitement. L'environnement de la machine est homogène, et certaines entorses aux standards sont acceptables. La topologie d'interconnexion peut être fixée, et l'ensemble de l'environnement est contrôlable par le système d'exploitation au niveau des noeuds de traitement. Les distances sont courtes, et on peut s'assurer que quelques cellules au plus soient en transit entre deux noeuds, jusqu'à des débits de plusieurs Gigabits par seconde.

Ces différences nous amènent à penser qu'il peut exister des solutions différentes pour l'implémentation de l'ATM à ces différents niveaux. En particulier, pour le réseau d'interconnexion d'une machine parallèle, il serait nécessaire de dégager des solutions moins coûteuses et plus performantes. Ces solutions devraient remplir les conditions suivantes:

- préserver les propriétés de l'ATM: maintient de l'ordre des cellules, possibilité de garantie de bande passante et de délai borné. Ceci est nécessaire pour permettre la compatibilité avec les couches d'adaptation à l'ATM (AAL) standard, et les applications multi-média ou temps réel. La remise en ordre des paquets constitue

par ailleurs un part importante des temps de traitement logiciels à l'émission et à la réception [39].

- ne pas introduire de propriétés nouvelles, non exploitables dans un réseau ATM (Ex : fiabilité du réseau).

- simplifier au maximum l'interface avec un réseau ATM standard.

On remarque que le problème de la compatibilité avec un réseau ATM standard revêt deux aspects :

- permettre l'échange de cellules avec un réseau standard : ceci demande simplement que le réseau fournisse un sous ensemble minimal des fonctionalités d'un réseau standard, et que les fonctionalités manquantes soient inutilisées, ou émulable facilement à l'interface (sans remonter au niveau AAL).

- permettre d'interfacer les réseaux d'interconnexion de deux machines par le biais d'un réseau standard, autrement dit, que le réseau standard puisse assurer le transport de cellules entre deux réseaux d'interconnexion. Ceci est possible en utilisant la notion de chemin virtuel, à condition que le transport des cellules ne suppose pas que les liens possèdent des propriétés qui ne soient pas assurées par le réseau standard, ou qui ne puissent pas être rétablies facilement à l'interface.

Le contexte du projet ILIAD nous a amenés à envisager la possibilité de concevoir des réseaux ATM autour de commutateurs du type de ceux généralement proposés pour les machines parallèles - et en particulier le Rcube. Ceci revient à proposer plusieurs voies de recherche pour l'implémentation de commutateurs ATM dédiés au calcul parallèle - la conception proprement dite d'un tel commutateur dépassant largement le cadre du projet.

En fait, l'amélioration des performances des réseaux ATM parait un objectif très ambitieux, en particulier si on se donne comme objectif d'atteindre des coûts comparables à ceux des circuits de routage utilisés dans les machines parallèles : la commutation ATM apporte en effet un degré de complexité suplémentaire. Il existe, de ce fait, un vaste ensemble de problèmes à résoudre, qui ne peuvent pas être traités en parallèle. Parmis ces problèmes, celui qui concerne le contrôle de flux est apparu fin 94 avec la définition des classes de services élastiques adaptées au transfert de données.

7.1 Les commutateurs

Le schéma général d'un commutateur ATM est donné par la figure 3:

- Les entrées ont pour rôle d'analyser les en-têtes de cellules pour déterminer la sortie sur laquelle elles doivent être réémises. En règle générale, on évite de mettre les messages en file d'attente au niveau des entrées, pour éviter le problème du blocage en tête de ligne : une cellule destinée à une sortie occupée bloque derrière elle des cellules qui pourraient être immédiatement transmises sur d'autres lignes. Ce problème a cependant été remis en question dans une publication récente [36]. L'analyse des données d'acheminement implique normalement une recherche dans une table à partir des identificateurs de connexions.

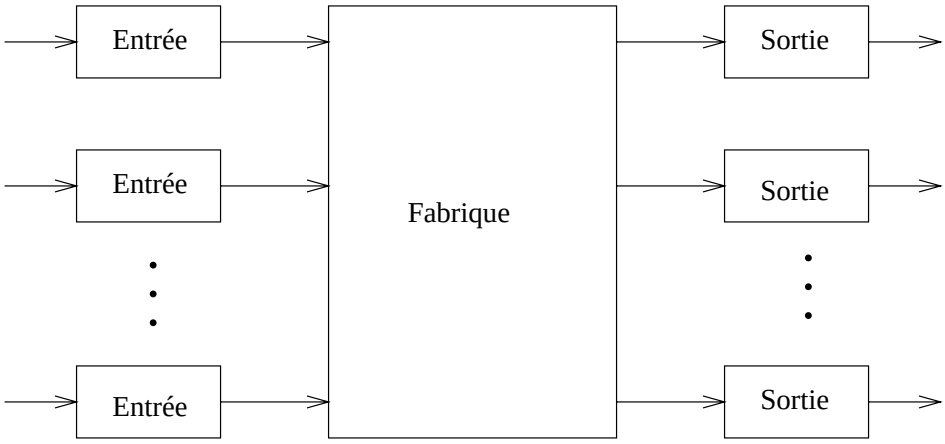


FIG. 3 – *Commutateur ATM*

- Les sorties ont pour rôle de réémettre les messages. Les files d'attente sont généralement placées à ce niveau. Ces files d'attente doivent permettre de traiter les messages de façon différente en fonction de leur classe de service.

