Évaluation de performances et spécification formelle sur un réseau de stations haut débit

Marc Herbert

15 mai — 15 décembre 1996

Mastère INT Système Répartis en Informatique et Télécommunications

Encadrement : Bernard Tourancheau

Laboratoire pour les Hautes Performances en Calcul — ENS Lyon

Projet INRIA ReMaP

Institut National des Télécommunications

LIGIM, UCB Lyon1, 43 Bd du 11 Novembre 1918, 69622 Villeurbanne cedex, FRANCE

Résumé

L'évolution naturelle du calcul intensif va des coûteuses machines dédiées vers les réseaux de stations de travail banalisées (« *of the shelves* », « sur étagères »). Ce qui gênait encore cette évolution est le goulet d'étranglement du réseau de communication. Avec l'arrivée de nouveaux matériels réseau haut débit et surtout faible latence comme Myrinet, ce potentiel peut enfin s'exprimer. Cependant, les logiciels prêts à utiliser pleinement de telles performances sont pour l'instant peu courants. C'est pourquoi l'équipe RHDAC a développé de nouvelles librairies de communication. Dans ce cadre, j'ai pris en charge dans un premier temps des évaluations de performances. Puis j'ai montré ce que pouvait apporter la validation formelle des logiciels conçus, en recherchant des outils théoriques adéquats, et en testant sur un modèle formel de nos logiciels une boîte à outils de l'INRIA basée sur le langage LOTOS

Mots-clé : Réseau haut débit, Myrinet, Réseau de stations, Mesure de performance, Méthodes formelles, LOTOS, CADP

Abstract

The high performance computing begins to leave the field of high cost, special designed powerful machines, to go to “of the shelves” commodity clusters of workstations. The performance of the network was previously the bottleneck, but new high-speed low-latency material like Myrinet solve the harware problem. But software is not yet ready, that's why the RHDAC team implement new communication librairies. In this context, I ran several benchmarks on our platform. Then I searched for validation theory and tools. A formal model of our software was used as a test-case with the LOTOS toolbox of the INRIA.

Keywords : High-speed network, Myrinet, Clusters of workstations, Benchmarks, Formal methods, LOTOS, CADP

Remerciements

Je souhaite remercier Christian Bac, qui a bien voulu être mon directeur d'études pour cette thèse professionnelle, Loïc Prylli, pour sa disponibilité et sa gentillesse, et tout spécialement Bernard Tourancheau, qui m'a fait confiance en me confiant en particulier des tâches théoriques difficiles.

Sommaire

Table des figures

Liste des tableaux

Introduction

J'ai effectué mon stage au sein de l'équipe RHDAC, à Lyon, une équipe appartenant à plusieurs structures présentées dans la première partie. Lors du début du stage, une des principales activités de l'équipe était la mesure des performances de la plate-forme expérimentale Myrinet. Ayant besoin de me familiariser avec la complexité technique de celle-ci et les logiciels développés, j'ai pris en charge une partie des *benchmarks* [1]. Les résultats de deux de ceux-ci sont exposés dans la deuxième partie. Ce travail a consisté en l'installation et la compilation de codes de calcul, la recherche des meilleurs paramètres, et le retour d'information vers les concepteurs de BIP et de MPI-BIP, afin de discuter et d'éprouver les choix techniques faits par ceux-ci. Les performances globales de la plate-forme, incluant d'autres benchs, sont disponibles sur le site web de l'équipe [2]. Un article sur les performances est à paraître dans la revue *Calculateurs Parallèles* [13].

Dans le cadre de la participation à la gestion courante de la plate-forme, j'ai ensuite entrepris une rénovation du site web, et une part active à l'administration courante de la plate-forme, notamment pendant la période d'absence de l'administrateur principal. Cela constitue à mon avis une excellente formation par la pratique, complément de la formation reçue à l'INT.

Le développement de nouveaux logiciels et protocoles de communication est très complexe. Il est très difficile, surtout en phase expérimentale et de recherche de performances, d'avoir une vue d'ensemble du fonctionnement d'une pile de communication, et d'éviter les erreurs, dont certaines peuvent être très subtiles et trouvées tard, bien après la diffusion du produit. C'est dans ce cadre que s'est inscrit le travail réalisé pendant la principale partie du stage, et présenté dans la partie 3 : rechercher les notions théoriques et les outils de vérification formelle susceptibles d'être utiles au développement des logiciels de communication de la plate-forme expérimentale, et évaluer concrètement leur intérêt.

L'INRIA Rhône-Alpes développe actuellement au sein du projet VASY[1], conjointement avec Bull, une boîte à outils de vérification de protocoles, s'appuyant sur les méthodes formelles. Cette boîte à outils CADP est basée sur le langage de spécification LOTOS, normalisé par l'ISO, Cette boîte à outils a été utilisée pour la modélisation de protocoles développés au sein de l'équipe.

1. bancs de mesure de performances

Chapitre 1

Les structures d'accueil

J'ai effectué mon stage au sein de l'équipe « Réseaux Haut Débit et Applications Coopératives » (RHDAC), équipe faisant partie de plusieurs laboratoires de recherche, aussi bien publics que privés. Ce chapitre présente les différents partenaires et l'équipe qui m'a accueilli.

1.1 Le Laboratoire pour les Hautes Performances en Calcul

Le LHPC [5] est un laboratoire commun de recherche sur les ordinateurs massivement parallèles et le calcul à haute performance créé entre le LIP, le CNRS, L'INRIA Rhône-Alpes, et la société Matra Systèmes et Information qui conçoit et commercialise des systèmes parallèles.

Installé à Lyon, au sein de l'École Normale Supérieure, le LHPC répond à une démarche originale : celle d'un laboratoire commun, où des équipes industrielles sont intégrées dans un laboratoire de recherche public pour profiter au maximum de l'environnement scientifique et de l'ouverture intellectuelle que procure ce dernier.

L'activité principale du LHPC est d'être un centre de compétences dans le domaine du calcul massivement parallèle et de réunir des partenaires d'origines variées autour d'un projet académique et industriel aux ambitions mondiales.

1.1.1 L'École Normale Supérieure de Lyon

L'École Normale Supérieure (ENS) de Lyon accueille près d'un millier d'étudiants, qui sont intégrés après les classes préparatoires. Ses quatre cents chercheurs, soixante auxiliaires et quarante visiteurs étrangers se répartissent sur une douzaine de laboratoires, et sont compétents dans des domaines aussi divers que l'informatique, les mathématiques, la physique, la biologie ou encore les sciences de la terre et de l'univers.

1.1.2 Le Laboratoire d'Informatique du Parallélisme

Le Laboratoire de l'Informatique du Parallélisme (LIP) est le laboratoire de recherche en informatique de l'École Normale Supérieure de Lyon (ENS Lyon), et est également une Unité Associée du Département des Sciences Physiques pour l'Ingénieur (SPI) du CNRS. Avec plus de 70 chercheurs et un laboratoire équipé avec un matériel informatique de pointe, c'est un des centres de recherche en calcul parallèle parmi les plus avancés en Europe.

Les activités du LIP sont dédiées à l'étude du parallélisme mais couvrent la plupart des domaines traditionnels de l'informatique et de ses applications. L'existence d'un thème commun qui recouvre de nombreux axes de recherche (algorithmes et architectures, langages et systèmes, modèles), permet le développement de synergies entre des chercheurs possédant des cultures et des orientations différentes. Plusieurs projets horizontaux (programmation d'ordinateurs massivement parallèles, super calculateurs, architectures matérielles et logicielles dédiées, environnements de programmation) contribuent à favoriser la cohésion du laboratoire. Le LIP est aussi engagé dans un grand nombre de projets de Recherche et Développement en collaboration avec l'industrie, le gouvernement et les laboratoires universitaires.

Une partie des recherches effectuées au sein du LIP est valorisée dans le cadre des activités du LHPC.

1.2 L'INRIA Rhône-Alpes

L'Institut National de Recherche en Informatique et Automatique est un établissement public à caractère scientifique et technologique placé sous la tutelle des ministres chargés de la recherche et de l'industrie. L'INRIA est réparti géographiquement sur 5 sites – dont une unité « Rhône-Alpes » à Grenoble – et organisé en petites entités autonomes de 15 à 20 personnes. 4 thèmes majeurs structurent de cette manière 80 projets de recherche. En incluant les près de 600 doctorants, cette structure rassemble ainsi 1700 scientifiques pour un effectif de 2100 personnes.

1.2.1 Le projet ReMaP

L'objectif du projet INRIA ReMaP (Régularité et Parallélisme Massif) est de contribuer à l'élaboration des connaissances dans le domaine du calcul massivement parallèle régulier. La classe des machines visée concerne les architectures massivement parallèles à mémoire distribuée (ou plus généralement tout environnement de calcul multi-sites décentralisé comme un réseau de stations de travail). La classe des applications visée est celle où la régularité des données et des calculs est la caractéristique principale. De façon interne au projet, les applications ciblées sont l'algèbre linéaire dense et l'imagerie volumique.

ReMaP est un projet abrité principalement par le LIP.

1.3 Le Laboratoire d'Informatique Graphique Image et Modélisation

Le LIGIM est un laboratoire de l'université Claude Bernard Lyon 1, qui propose, à côté de son activité d'image et modélisation, une formation réseau. À ce titre, il emploie Bernard Tourancheau comme professeur, et accueille l'équipe RHDAC dans ses locaux. Le domaine de l'image est naturellement demandeur de fortes puissances de calcul. C'est pourquoi la parallélisation de ses applications et l'utilisation de la plate-forme du RHDAC pour les effectuer sont en cours de réalisation.

1.3.1 L'équipe Réseau Haut Débit et Applications Coopératives

Au sein du LIGIM, Bernard Tourancheau a mis en place une petite équipe de recherche sur les réseaux haut débit et les applications coopératives. Cette équipe est composée de :

- un professeur : Bernard Tourancheau
- un maître de conférences : Laurent Lefèvre
- un doctorant : Loïc Prylli
- des stagiaires en formation ingénieur et DEA

... et appartient aux projets LHPC et INRIA ReMaP.

Chapitre 2

Évaluation de la plate-forme Myrinet

2.1 Les réseaux locaux

La décennie 90 sera celle des télécommunications. Devenues aussi indispensables que l'informatique au monde moderne, les télécommunications ont pourtant connu des progrès techniques moins rapides : alors que la puissance des ordinateurs doublait régulièrement tous les deux ans, il fallut près de 20 ans aux réseaux pour passer de quelques kilo-bits par seconde à une centaine de kilo-bits par seconde. Mais d'ici à la fin du siècle, la situation risque de s'inverser. Alors que les progrès de l'informatique semblent se ralentir, ceux des télécommunications connaissent une accélération sans précédent. Cette progression spectaculaire s'appuie sur des avancées technologiques décisives dans deux domaines : celui des supports physiques, d'une part (fibre optique, composants hyper fréquences...), et celui des supports logiques d'autre part (architectures et protocoles nécessaires pour maîtriser de tels débits...).

Dans ce contexte, apparaissent de nouvelles technologies approchant ou atteignant le gigabit/s, telles que l'ATM, le *Gigabit Ethernet* et de nombreux autres projets, dont Myrinet.

2.2 Évolution du calcul parallèle

Dans le même temps, la puissance croissante de microprocesseurs bon marché, permet de construire des réseaux de stations, plus communément appelés *clusters* ou grappes de stations, qui rivalisent maintenant avec des calculateurs dédiés.

Les nombreux avantages de cette architecture (faible coût par unité, unité indépendante facilement remplaçable, extensibilité) sont grevés par des limitations matérielles (réseau d'interconnexion mal adapté : latence élevée et bande passante trop faible) ou logicielles (notamment, absence de logiciels permettant de traiter un cluster comme un simple système).

Mais, l'évolution rapide des technologies réseaux d'une part, et les recherches conduites par les universitaires et les laboratoires privés d'autre part, sont en train de résoudre ces problèmes. C'est dans ce contexte que nous présentons notre réseau Myrinet de machines de type « PC ».

2.3 Présentation du réseau local Myrinet

2.3.1 Genèse de Myrinet

Myrinet est un nouveau type de réseau local basé sur les communications et la commutation utilisées pour les machines massivement parallèles.

Myrinet est né de deux projets de recherche : Caltech Mosaic C Multicomputer [20] et USC/ISI ATOMIC LAN [10]. Cette technologie est commercialisée par la jeune société californienne Myricom Inc. [3], fondée en avril 1994 par les membres de ces deux équipes.

Caltech Mosaic était un projet expérimental développé à l'Université Technologique de Californie. Les machines massivement parallèles sont constituées de plusieurs centaines de nœuds de calcul évolués interconnectés à un réseau de communication simple. L'idée de Caltech Mosaic était de créer des machines composées de dizaines de milliers de nœuds de calcul simples connectés à un réseau très performant [19]. Les objectifs de Caltech Mosaic ont été atteints, et le réseau construit possède les caractéristiques espérées (bande passante élevée, taux d'erreur négligeable, routage optimisé...).