- La fabrique a pour rôle d'acheminer les messages de l'entrée vers la sortie. Elle peut être implémentée de façons très diverses [40] : bus partagé, mémoire commune, cross-bar, réseaux multi-étages... Elle doit avoir des propriétés bien précises :

  - Une bande passante agrégée supérieure ou égale à la somme des bandes passantes des entrées,
  - Ne pas introduire de blocage entre deux messages provenant d'entrées et de sorties différentes (ou la probabilité d'un tel blocage doit être négligeable).
  - Respecter les propriétés générales de l'ATM (maintient de l'ordre des messages, garantie de bande passante et de délai d'acheminement borné...).

Sur la figure 3, on remarque qu'il existe en fait deux niveaux de réseaux : le réseau ATM proprement dit, formé par l'interconnexion des commutateurs, et le réseau interne au commutateur, constitué par la fabrique. A l'intérieur de la fabrique, les cellules sont généralement auto-acheminées, grâce à des informations de routage ajoutées en tête des cellules par les entrées.

Ceci nous suggère deux approches pour la réalisation d'un réseau ATM adapté au calcul parallèle : l'utilisation de petits commutateurs ultra-rapides, analogues aux routeurs utilisés dans les machines parallèles, ou l'interconnexion des éléments de traitement autour d'un seul gros commutateur ATM scalable, dont la fabrique peut être réalisée avec des routeurs classiques.

7.1.1 Commutateurs ultra rapides

La première solution va consister, par exemple, à réaliser des petits commutateurs 5Œ5 intégrables en VLSI destinés à réaliser des grilles 2D. Pour avoir des performances

proches de celles des réseaux d'interconnexion des machines parallèles, un tel commutateur doit fonctionner en worm-hole ou en virtual-cut-through, et avoir des latences de quelques dizaines de nanosecondes. Une structure de réseau analogue à celle de l'IBM SP1 peut être obtenue avec des commutateurs 8OE8 d'une latence d'une centaine de nanosecondes.

Pour cela, il est essentiel de simplifier le traitement des données d'acheminement par les entrées. La solution inévitable semble être de coder (au moins partiellement) la destination dans une partie des identificateurs de connexion. Il faut, par ailleurs, être capable de gérer des flots de données de priorités différentes au niveau des sorties [41].

7.1.2 Commutateurs scalables

La seconde solution consiste à reporter à la périphérie le problème de la commutation ATM, en se ramenant à des solutions de type réseau indirect multi-étages pour la fabrique. Ce réseau peut alors être composé à partir de routeurs tel que le Rcube, ce qui permettrait de bénéficier dans une large mesure des faibles latences permises par ce type de composant, avec les avantages de l'ATM. Si certains traitements spécifiques à l'ATM (traitement des en-têtes) sont reportés à la périphérie du réseau, celui-ci doit cependant respecter les contraintes d'un réseau ATM : garantie de bande passante et de délai borné pour certains flots de données, et réception des messages dans l'ordre de leur émission.

La réception des messages dans l'ordre de leur émission est probablement une des caractéristiques les plus fondamentales de l'ATM : les protocoles ATM ne comportent à priori aucun moyen de restaurer cet ordre s'il n'est pas garanti par le réseau, et cette restauration. lorsqu'elle est possible, représente une composante importante du temps de traitement des communications.

Cette contrainte ne peut, en principe, par être respectée dans une réseau utilisant le routage adpatatif, où des messages successifs peuvent suivre des chemins différents, de longueur différente : un message peut se voir bloqué sur un chemin, alors que le message suivant, qui aura pris un chemin différent, peut ne rencontrer aucun blocage, et arriver avant le premier message.

Avec le routage worm-hole, l'ordre peut cependant être préservé, si le réseau est tel qu'un message soit assuré d'arriver à destination avant d'avoir totalement quitté le noeud d'origine : c'est le cas si le volume de données stocké dans chaque noeud est suffisemment réduit (quelques octets), la longueur minimale des messages suffisemment longue, et la longueur maximale des chemin suffisemment courte. Avec des cellules ATM (53 octets), cette condition peut être vérifiée pour des routeurs stockant au plus quelques octets : pour quatre octets, le chemin parcouru pourra faire jusqu'à 14 noeuds, ce qui donne un assez large choix de topologies et de tailles de réseau. Malheureusement, cette condition ne peut pas être assurée avec le Rcube, dans lequel l'unité de contrôle de flux (flit) est de 32 octets.

Par ailleurs, dans les réseaux fonctionnant en worm-hole, la latence dépend assez directement du taux d'utilisation de la bande passante. Lors des pointes de charge du réseau, il est difficile de garantir un délai d'acheminement borné pour certaines classes de messages. L'utilisation de tels réseaux pour acheminer des cellules ATM se fait donc en perdant les propriétés permettant le transport de flots de données multi-média.

Il est probable qu'une large proportion des applications actuelles du calcul parallèle

n'ont pas besoin de ces propriétés. Dans ces conditions, la capacité de transporter des cellules ATM sur le réseau d'interconnexion d'une machine parallèle apporte une ouverture sur les réseaux locaux et à grande distance. Cependant, il est également probable que le développement du multi-média, du traitement d'images et du son, et de la synthèse d'images et de sons apportera le besoin de réseaux de communications possédant les propriétés de l'ATM.