En 1991, un groupe de recherche de l'USC/ISI a commencé à développer un réseau local (LAN) à haut débit (ATOMIC LAN) en utilisant les composants de Caltech Mosaic. Les premiers essais furent concluants (bande passante de 400 Mbits/s), mais montra quelques restrictions (faible distance, performances limitées car n'utilisant pas le DMA.

La conception de Myrinet est directement basée sur les expériences d'ATOMIC LAN. Myricom a développé de nouveaux composants pour remédier aux limites de ces projets.

2.3.2 Descriptif

Myrinet est une technologie réseau issue des machines parallèles, basée sur des liens de communication point à point reliant entre eux les commutateurs et les nœuds finaux du réseau. Cette technologie permet d'obtenir des débits de 1,28 Gbits/s sur chaque lien, tout en garantissant des latences très faibles. Les contrôles de flux et d'erreur sur les liens assurent leur fiabilité. Les commutateurs possèdent une bande passante agrégée suffisante pour supporter simultanément un débit maximal sur tous les liens entrants et sortants.

Myrinet est décliné en version réseau local courte distance, (longueur maximale d'un câble : 10 m), ou réseau *système* SAN de très courte distance (câble < 3 m). Des répéteurs sont disponibles. Des interfaces en fibre optique portant ces distances à 500 m sont à l'étude, incluant des mécanismes d'adaptation automatique du débit en fonction du taux d'erreur.

2.3.3 les composants matériels et logiciels

La technologie Myrinet s'appuie sur des composants matériels standards :

– Câble électrique : il est formé de 18 paires torsadées blindées (9 pour

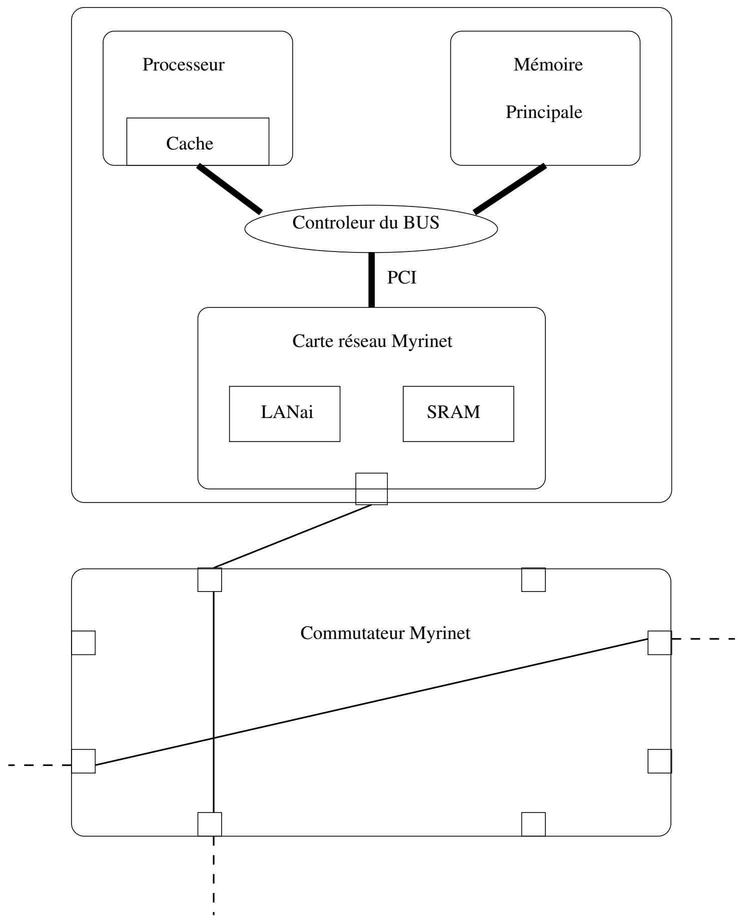


FIG. 2.1 – *Architecture Myrinet*

chaque direction, les liens sont bi-directionnels « full duplex »).

- Commutateur Myrinet : Myricom fournit des commutateurs ayant 4, 8, 16 ou 32 ports. Ils reposent sur deux puces VLSI : la première assure la commutation (crossbar) et la seconde gère le protocole et le contrôle de flot (niveau 4).
- La carte d'interface réseau : elle possède 1 port permettant la connexion entre l'ordinateur hôte et le réseau. Elle intègre un processeur et une mémoire SRAM (de 128 Ko à 256 Ko). Cette mémoire sert de tampon pour la gestion des communications ainsi que pour contenir le programme de contrôle MCP.. L'ensemble formé par le processeur, la connexion au réseau et le mécanisme de DMA est une puce VLSI appelé LANai.

et sur des logiciels :

- Le programme de contrôle MCP : ce programme est chargé initialement au démarrage de la machine hôte et s'exécute dès que le pilote de la carte en reçoit l'ordre. Il assure la réception et l'émission des paquets, gère les files d'attente et les acquittements. Il doit contrôler le DMA, déterminer le routage, vérifier l'interface. Il a également en charge les fonctions de *mapping* et *monitoring* du réseau ce qui rend le réseau auto-reconfigurable.

- Logiciels utilisateurs : Myricom fournit à la fois les outils nécessaire au développement d'applications TCP/IP et UDP/IP standards, mais également une API spécifique.

2.4 Présentation de la plate-forme expérimentale

2.4.1 Le matériel

La plate forme actuelle est centrée autour d'un commutateur 8 ports (4 ports SAN et 4 ports LAN) auxquels sont connectées :

- 4 stations à base de Pentium Pro 200 MHz avec 64 Mo ou 128 Mo de mémoire vive
- 2 stations à base de Pentium 133 MHz avec 64 Mo de mémoire vive
- une station DEC Alpha 500 MHz actuellement en prêt (l'achat de 2 machines de ce type est envisagé)

Ces stations comprennent chacune 1 ou 2 cartes d'interface Myrinet/PCI 256 Ko. Toutes possèdent également une carte Ethernet 10 Mbits/s standard. Les communications sont assurées par deux réseaux :

- Réseau principal à base d'Ethernet à 10 Mbits/s : il gère le trafic IP à destination des adresses standards lorsque celui-ci ne passe pas par Myrinet. Il est utilisé comme sur n'importe quel réseau de stations classique.
- Réseau secondaire sous Myrinet : il est utilisé pour les applications réparties, pour le développement des nouveaux logiciels, et pour les tests de performances. Deux machines possèdent une carte supplémentaire, ce qui augmente la souplesse lors des développements et des expérimentations.

2.4.2 Le logiciel

Le système d'exploitation est Linux 2.0 sous la distribution Debian 1.2. L'utilisation d'un système libre était quasiment inévitable, les développements ayant nécessité la modification du noyau du système. Les logiciels et interfaces de communication installés sont :

- MyriApi, la bibliothèque de communication bas niveau développée par la société Myricom. Myricom fournit également avec son matériel une interface pour IP sur de nombreuses plates-formes différentes.
- BIP *Messages*, l'API conçue et réalisée par l'équipe RHDAC. Cette bibliothèque bas niveau, réécriture de celle de Myricom, inclut un nouveau MCP. Elle permet de délivrer 1 gigabit/s et un temps de latence inférieur à 5 μs à l'utilisateur[18][17].
- Une interface BIP-IP pour la pile IP de Linux, permettant d'utiliser les nombreuses applications IP existantes avec de très hautes performances.
- Des bibliothèques de communication standard de plus haut niveau : PVM 3.3.11, LAM MPI 6.1, La bibliothèque standard des Sockets Unix, Fast Messages,...

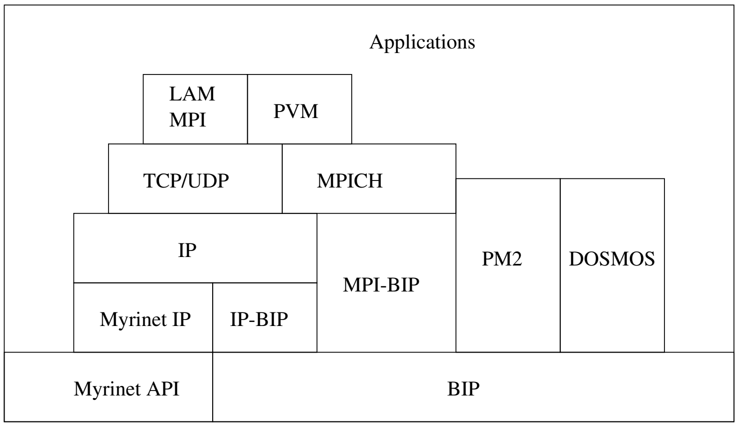


FIG. 2.2 – *Principaux logiciels de notre plate-forme*

- *MPI-BIP* Une mise en œuvre de l'interface standard MPI développée par l'équipe. Cette version a été réalisée en « portant » sur BIP le logiciel MPICH[6] de l'ANL.
- D'autres modifications du noyau Linux, offrant l'accès à des bibliothèques de mémoire virtuellement partagée (DOSMOS[9]) et de migration de threads (PM^2).

2.4.3 Utilisation de la plate forme

Les deux Pentium sont privilégiés pour l'utilisation courante. Ils sont toujours en mode multi utilisateur, et accueillent aussi des connexions de terminaux X-windows.

Les quatre Pentium Pro peuvent servir pour faire des tests et, dans cette optique, il est possible de les réserver. La réservation, déloge les autres utilisateurs.

2.5 Gestion et évolution de la plate-forme

2.5.1 Le site web

Le site web de l'équipe avait été mis en place une première fois par une personne dédiée à cette tâche, puis ensuite mis à jour au fil de l'eau, chacun ajoutant sa contribution, sans vue d'ensemble. Les défauts présentés étaient donc un manque de cohérence et de structure.

Le travail effectué a consisté en :

- La mise en conformité des pages à la norme HTML 3.2
- La remise en forme graphique, en éliminant l'utilisation de particularités de mise en page du navigateur Netscape
- La rédaction d'un petit guide de « bonne conduite web » pour l'équipe, à partir des erreurs fréquemment constatées
- La conception d'une structure plus lisible du site, et le le début d'application de celle-ci.

La mise en conformité des pages s'est effectuée grâce à l'étude de la norme bien entendu, mais également grâce à l'utilisation d'outils de contrôle automatisés, comme le module DTD du logiciel libre XEmacs, et l'éditeur HTML « amaya », en cours de développement par l'INRIA. Les problèmes de conformité peuvent se répartir essentiellement en 3 niveaux. Tout d'abord l'utilisation franche et directe de balises invalides (reconnues par certains logiciels seulement), ensuite l'utilisation de balises valides mais dans un but graphique lié au mode de présentation d'un navigateur donné, et enfin des constructions syntaxiques invalides, c'est à dire inclusion de balises dans des contextes non autorisés. Ces erreurs sont plus subtiles, car ces structures sont malgré tout interprétées (« corrigées» ?) par la plupart des browsers.

« Début d'application » et non pas réforme en profondeur de la structure du site car certaines adresses référencées à l'extérieur doivent être conservées telles quelles, ce qui représente une contrainte très gênante, et pas forcément prévue ni prévisible avant l'enrichissement du site. La prise de conscience de cette difficulté en particulier, et de toutes les autres liées à la tâche de webmestre en général, est je pense un bon enseignement, à mon avis très utile aujourd'hui à l'ère du « tout-web ».

2.5.2 Administration du site

L'équipe étant petite et l'administrateur principal étant actuellement soumis à de tristes obligations militaires, j'ai pris en charge l'administration courante de la plate-forme à la fin du stage. Ceci implique la résolution des problèmes typiques de l'administration système : impression réseau, ouverture de comptes, messagerie, installation et configuration de logiciels [1], etc.

1. à commencer par la boîte à outils CADP

Couches logicielles utilisées	*Performances en Mop/p/s*	
bench IS (Integer Sort)		
	Taille S	Taille A
LAM / IP / BIP	0.16	0.62
MPICH & MPI-BIP / BIP	0.49	2.02
bench décomposition matrice LU		
	Taille S	Taille A
LAM / IP / BIP	8.39	20.21
MPICH / IP / BIP	5.64	13.02
MPICH & MPI-BIP / BIP	21.47	

TAB. 2.1 – *Bench NAS*

2.6 Les mesures de performances

2.6.1 Les benchs NAS

La NASA est un demandeur traditionnel de fortes puissances de calcul. Son équipe NAS a programmé des codes de calcul parallèles portables, appelés NPB[4], qui donnent directement en sortie une mesure de performances. Ces codes correspondent à des calculs de type aérodynamique pour la plupart. Selon les benchs, les paramètres varient : granularité du parallélisme, quantité de calcul flottant ou parallèle, etc. L'intérêt pour la NASA est de faire mesurer objectivement, aussi bien sur ses machines que par les fabricants eux-mêmes, et sur des applications réelles, les performances des différents super calculateurs.