8 CONCLUSIONS

Nos objectifs à l'origine du projet ILIAD étaient d'étudier l'interconnexion de machines du type M3S par un réseau ATM, et de définir les primitives d'un processeur de communication. Il s'agissait en fait d'une problématique assez vaste touchant à plusieurs domaines en pleine évolution, sur lesquels nous avons dû effectuer une recherche bibliographique conséquente :

- La communication par mémoire partagée : cohérence de caches et mémoire virtuelle distribuée. Un simulateur de mémoire virtuelle distribuée a été implémenté et fournira un moyen d'expérimentation sur la gestion de la cohérence et sur l'utilisation des réseaux.

- La communication par messages : PVM et MPI sont les deux interfaces de communication qui ont fait l'objet d'étude. Leurs modèles de communication sous-jacents sont assez proches, et peuvent fournir la base de primitives de communication à implémenter en matériel. Un simulateur de réseau ATM interfacé avec PVM a été implémenté, et fournira un outil pour des expérimentations ultérieures.

- Le calcul parallèle sur réseau ATM : les travaux effectués jusqu'ici montrent l'intérêt de solutions basées sur l'ATM pour le calcul parallèle, mais aussi l'importance d'une minimisation des latences logicielles. Nous pensons que seules des solutions matérielles permettent d'obtenir une amélioration significative sans compromis sur la sécurité.

- La commutation ATM : la commutation rapide et à haut débit fait l'objet de nombreux travaux de recherches, et il nous semble un peu ambitieux et, de plus, prématuré d'aborder ce domaine. L'étude d'implémentations de réseaux ATM adaptés au calcul parallèle peut être un objectif à plus long terme, mais il semble nécessaire, dans un premier temps, de se contenter des possibilités offertes par les implémentations disponibles.

- Le contrôle de flux : l'efficacité du contrôle de flux pour certaines applications de calcul parallèle nous semble devoir être vérifiée, et cela sera probablement un de nos axes de recherches prioritaires dans ce domaine par la suite.

L'évolution du domaine de recherche a remis en cause un certain nombre de nos objectifs, et a modifié profondément notre approche de certains problèmes. Ainsi, le problème de l'allocation de la bande passante a disparu avec la définition de la classe de service ABR. Par contre, le contrôle de flux, qui semblait opposé à la philosophie de l'ATM à l'origine est désormais partie intégrante du standard. Le problème du mode

connecté, qui nous semblait important à l'origine, a perdu une bonne part de cette importance après la publication de résultats de recherche montrant que les destinations des messages ont les propriétés de localité nécessaires à l'exploitation du mode connecté. Enfin, les délais des commutateurs ATM, s'ils sont supérieurs d'un à deux ordres de grandeur aux latences des réseaux d'interconnexion des machines parallèles, ne semblent pas pour autant être un problème à traiter en priorité : en effet, les temps de traitement logiciels restent la composante essentielle des latences d'application à application.

Par ailleurs, on voit se développer une vague de scepticisme vis à vis des machines massivement parallèles. Face à des architectures massivement parallèles telles que la CM5, la Paragon, ou la Cray T3D, on voit se développer l'idée de grappes de stations de travail interconnectées par des réseaux hautes performances pour permettre le calcul parallèle. L'intérêt de telles architectures est que les noeuds de traitement bénéficient du vaste marché des stations de travail ou des micro-ordinateurs, ce qui en fait une solution particulièrement économique. Deux approches sont possibles. La première fait appel à un réseau spécialisé, tel que celui de la SP2 d'IBM. Les réseaux Myrinet [42] et S-Connect [43] sont d'autres implémentations de la même approche, et d'autres solutions sont possibles, par exemple à partir du routeur Rcube développé par le MASI.

La seconde approche consiste à utiliser des technologies de réseaux, peut être moins performantes, mais plus standardisées. Cette approche peut apporter des solutions moins coûteuses, et, surtout, permet une connexion facile aux réseaux locaux, voire aux réseaux de télécommunication. L'ATM est un candidat pour une telle approche.

Deux problèmes importants nous semblent émerger, et devoir être traités en priorité. Ces problèmes ne sont apparus qu'en cours du projet, du fait de l'évolution de l'état de l'art, et ils n'ont donc pas pu être traités dans le cadre du projet.

Le premier problème concerne la réduction des latences logicielles de communication. Ceci peut passer par l'intégration d'un niveau de communication dans les couches basses du système d'exploitation, mais il nous semble que l'utilisation de matériel spécifique soit indispensable pour obtenir des performances élevées sans sacrifier les protections entre tâches. Ces travaux devraient se baser sur des interfaces de communication standard comme PVM ou MPI. La définition d'un interface de communication de bas niveau qui puisse être commun à plusieurs types de réseaux serait une avancée intéressante, en permettant de rendre les logiciels de plus haut niveau relativement indépendants du réseau physique.

Le second problème concerne le contrôle de flux. Il est nécessaire de vérifier l'efficacité des techniques préconisées par l'ATM Forum. A partir de là, il sera possible de définir les limites de l'utilisation de l'ATM pour le calcul parallèle, et de déterminer les axes de recherche prioritaires pour l'implémentation de réseaux ATM plus efficaces pour ce domaine d'application.

Références

[1] G. Pujolle, "Les réseaux large bande et l'ATM", *De nouvelles architectures pour les communications,* pp. 1-38, Eyrolles, 1994.