Pour pouvoir comparer facilement les performances de machines différentes, il faut que le même programme puisse s'exécuter sur toutes. L'équipe de la NASA a choisi d'utiliser l'interface de programmation standard MPI[6]. La promotion du standard MPI incite les fabricants à abandonner leurs langages parallèles exotiques, source de coûts supplémentaires pour les utilisateurs. L'utilisation des benchs NAS présentait pour nous un intérêt particulier, car cela permettait de déboguer et mesurer les performances de notre mise en œuvre de MPI, qui était en cours de développement et d'amélioration.

Le tableau 2.1 démontre l'intérêt de l'approche, consistant à construire directement une interface MPI sur une bibliothèque de bas niveau, sans passer par la couche IP de Linux (voir Figure 2.2 page 14). Les performances sont mesurées par une unité spécifique à chaque bench : le Mopérations/processeur/seconde

2.6.2 Les benchs Netperf

Les benchs Netperf sont des mesures de débit et de latence pour différents types d'environnements de communication, dont voici la liste :

- **TCP and UDP via BSD Sockets**
- DLPI
- Unix Domain Sockets
- Fore ATM API

– HP HiPPI Link Level Access

Sur notre plate-forme, ces benchs ont permis de mesurer les performances de la pile TCP/IP de Linux, dopée par notre pilote BIP-IP pour Myrinet. Il y a essentiellement 2 benchs, une mesure de *débit*, et une mesure de *latence*, qui s'exprime sous forme d'une vitesse de transactions élémentaires. Une transaction élémentaire est de la forme Requête/Réponse. Les résultats complets[8], dont voici un extrait, montrent que :

– Concernant le *débit* atteint, seul un réseau de type HiPPI[7], interfacé avec un bus machine spécifique à SGI excède de loin nos performances, avec 750 Mb/s. Mais une extension de TCP, utilisant des fenêtres supérieures à 64Ko, a été utilisée pour obtenir cette performance.
  Ensuite, seuls des réseaux de type Myrinet franchissent la barre des 200 Mb/s. Nos performances Myrinet, de l'ordre de 350 Mb/s, ont pu être dépassées seulement par des machines de type DEC Alpha.
– Les performances exceptionnelles de BIP concernant la *latence*, se répercutent directement sur les performances de la pile IP. Le nombre de transactions par seconde obtenu sur notre plate-forme est environ le double de la seconde performance.

Il est à noter que l'obtention de ces performances a nécessité certaines modifications de la pile TCP/IP de Linux, qui n'avait pas été conçue sous de telles contraintes. Ces performances ont été sanctionnées par l'envoi par l'auteur des bench Netperf d'un colis de cookies californiens [2].

2. véridique

Machines	Réseau & bus	Pile IP	Taille message (octets)	Débit (Mb/s)
DEC 500/266	Myrinet/PCI	OSF 4.0	8192	477.93
DEC500/Dual PPro	Myrinet/PCI	OSF/FreeBSD	16384	356.11
PPro 200	**Myrinet/PCI**	**Linux 2.0**	**7872**	**352.42**
DEC 500/266	Myrinet/PCI	OSF 4.0	8192	287.79

TAB. 2.2 – *Bench Netperf débit UDP*

Machines	Réseau & bus	Pile IP	Taille message (octets)	Débit (Mb/s)
SGI Power Challenge	HiPPI/HIO	IRIX 6.1	1048576	750.16
SGI O2000	HiPPI/XIO	IRIX 6.4	524288	745.89
PPro 200	**Myrinet/PCI**	**Linux 2.0**	**16384**	**338.05**
PPro 200	**Myrinet/PCI**	**Linux 2.0**	**7872**	**315.89**
DEC 500/266	Myrinet/PCI	OSF 4.0	65536	271.35

TAB. 2.3 – *Bench Netperf débit TCP*

Machines	Réseau & bus	Pile IP	Latence en (Transactions/s)
PPro 200	**Myrinet/PCI**	**Linux 2.0**	**8826.16**
HP K460	FC-266 [a]/HSC	HP-UX 10.20	4403.99
HP 735/99	FDDI/HP	HP-UX 10.00	3011.73

TAB. 2.4 – *Bench Netperf latence UDP*

Machines	Réseau & bus	Pile IP	Latence en (Transactions/s)
PPro 200	**Myrinet/PCI**	**Linux 2.0**	**7506.20**
Dual PPro	FastEth/PCI	Linux 2.0	4184.95
HP K460	FC-266/HSC	HP-UX 10.20	4159.73
PPro	FastEth/PCI	FreeBSD 2.2b	3536.08

TAB. 2.5 – *Bench Netperf latence TCP*

[a] Fiber Channel

Chapitre 3

Les méthodes formelles

3.1 Introduction générale

L'informatique est une technique et une science jeune. Elle ne possède pas le recul et les bases théoriques solides d'autres disciplines comme la mécanique ou la chimie [1]. Le génie logiciel, encore plus récent, a été le premier pas dans la recherche d'une méthodologie de la programmation. Il répondait à un besoin criant. Encore aujourd'hui, de nombreux codes écrits rapidement, au fil des besoins, et sans autre validation que quelques tests expérimentaux, sont en service dans tous les domaines de l'activité humaine. De plus, les premiers langages informatiques, souvent très proches de l'électronique des machines, ne permettaient ni de prendre le recul nécessaire à une vision d'ensemble d'un système, ni de démontrer rigoureusement – et donc de garantir – son bon fonctionnement. Les langages dits « objets », ont apporté un premier remède en apportant l'idée d'indépendance vis à vis du matériel, c'est à dire vis à vis de l'électronique, et une abstraction plus poussée, permettant de se rapprocher du problème à résoudre. Mais ils ne fournissent pas la notion mathématique de preuve, qui arrive en informatique seulement avec ce qu'on appelle « les méthodes formelles ». Ceci est rendu possible grâce à une abstraction encore plus poussée, et des modèles et théorèmes mathématiques associés.

3.1.1 Motivations

Bien entendu, dans une vision à court terme, l'utilisation des techniques [2] formelles est coûteuse, comme on pourra le voir par la suite. Alors pourquoi leur succès récents en informatique ? On peut essayer de citer quatre raisons majeures :

1. On a coutume de présenter la comparaison suivante, qui est discutable, mais qui a son intérêt pour se faire une idée de ce que sont les méthodes formelles : Pourquoi un informaticien écrit-il d'abord un programme, et ensuite vérifie son fonctionnement, alors qu'un mécanicien calcule un pont, et prouve ainsi qu'il tient, *avant* de le construire ?