[2] R. J. Vetter, "ATM concepts, architectures, and protocols", *Communications of the ACM,* vol. 38, num. 2, pp. 30-38, février 1995.

[3] B. G. Kim, P. Wang, "ATM network : goals and challenges", *Communications of the ACM,* vol. 38, num. 2, pp. 39-44, février 1995.

[4] E. Biagioni, E.Cooper, R. Sansom, "Designing a practical ATM LAN", *IEEE Network,* pp. 32-39, mars 1993.

[5] K-Y. Siu, R. Jain, "A brief overview of ATM : protocol layers, LAN emulation, and traffic management", *ACM SIGCOMM, Computer Communication Review,* vol. 25, num. 2, pp. 6-20, avril 1995.

[6] H. L. Truong, W. W. Ellington Jr., J-Y. Le Boudec, A. X. Meier, J. WaynePace, "LAN emulation on an ATM network", *IEEE communications magazine,* vol. 33, num. 5, pp. 70-85, May 1995.

[7] M. W. Sachs, A. Leff, D. Sevigny, "LAN and I/O convergence : a survey of the issues", *IEEE Computer,* vol. 27, num. 12, pp. 24-32, decembre 1994.

[8] M. Hayter, D. McAuley, "The Desk Area Network", *ACM Operating System Review,* vol. 25, num. 4, octobre 1991.

[9] Mark David Hayter, *A workstation architecture to support multimedia,* Phd dissertation, St. John's College, University of Cambridge, September 1993.

[10] C. Huang, P. K. McKinley, *Communication Issues in parallel computing across ATM networks,* num. TR MSU-CPS-94-34, Michigan State University Communication Research Group, URL: ftp://ftp.cps.msu.edu/pub/crg/PAPERS/msu-cps-94-34.ps.Z, juin 1994.

[11] C. Huang, E.P. Kasten, P. McKinley, "Design and implementation of multicast operations for ATM-based high performance computing", *Supercomputing 94,* pp. 164-173, IEEE, Washington, November 1994.

[12] C. Huang, P.K. McKinley, "Design and implementation of global reduction operations across ATM networks", *3rd Int. symposium on high-performance distributed computing,* pp. 43-50, San Francisco, Aout 1994.

[13] M. Lin, J. Hsieh, D. H. C. Du, J. P. Thomas, J. A. MacDonald, "Distributed network computing over local ATM network", *Supercomputing 94,* pp. 154-163, IEEE, Washington, Aout 1994.

[14] J.J. Dongarra, G. A. Geist, R. Manchek, V. S. Sunderam, "Integrated PVM framework supports heterogeneous network computing", *Computers in physics,* vol. 7, num. 2, pp. 166-175, April 1993.

[15] G.A. Geist, J.A. Kohl, R.J. Manchek, P.M. Papadopoulos, "New features of PVM 3.4 and beyond", *EuroPVM'95,* J. Dongarra & al., ed., pp. 1-9, Hermes, Lyon, septembre 1995.

[16] C. A. Thekkath, H. M. Levy, E. D. Lasowska, *Efficient support for multicomputing on ATM networks,* num. TR 93-04-03, Department of Computer Science an Engirneering, University of Washington, Seattle, Avril 1993.

[17] T. Von Eicken, D. E. Culler, S. S. Goldstein, K. E. Schauser, "Active Messages : a mechanism for integrated communication and computation", *19th International conference onf computer architecture,* pp. 256-266, ACM, Gold Coast, mai 1992.

[18] T. Von Eicken, V. Avula, A. Basu, V. Buch, "Low-Latency communication over ATM networks using active messages", *Hot Interconnects II,* pp. 60-71, IEEE, Palo-Alto, aout 1994.

[19] M. Hausner, M. Burrows, C. Thekkath, "Efficient implementation of PVM on the AN2 ATM network", *High perofrmance computing and networking,* pp. 562-569, Milan, may 1995.

[20] T.E. Anderson, S. S. Owicki, J. B. Saxe, C. P. Thacker, *High speed switch scheduling for local area networks,* num. TR UCB/CSD-94.803, Unviversity of California, Berkeley, Computer science division, mars 1994.

[21] S. Dwarkadas, P. Keleher, A. L. Cox, W. Zwaenepoel, "Evaluation of release consistent software distributed shared memory on emerging network topology", *20th Int. Symposium on Computer Architecture,* pp. 144-155, ACM, San-Diego, Mai 1993.

[22] J. P. Singh, W. Weber, A. Gupta, "SPLASH : Stanford parallel applications for shared-memory", *ACM-SIGARCH - Computer Architecture News,* vol. 20, num. 1, pp. 5-43, mars 1992.

[23] T.E. Anderson, D.E. Culler, D.A. Patterson, "A Case for NOW (Networks of Workstations)", *IEEE Micro,* vol. 15, num. 1, pp. 54-64, Fevrier 1995.

[24] C. B. Stunkel, D. G. Shea, D. G. Grice, P. H. Hockschild, M Tsao, "The SP1 high-performance switch", *Scalable High Performance Computing,* pp. 150-157, Knoxville, mai 1994.

[25] J. Hsu, P. Banerjee, "Performance measurement and trace-driven simulation of parallel CAD and numeric applications on a hypercube multicomputer", *17th Int. Symposium on Computer Architecture,* pp. 260-269, IEEE, Seattle, mai 1990.