2. le terme de « techniques » est parfois plus judicieux que celui de « méthodes », car les outils existants sont maintenant nombreux, mais les « instructions » pour s'en servir plus rares

Haute exigence de qualité

D'une manière générale, les systèmes où la vie humaine ou bien des sommes importantes sont en jeu, comme par exemple les transports ou l'aéronautique et l'espace, sont demandeurs d'un très haut niveau de garantie de sécurité. Mais l'informatique de ces systèmes est de plus en plus riche, remplit un rôle de plus en plus grand, et est de plus en plus difficile à concevoir, contrôler, maintenir, documenter et faire évoluer. Les méthodes formelles sont peut-être la réponse à la plupart de ces problèmes.

On peut aussi citer le cas de la conception des circuits intégrés, qui a trouvé depuis longtemps avec les méthodes formelles une solution vitale pour gérer une incroyable complexité.

Systèmes répartis

Les systèmes répartis, du parallélisme jusqu'au client/serveur longue distance, font sortir le fonctionnement des calculateurs du cadre théorique bien défini de la machine de Türing. Certaines techniques de modélisation formelles ont pour base mathématique les systèmes concurrents. Ces techniques permettent de prévoir des comportements difficilement prévisibles à l'avance.

Protocoles de communication

Les organismes de normalisation, au premier rang desquels l'ISO, publient des spécifications de protocoles pour les constructeurs. Ces normes définissent la nature des interactions au sein de systèmes répartis. Ceci afin de stimuler la concurrence, et défendre le consommateur, en permettant « l'interopérabilité » de systèmes provenant de constructeurs différents.

Malheureusement, les langages naturels[3] comportent des ambiguïtés qui en font les charmes et qui ravissent les poètes, mais qui peuvent être désastreuses dans ce cadre car source d'incompréhension entre les machines. Les langages de spécification, par leur capacité de description abstraite et leur rigueur mathématique, offrent une réponse parfaite à ce problème. En fait, certains de ces langages, dont LOTOS, ont été directement impulsés voire conçus par l'ISO.

Et ailleurs...

Dans le domaine du développement logiciel « classique », les avantages apportés par l'utilisation des méthodes formelles sont les mêmes que dans les cas vus ci-dessus, même si leur besoin se fait moins pressant. Elles permettent notamment d'anticiper les problèmes de conception, améliorer la communication au sein des équipes de développement, mais aussi et surtout entre le donneur d'ordre et le réalisateur, faciliter la maintenance (qui peut représenter jusqu'à 2/3 du coût du logiciel) et améliorer la documentation, générer automatiquement des jeux de tests, et même parfois du code exécutable. Même dans les cas où les outils disponibles étaient capables de fournir peu de résultats, on a souvent constaté que la modification des méthodes de travail dans les équipes de développement logiciel, notamment l'effort de rigueur, était rentable à long terme.

3. c'est à dire concrètement l'Anglais

3.1.2 Exemples

Toutes les techniques formelles sont basées sur une théorie mathématique précise, en générale récente. Le choix d'une technique ou d'une autre va donc conditionner les possibilités de spécification, de modélisation et de preuve. On peut noter d'autre part que l'attachement à une théorie est matérialisé en général par un langage de spécification, autour duquel gravitent des outils logiques. Cela est parfaitement naturel, car ce langage est le plus souvent une « traduction informatique » de la théorie mathématique sous-jacente. On peut remarquer aussi qu'un compromis entre expressivité, (c'est à dire facilité de modélisation) et puissance des outils de preuve est nécessaire.

Méthode Z et B

Le langage Z, qui a plus de 30 ans, repose sur la théorie des ensembles. Les états, les types, les opérations sont représentés par des ensembles. Cette méthode plutôt mathématique, conçue un peu loin de l'informatique, présente l'inconvénient d'être peu adaptée aux traitements automatisés. L'expression de l'itération ou de la récursion ne sont pas facilités. La méthode B a été conçue à partir des mêmes fondements que Z, mais avec des applications pratiques à l'esprit, notamment la génération automatique de code exécutable. Elle a été utilisé avec succès dans la conception de systèmes de pilotages, par exemple la nouvelle ligne Eole du réseau de trains de banlieue parisiens, ou le métro de Calcutta.

Grâce aux méthodes de ce type on peut, avec plus ou moins d'assistance humaine, « dérouler » de véritables démonstrations mathématiques, qui permettent de prouver par exemple, qu'une variable appartient toujours à un certain domaine, ou qu'une boucle se termine. Ce type de méthodes n'a pas été étudié ni utilisé pendant le stage.

Méthodes par évaluation de modèles [4]

Les méthodes que nous avons utilisés ici, sont basées sur les travaux théoriques de Milner[14] et de Hoare[12] qui ont fondé pendant les années 80 ce qu'on appelle le calcul ou l'algèbre de processus. Ces auteurs introduisent la notion « d'agent » ou « processus », qui permettra de modéliser parfaitement les systèmes concurrents et les protocoles de communication. Un langage de programmation parallèle, OCCAM, est directement issu de ces travaux. Aujourd'hui, un langage de spécification (LOTOS) traduction informatique quasi conforme des notations mathématiques de Milner et Hoare, a été normalisé par l'ISO. Et de nombreux outils ont été développés à partir de ces notions, comme par exemple la boîte à outils CADP[1] que nous avons utilisée. D'autres langages de ce type (model-checking) sont utilisés et normalisés auprès de l'ISO : ESTELLE et LDS. Ils ne sont pas algébriques et donc moins mathématiques, et plutôt basés sur les modèles d'automates communiquants.

Ces langages, associés à des outils de simulation ou des logiques temporelles, adressent naturellement les problèmes de synchronisations, d'interblocages [5], de séquences de messages, etc.

4. model-checking
5. deadlock

3.1.3 Limitations

Les techniques formelles ne sont pas la solution miracle à tous les problèmes de développement logiciel. Leurs limitations proviennent soit de leur nature même, soit de la récence de leur développement.

Limitations théoriques

Les méthodes formelles travaillent à partir d'une abstraction d'un problème. Cette abstraction est réalisée par l'homme. Un problème peut être tout simplement dissimulé par le mécanisme d'abstraction. Soit parce qu'il est réellement de bas niveau, et que sa résolution ne relève pas des méthodes formelles, soit suite à une erreur lors de la construction du modèle abstrait.

Quelque soit la branche des mathématiques sur laquelle s'appuie une méthode formelle, il y a toujours nécessité de faire un compromis, déjà évoqué dans la section 3.1.2, entre les possibilités d'expression du langage, qui vont souvent de pair avec la facilité de conception d'outils informatiques simples et performants, et la richesse et la « pureté » mathématique des modèles, qui augmentent elles les possibilités de démonstration théoriques.