[26] J.M. Hsu, P. Banerjee, "A message-passing coprocessor for distributed memory multicomputers", *Supercomputing 90,* pp. 720-729, IEEE, New-York, November 1990.

[27] J-M. Hsu, P. Banerjee, "Hardware support for message routing in a distributed memory multicomputer", *International conference on parallel processing,* pp. I-508-515, août 1990.

[28] M.A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E.W. Felten, J. Sandberg, "Virtual memory mapped network interface for the SHRIMP multicomputer", *21st Int. Symposium on Computer Architecture,* pp. 142-153, ACM, Chicago, April 1994.

[29] C. Paetz, "Homogeneous integration of massive parallel computers in ATM networks", *5th Int. Conference on parallel computing (ParCo95),* Gent (Belgique), septembre 1995.

[30] R. Cypher, A. Ho, S. Konstantinidou, P. Messini, "Architectural requirements of parallel scientific applications with explicit communications", *20th Int. Symposium on Computer Architecture,* pp. 2-13, IEEE, San Diego, mai 1993.

[31] T. Horie, K. Hayashi, T. Shimizu, H. Ishihata, "Improving AP1000 parallel computer performance with message communication", *20th Int. symposium on computer architecture,* pp. 314-325, IEEE, San Diego, May 1993.

[32] Message Passing Interface Forum, *DRAFT, Document for a Standard Message Passing Interface,* URL: http://www.netlib.org/mpi/draft0220.ps, November 1993.

[33] G. Berger Sabbatel, *Présentation de MPI,* num. GRAM-RR1, IMAG-LGI, groupe de recherches en réseaux, architecture et validation de machines (GRAM), Grenoble, janvier 1994.

[34] H. Ohsaki, M. Murata, H. Suzuki, C. Ikeda, H. Miyahara, "Rate-Based congestion control for ATM networks", *ACM SIGCOMM, Computer Communication Review,* vol. 25, num. 2, pp. 60-72, avril 1995.

[35] H. T. Kung, T. Blackwell, A. Chapman, "Credit-based flow control for ATM networks: credit update protocol, adaptive credit allocation, and statistical multiplexing", *ACM-SIGCOMM'94 Symposium on communication architectures, protocols and applications,* pp. 101-114, ACM, Londres, août 1994.

[36] R. J. Simcoe, Tong-Bi Pei, "Perspectives on ATM switch architecture and the influence of traffic pattern assumptions on switch design", *ACM SIGCOMM, Computer Communication Review,* vol. 25, num. 2, pp. 93-105, avril 1995.

[37] D. Lenoski, J. Laudon, K. Gharachorloo, W-D. Weber, A. Gupta, J. Hennessy, M. Horowotz, M.S. Lam, "The Stanford DASH multiprocessor", *IEEE Computer,* vol. 25, num. 3, pp. 63-79, mars 1992.

[38] S. Ahuja, N. Carriero, D. Gelernter, "Linda and Friends", *IEEE Computer,* vol. 19, num. 8, pp. 26-34, août 1986.

[39] V. Karamcheti, A.A. Chien, "Software overhead in mesaging layers: where does the time go?", *6th Int. Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS),* pp. 51-60, ACM, San Jose, October 1994.

[40] R.Rooholamini , V.Cherkassky , M. Garver, "Finding the right ATM switch for the market", *IEEE Computer,* vol. 27, num. 4, pp. 16-28, April 1994.

[41] Y. Tamir, G.L. Frazier, "Hardware support for high-priority traffic in VLSI communication switches", *Journal of parallel and distributed computing,* vol. 14, pp. 402-416, April 1992.

[42] N. J. Boden, D. Cohen, R.E. Felderman, A. E Kulawik, C. L. Seitz, J., N. Seizovic, Wen-King Su, "Myrinet: a Gigabit-per-second local area network", *IEEE Micro,* vol. 15, num. 1, pp. 29-35, fevrier 1995.

[43] A. G. Nowatzyk, M. C. Browne, A. J. Kelly, M. Parkin, "S-Connect : from networks of workstations to supercomputer performance", *22nd Int. Conference on Computer Architecture,* pp. 71-82, IEEE, Santa Margherita Ligure, juin 1995.

ANNEXE 5
Rencontres Optique-Informatique (ROI)

Pascal Berthomé

26 février 1996

Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon
46, allée d'Italie
69 364 Lyon Cedex 07

Table des matières

1 Problématique

L'efficacité des machines parallèles peut se mesurer en terme d'adéquation entre plusieurs paramètres. Premièrement, il est indispensable que l'architecture du réseau d'interconnexion soit *faisable* au niveau VLSI. De plus, il est nécessaire de pouvoir décrire les *algorithmes* de manière *simple*, afin de pouvoir contrôler plus facilement leur correction. Enfin, le réseau d'interconnexion doit pouvoir supporter des *communications* globales *efficaces* : en effet, les communications représentent souvent le goulot d'étranglement des algorithmes parallèles.

Si on considère les diverses architectures parallèles déjà proposées, celles-ci réalisent souvent une ou deux de ces conditions, mais jamais les trois. Quelques solutions simples ont été avancées pour améliorer le modèle le plus utilisé: le réseau de type point-à-point. Il s'agit d'une part de trouver de nouveaux types de réseaux ayant de très bonnes propriétés de routage. Cette solution engendre des réseaux très difficiles à implanter, comme les *De Bruijn* ou les *star-graph*s. Une autre possibilité pour améliorer les communications générales est d'explorer les nouveaux modes de routage des messages dans les systèmes parallèles, comme le routage de type "wormhole".

L'utilisation de nouvelles technologies comme l'optique permet d'espérer réunir les trois critères de qualité pour un même système parallèle. En effet, il faut noter que, sous certaines conditions, les moyens optiques permettent de s'approcher des réseaux de type graphe complet : une machine pratiquement équivalente au modèle théorique des machines à mémoire partagée.

La technologie optique commence à être largement utilisée dans les grands réseaux, où de très hauts débits de communication sont nécessaires. Elle laisse donc entrevoir la possibilité d'avoir des communications très rapides au niveau d'une machine parallèle. Les principales limitations proviennent des interfaces optique/électronique. Les interactions entre deux faisceaux lumineux indépendants ne sont pas gênantes, contrairement à l'interaction entre deux fils électriques qui restreignent la densité effective des connexions pour les technologies classiques. On peut aussi noter que certains éléments optiques permettent de réaliser des diffusions d'information très simplement : on peut envoyer une information d'un processeur vers tous les autres en une seule étape. Il est d'ailleurs parfois réaliste de parler d'horloge globale avec les systèmes optiques.

En contrepartie à ces grands avantages technologiques, il faut noter le coût élevé des divers composants. Cela vient principalement du fait qu'ils sont pour la plupart encore expérimentaux, et la recherche progressant, ce coût devrait devenir rapidement raisonnable. Un des axes de recherche est aussi la miniaturisation des éléments de base de l'optique.

La problématique nouvelle qu'offre l'optique se situe dans un premier temps dans l'étude de nouveaux types de réseaux d'interconnexion. Il reste aussi à savoir utiliser l'énorme potentiel de communication.

Il est donc indispensable pour les deux communautés, optique et informatique, d'unir leurs efforts pour bien définir les possibilités réalistes en terme d'optique en vue de définir, puis à plus long terme réaliser une machine parallèle dont les interconnexions seraient optiques.

C'est dans ce cadre que le groupe de travail des **Rencontres Optique - Informatique (ROI)** s'est constitué. Par la suite, nous présentons les diverses équipes participant à ce groupe, les divers buts du groupe ainsi que les modalités de ses activités.

2 Les équipes participantes

Le groupe de travail ROI s'est constitué petit à petit au cours de l'année 1994/95. Il est issu de différents projets français et internationaux. De plus, différents laboratoires ont participé à des projets franco-israéliens sur divers thèmes tournant autour des interconnexions optiques. D'autre part, des rencontres ont aussi eu lieu par l'intermédiaire du groupe *Rumeur* du PRC GDR PRS et de l'action inter-PRC *ILIAD* où des études sur l'optique dans les machines parallèles ont fait l'objet de discussions.

La responsabilité de ROI est assurée par Pascal Berthomé, AMN au LIP-ENS-Lyon, et pas les co-responsables scientifiques Pierre Chavel, DR au CNRS, IOTA à Orsay, et Afonso Ferreira, CR au CNRS, LIP- ENS-Lyon. ROI est constitué des équipes suivantes.

2.1 Institut d'Optique Théorique et Appliquée

(URA 14, 25 chercheurs, 15 enseignants-chercheurs, 44 ITA, 60 doctorants et stagiaires)

Au sein de l'Institut d'Optique, le groupe de Physique des Images maintient depuis dix ans une activite d'exploration des possibilites de l'informatique en mettant l'accent sur la définition et la démonstration de systèmes. Les problématiques essentielles sont actuellement :

- la définition d'architectures largement parallèles à grain très fin menant à des processeurs dédies sans équivalent purement électronique ; exemple : parallélisation du recuit simulé pour l'application au traitement d'images bas niveau ;
- la réalisation de commutations tout optiques à partir de principes de traitement qui pourraient devenir pertinents en télécommunications et en informatique parallèle ;
- la conception et la réalisation de composants micro-optiques. Cet effort technologique constitue en fait une inflexion récente et importante : sa nécessité découle du constat que les réalisations opto-informatiques sont actuellement fortement handicapées par le manque de procédés fiables et bon marché d'intégration des systèmes au moins autant que par la disponibilité des composants optoélectroniques et optiques non linéaires, qui a brillamment progressé depuis environ 5 ans.

Responsable ROI : Pierre Chavel (DR CNRS)
Participants : Nicole de Beaucoudrey (CR CNRS)
Philippe Lalanne

2.2 Laboatoire de l'Informatique du Parallélisme

(URA 1398, 24 chercheurs permanents, 36 doctorants)

Les communications optiques ont pris leur importance au sein du Laboratoire de l'Informatique du Parallélisme (LIP) au travers de l'équipe Communication, Réseaux, Algorithmes et Complexité (CRAC).

Ce groupe s'est particulièrement intéressé aux différentes méthodes de routage optique, comme l'utilisation du multiplexage en fréquence et de l'optique libre. Divers

modèles optiques ont été proposés, en collaboration avec les autres équipes de ROI. Les stratégies de communicaions globales ont été étudiées, ainsi que diverses stratégies d'allocation de ressource pour des réseaux locaux ou métropolitains.

Le groupe a accueilli pendant quelques mois des chercheurs étrangers.