Il est difficile d'inclure la notion de temps quantifié dans une théorie mathématique (à ne pas confondre avec l'ordonnancement). Pourtant, la plupart des systèmes informatiques utilisent le temps quantifié. On pense bien sûr immédiatement aux systèmes temps réel, mais il n'y a pas besoin d'aller aussi loin pour trouver des mécanismes de temporisation, avec des bornes bien définies (au bout de x secondes, faire...). En général, l'introduction du temps diminue la puissance théorique du modèle.

Limitations techniques

Le problème d'une approche mathématique, c'est qu'elle ne s'accorde pas toujours avec les problèmes pratiques d'efficacité de calcul sur les machines. Dans le domaine qui nous concerne, on verra par exemple que la génération de graphes peut « exploser », à cause de la difficulté de trouver le graphe minimal équivalent lors de la première passe de calcul. Régulièrement, des améliorations algorithmiques ou plus techniques apparaissent, la puissance des machines augmente, mais beaucoup de problèmes restent encore hors de portée.

D'une manière plus générale, l'intérêt porté au domaine étant neuf, les outils disponibles sont plutôt peu nombreux, en phase expérimentale, non stables, non optimisés, etc. On a alors un cercle vicieux, car investir lourdement dans un domaine à l'apparence expérimentale n'attire aucun industriel. Une des raisons qui sortira peut-être les méthodes formelles de ce cercle vicieux est la demande de clients influents pour des logiciels certifiées par des méthodes formelles. On peut citer l'exemple en Europe des normes ITSEC, qui établissent des critères à remplir dans le cadre de la sécurité, par exemple pour le commerce électronique.

3.1.4 Inconvénients

Comme on vient de le voir, il est difficile de prendre la décision d'investir dans les méthodes formelles, surtout lorsque des concurrents vendent sans s'en préoccuper. Aujourd'hui, les mécanismes économiques ont plutôt tendance à favoriser les investissements à court voire très court terme, et on a du mal à voir ce qui pourrait faire sortir les méthodes formelles des domaines où elles sont indispensables.

L'investissement est en effet lourd, et malheureusement, le coût est surtout « humain » c'est à dire lié à l'apprentissage de notions mathématiques riches et encore en plein développement, à ajouter à une maîtrise d'outils informatiques tout aussi instables.

3.2 LOTOS et CADP

Nous allons décrire plus concrètement dans cette section les outils théoriques et pratiques qui ont été utilisés.

3.2.1 L'algèbre de processus

	Contrôle	Données	Communication
SDL	Automates	ACT-ONE	Files d'attente
ESTELLE	Automates	PASCAL	Files d'attente
LOTOS	Algèbre de Processus	ACT-ONE	Rendez-vous

TAB. 3.1 – *Langages ISO*

Comme mentionné en section 3.1.2, LOTOS fait partie des 3 langages normalisés par l'ISO, dans le but d'exprimer sans ambiguïté des normes de protocoles de communication. On trouvera dans le tableau 3.1 les principales différences entre ces langages. Le parti pris de LOTOS est celui de l'abstraction mathématique. Les choix de la méthode de communication par rendez-vous, et d'éviter un langage algorithmique comme le Pascal pour les données, présentent l'avantage de favoriser le développement de théories rigoureuses, mais l'inconvénient de rendre plus difficile la mise en œuvre en machine, et l'apprentissage par l'utilisateur. Par exemple, le mécanisme de rendez-vous a été choisi parce qu'il permet de reconstruire une file d'attente, mais au prix d'un effort. Alors que le modèle par files d'attente ne permet pas lui de construire un rendez-vous.

On va tenter dans ce qui suit de présenter succinctement les bases de l'algèbre de processus, ou encore CCS, du titre du premier ouvrage de Milner. Ce terme de CCS a été regretté par l'auteur dans la révision [14] de cet ouvrage, mais est encore très utilisé. Pourquoi présenter CCS ? Parce qu'on a vu en section 3.1.2 que LOTOS était une traduction pratique quasi directe de CCS.

Pour des compléments, on trouvera dans les ouvrages de référence [14] et [12] des présentations complètes de respectivement CCS et CSP, et pour une excellente présentation du langage LOTOS en Français, on pourra se reporter à la thèse d'Hubert Garavel [11].

Un « processus » ou « agent » ou encore « comportement » est l'entité de base de CCS. C'est une entité qui n'existe en soi que par le biais de ses actions de communications/synchronisations avec l'extérieur, sous la forme de rendez-vous. C'est à dire que la définition de cet agent *est* la ou les suites d'actions qu'il est capable d'effectuer. Ainsi, voilà comment on pourra décrire le fonctionnement d'un distributeur de café, qui ne contient plus qu'un seul café :

$$1F \cdot Café \cdot stop$$

Le point $\cdot$ sert dans cette notation [6] à séparer séquentiellement les actions possibles. Le distributeur est premièrement en attente sur l'action externe $2F$, puis attend que le café soit sorti, et enfin tombe en panne.

On introduit ensuite un opérateur de choix, ou encore *sommation*, noté ici par le signe $+$. Il exprime le fait qu'un processus propose plusieurs actions

6. Selon les auteurs, ou les contraintes matérielles (LOTOS doit pouvoir être facilement entré sur un clavier), les notations ont de nombreuses variantes, mais toutes s'assimilent aisément

externes différentes, et que le choix s'effectuera en fonction de l'environnement ou du hasard. Cet opérateur permet d'introduire le non-déterminisme. Illustrons ceci avec l'exemple d'un distributeur qui ne rend pas la monnaie, et qui est capable de vendre un Coca 2F ou (« + ») un Café 1F puis de tomber en panne.

$$2F \cdot Coca \cdot stop + 1F \cdot (Café \cdot stop + 1F \cdot Coca \cdot stop) \tag{3.1}$$

Le symbole + signifie que le distributeur accepte de l'extérieur indifféremment au départ une pièce de 1F ou de 2F. Si l'environnement propose 2F, le comportement devient :

$$Coca \cdot stop$$

Alors que dans le cas de 1F, le distributeur est à nouveau devant un choix : donner un café ou accepter à nouveau 1F.

Une autre notion primordiale, est celle de la récursivité, qui permet d'exprimer des comportements qui bouclent. Voyons comment modéliser un simple tampon (en anglais *buffer*), ayant une capacité de 1 donnée seulement.

$$Buffer = Entrée(x) \cdot Sortie(x) \cdot Buffer$$

Vide au départ, ce tampon attend une donnée x, par l'action de communication *Entrée*, ce qui remplit sa capacité. Il n'est plus alors capable que de retransmettre cette donnée x sur la porte *Sortie*, et de recommencer.