Responsable ROI: Pascal Berthomé (AMN)
Participants: Afonso Ferreira (CR CNRS)
Paraskevi Fragopoulou (Postdoc)
Ahmed Louri (Professeur invité)
Nashib Qadri (Mastere Canadien)

2.3 I3S

Responsable ROI: Michel Syska (MC)
Participants: Bruno Pineda (DEA)

2.4 Laboratoire de Recherche en Informatique

Le *Laboratoire de Recherche en Informatique* a été co-fondé en 1974 par le CNRS et l'Université de Paris-Sud. Le laboratoire comprend 186 personnes dont 89 chercheurs, 19 techniciens et personnels administratifs et 82 étudiants en thèse. Il est organisé en 9 équipes impliquées dans différents thèmes de recherche.

L'équipe *Graphes et Combinatoire*, dirigée par Dominique Sotteau (directeur de recherche CNRS), est constituée de 16 chercheurs en poste et 14 étudiants en thèse. Dans cette équipe, le groupe de travail *graphes et réseaux d'interconnexion* s'intéresse en particulier à des problèmes de communications dans des réseaux de processeurs. Les réseaux à bus, modélisés par des hypergraphes, ainsi que divers autres tels que le "Hot Potato", trouvent déjà leurs applications dans la conception d'architectures optiques. Par ses compétences, ce groupe trouve naturellement sa place au sein des activités de ROI. Les réunions et les échanges qui ont eu lieu dans le cadre de ROI ont déjà permis de cibler différents problèmes de communications optiques, dont l'étude est en cours.

L'intérêt du groupe pour ROI est donc multiple: acquérir des connaissances dans le domaine de l'optique pour affiner nos modèles, et apporter notre expérience dans la manipulation des concepts d'informatique théorique utilisés.

Responsable ROI: Dominique Barth (MC)
Participants: Dominique Sotteau (DR CNRS)
Jean-Claude König (Professeur)
Christian Laforest (Étudiant en thèse)

2.5 Georgiatech-Lorraine, projet de laboratoire LOADS

(Laboratory for Optics in Advanced Digital Systems)

Georgia Tech a installe en 1990 une antenne en Lorraine, sur la technopole de Metz. L'établissement délivre des maîtrises (MSc) qui reposent en partie sur des collaborations avec des établissements européens. Il souhaite maintenant s'étoffer de laboratoires de recherche et a choisi comme premier thème l'optique dans les systèmes digitaux avances. T.J. Drabik, Assistant Professor, s'installe donc a Metz pour y créer une équipe autour

d'un laboratoire de systemes et de caractérisation de composants optoélectroniques. Il souhaite vivement participer à des collaborations dans le milieu de la recherche française et s'est donc joint dernièrement à ROI, dont les thèmes correspondent parfaitement à ses objectifs.

Responsable ROI : Thimothy Drabik

2.6 Partenaires envisagés

Au cours des diverses réunions, il s'est avéré intéressant d'intégrer dans un futur assez proche d'autres compétences à notre groupe.

IEF Orsay

CNET, Lannion

3 Objectifs

3.1 Étude d'une machine parallèle à interconnexions optiques

Le but principal de ce groupe de travail est de définir en détail une machine parallèle dont les interconnexions seraient optiques. À terme, sous réserve de trouver les partenaires nécessaires au niveau industriel, la réalisation d'un prototype est envisagée.

Il sera important de réaliser des études théoriques de complexité des circuits optiques. Les termes de complexité devront être explicités : coût de fabrication, extensibilité, ainsi que les coûts classiques des réseaux, comme les paramètres des structures sous-jacentes (diamètre, connectivité).

3.2 Établissement d'un langage commun-échange de compétences.

Les deux communautés cohabitent pour l'instant chacune de leur côté, avec leur langage particulier. En regardant de près, on s'aperçoit que divers problèmes ont déjà été traités par les uns ou les autres, mais sous des terminologies très diverses et des problématiques différentes. Il est donc indispensable de parler le même langage, d'échanger les bases bibliographiques afin de progresser dans nos recherches respectives.

3.3 Autres thèmes

D'autres thèmes issus des télécommunications pourraient être abordés. Il s'agit de problèmes d'optimisation combinatoire.

4 Fonctionnement

4.1 Les réunions

Pour l'instant, les échanges entre les deux communautés se sont principalement réalisés autour de réunions. Celles-ci se sont déroulées sur les deux sites centraux de Lyon et d'Orsay à raison d'une tous les deux mois environ.

Cette fréquence de réunions semble satisfaire toutes les parties, et devrait être maintenue l'an prochain.

Une journée ROI est organisée en deux demi-journées. Dans un premier temps, nous essayons de compléter nos cultures générales respectives en optique et informatique par des exposés assez généraux. L'autre demi-journée est alors disponible pour des discussions ouvertes sur les modèles, les capacités des composants, et sur les diverses recherches en cours.

Depuis le mois de septembre 1994, le groupe ROI s'est réuni 6 fois. Les principaux exposés étaient :

Date	Lieu	Programme
26/09/94	IOTA	*P. Berthomé*: "Présentation du modèle Optical Passive Star" *P. Chavel*: "Principes de l'interconnexion optique en espace libre et composants optoélectroniques utilisables"
22/11/94	LIP	*N. Qadri*: "Allocateur de ressource pour modèle multiwavelength" *P. Chavel*: "Démonstration de cube d'interconnexions optiques par voie acousto-optique"
21/02/95	IOTA	*P. Chavel*: "Transformée de Fourier optique" *P. Berthomé*: "Vers un modèle optique pour les informaticiens"
18/04/95	LIP	*P. Chavel*: "Principe de l'ordinateur tout-optique de Guilfoyle" *A. Ferreira*: "Hypertopologies, performances et implantantions"
21/06/95	LRI	*T. Drabik*: "Optoelectronic and optomechanical integration for high performance computing" *P. Chavel*: "Automates cellulairs optiques" *J.-C. König*: "Hypergraphes"
14/09/95	LIP	*A. Louri*: "Role of optics in high performance computing" *C. Laforest*: "Routage hot-potato et bus"

4.2 Publications

Quelques publications des chercheurs sont au moins en partie issues des recherches communes. On peut citer en particulier :

- Pascal Berthomé. *Contribution a l'algorithmique des architectures parallèles : des réseaux point-à-point aux réseaux optiques*, Thèse de doctorat, Janvier 1995.

  La deuxième partie de cette thèse traite des problèmes de communication dans les machines à interconnexions optiques. Divers modèles sont présentés, ainsi que des techniques de communication générale. Enfin, les propriétés d'auto-simulation ont aussi été abordées. Pierre Chavel a été membre du jury.

- Pascal Berthomé, Pierre Chavel. *Opto-Informatique*, Chapitre de "Parallélisme et Applications Irrégulières", 1995, Hermes.

  Pierre Chavel et Pascal Berthomé sont intervenus au cours de l'École d'hiver du groupe CAPA à Cauterets. Il s'agissait alors de faire connaître les problématiques liées à l'optique à un public d'informaticiens. Cette intervention a fait l'objet de la rédaction de ce chapitre d'ouvrage. Dans ce chapitre, les apports de l'optique pour les machines parallèles sont exposés, ainsi que diverses expériences menées

au laboratoire IOTA. On présente aussi une étude "plus théorique" des modèles optiques par le biais du problème d'auto-simulation.

– Bruno Pineda. stage de DEA, Juin 1995.

  L'objet de ce stage était de montrer la faisabilité des interconnexions optiques dans le cadre particulier des réseaux multi-étages.

5 Conclusion

5.1 Bilan

Cette année de travail commun a permis de constituer un axe de recherche commun et interdisciplinaire entre deux communautés qui n'ont pratiquement jamais collaboré auparavant. Les informaticiens et les opticiens se sont souvent intéressés aux mêmes problèmes d'efficacité dans les communications.

L'acitivité de ce groupe en plein essor a été marqué par quelques publications résultant d'un travail commun, et l'apport de chercheurs extérieurs, comme professeur invité et chercheur post-doctoral.

Cette collaboration a scellé un but commun de définition d'une machine parallèle dont les interconnexions sont optiques.

5.2 Perspectives

Les perspectives sont de plusieurs ordres. Il s'agit en premier lieu de continuer les rencontres sous leur forme actuelle, afin de faire progresser les réalisations communes.

Il s'agit aussi d'intégrer des compétences nouvelles, indispensables à la réalisation de notre projet. On pense particulièrement aux électroniciens pour ce qui concerne les interfaces optiques/électroniques. Il serait aussi intéressant de rechercher un partenariat industriel afin d'obtenir un soutien logistique pour les expériences nécessaires à la validation des modèles. Enfin, il est nécessaire de développer une thèse en optique sur les différents thèmes induits par cette recherche coordonnée.

Un des projets du groupe est aussi de réaliser un colloque sur les interconnexions optiques, qui pourrait drainer des personnalités importantes de ce domaine neuf, en plein essor, qu'est l'opto-informatique. Ce colloque aura lieu sur le site de GeorgiaTech-Lorraine au mois de décembre de cette année.

Les équipes informatiques de ROI participent au projet européen Capital Humain et Mobilité (HCM) MAP. Dans la proposition de renouvellement de ce contrat, il a été inséré un axe optique avec des collaborations avec d'autres centres comme Weizmann Institute en Israël.

ANNEXE 6
A low latency adaptive message router
Un routeur de message adaptatif à faible latence

B. Zerrouk
V. Reibaldi
F. Potter
A. Greiner
A. Derieux

26 février 1996

MASI, équipe CAO et VLSI

Recent developments in parallel computers make the need for high performance communication network real. Our contribution in this field is the design of hardware communication primitives for such systems. This includes both processor/router interfacing and routing hardware. In this paper, we focus on the presentation of a new routing device, called RCube (Rapid Reconfigurable Router).

RCube is based on high speed CMOS serial links and provides simple but efficient support for adaptivity. It implements a worm-hole flow-control technique, interval and prefix routing schemes.

Les développements récents des calculateurs parallèles entraînent le besoin de réseaux de communication à hautes performances. Notre contribution dans ce domaine est la conception de primitives de communication matérielles pour de tels systèmes. Ceci comprend à la fois l'interface entre processeurs et routeurs, et le matériel de routage. Dans ce papier, nous présentons plus particulièrement un nouveau circuit de routage, le RCube (Rapid Reconfigurable Router).

Le RCube est basé sur des liens série rapides CMOS et supporte l'adaptativité de manère simple, mais efficace. It implémente une technique de contrôle de flux de type worm-hole, et le routage préfixé par intervalles.