Les opérateurs qui suivent maintenant, servent à « assembler » des processus qui vont communiquer entre eux. C'est l'opérateur $||$ de *composition* parallèle qui remplit essentiellement ce rôle. Grâce à lui, 2 tampons de capacité 1 du type précédent vont communiquer sur une action de communication *Intermédiaire* pour créer un tampon de capacité totale 2.

$$\begin{aligned} Buffer1 &= Entrée(x) \cdot Intermédiaire(x) \cdot Buffer1 \\ Buffer2 &= Intermédiaire(y) \cdot Sortie(y) \cdot Buffer2 \\ BufferDouble &= Buffer1 || Buffer2 \end{aligned}$$

La dernière notion de base, est celle qui apporte l'approche « boîte noire », c.-à-d. la hiérarchisation et l'abstraction. C'est elle qui permet de masquer des portes de communication, pour les réserver à un usage interne. Dans l'exemple précédent, l'environnement n'a pas à connaître le fonctionnement interne du tampon de taille 2. Il n'a pas à connaître l'existence du point de rendez-vous *Intermédiaire*. Ce qui signifie concrètement que le transfert d'une donnée du premier tampon au deuxième est un détail interne à masquer. Cette porte doit être cachée à l'extérieur de *BufferDouble*. Milner dans CCS note ceci de cette manière :

$$BufferDouble = (Buffer1 || Buffer2) \backslash Intermédiaire$$

On désigne alors l'action interne *Intermédiaire* sous le nom de τ*−transition*[7], ce qui signifie que cette transition peut être tirée indépendamment des événements extérieurs.

7. ou « i »(comme invisible ou interne) en LOTOS

On prendra note que dans LOTOS, les portes sont au contraire masquées par défaut, et qu'il faut préciser celles que l'on souhaite transmettre ou partager. Voici par exemple comment on définirait le $BufferDouble$ en LOTOS :

$$
\begin{aligned}
&BufferDouble = \\
&\qquad Buffer1(Entrée, Intermédiaire) \\
&\qquad\qquad\qquad\qquad |[Intermédiaire]| \\
&\qquad\qquad\qquad\qquad\qquad Buffer2(Intermédiaire, Sortie)
\end{aligned}
$$

Ceci peut d'ailleurs poser un problème pratique de lisibilité du code car la longueur des lignes de paramètres vite submergeante, mais ce fonctionnement de type « variables locales » est pourtant plus naturel d'emblée pour un programmeur. Programmeur qui déchante vite en découvrant qu'il n'a pas droit à aucune déclaration globale, ni même à des variables locales, rigueur mathématique oblige. Seules peuvent être déclarées des valeurs constantes (ce qui inclut des paramètres formels).

Ce point est aussi en fait l'une des principales différences théoriques entre LOTOS et CCS, car il ne s'agit pas seulement d'un problème de notation. En effet, avec CCS, le rendez-vous n-aire est impossible, et le rendez-vous 2 parmi n l'est, alors qu'en LOTOS, c'est exactement l'inverse[8].

3.2.2 La partie données de LOTOS

On a pu noter l'aspect purement fonctionnel de CCS, et l'absence de description d'un modèle de données. En LOTOS, qui se veut pas seulement théorie mathématique, il importait de choisir un modèle pour les données. Cette partie du modèle est complètement orthogonale à la partie fonctionnelle. Le style de « programmation » est d'ailleurs laissé à la totale liberté du programmeur, qui peut opter pour n'importe quelle option située entre les deux extrêmes. En effet, la plupart des problèmes peuvent s'aborder, plus ou moins facilement bien sûr, par l'approche fonctionnelle ou données. Par exemple, le tampon vu fonctionnellement dans la partie précédente, peut aussi être modélisé par une seule fonction manipulant une donnée de type liste.

Le modèle choisi par LOTOS s'appelle ACT-ONE, et est extrêmement abstrait. Il ne contient donc aucun type prédéfini, seule la congruence existe au départ. Même les entiers naturels doivent être définis récursivement. Heureusement, la définition de certains types de base se trouve dans la norme, et dans les outils LOTOS. Voici par exemple comment l'addition d'entiers naturels est définie en LOTOS :

```
opns  0    (*! constructor *) :          -> Nat
      Succ (*! constructor *) : Nat      -> Nat
      +                       : Nat, Nat -> Nat
eqns
      forall m, n : Nat
      ofsort Nat
         m + 0       =   m         ;
         m + Succ(n) = Succ(m) + n ;
```

8. Du moins sans astuces particulières

Grâce à cette définition par de simples équations, on peut déduire par exemple (« calculer ») que l'expression $3 + 2$ représente le même objet (est congruente à) l'expression $4 + 1$ qui est congruente à $5 + 0$ qui est congruente à l'expression 5. De même l'expression 5 ne fait que désigner sous une autre forme, l'expression primaire $Succ(Succ(Succ(Succ(Succ(0)))))$ elle-même irréductible, puisque formée uniquement à partir de `constructeurs`.

Cette représentation formelle des données, très abstraite et totalement indépendante des architectures, peut paraître au premier abord très contraignante à manipuler. En fait, ce n'est pas du tout cela qui pose problème, mais plutôt l'absence de la notion de variables dans le langage. Seules des constantes et des paramètres formels peuvent être utilisés.

3.2.3 Les systèmes de transitions étiquetés

Une spécification comportementale LOTOS est un code modélisant de façon abstraite le comportement d'un système. D'un point de vue pratique, cette spécification n'est pas utilisable directement. C'est pourquoi elle peut être traduite sous différentes formes, et notamment ce qu'on appelle les *systèmes de transitions étiquetés*, ou plus couramment LTS [9].

Un LTS décrit sous forme d'un graphe orienté l'ensemble des états dans lequel peut se trouver un système, les arcs reliant les nœuds étants *étiquetés* par les actions et donc *transitions* entre états que le système peut effectuer. On donne ici (Figure 3.1) l'exemple du graphe du LTS du $BufferDouble$ défini dans la section précédente. Les données x sont ici restreintes à 2 valeurs possibles, « 1 » et « 2 », pour limiter le nombre d'états (qui est une fonction exponentielle du nombre de valeurs possibles pour les données).

L'état où le tampon est complètement vide est l'état 0 en bas, et ceux où il est complètement plein (2 données) sont ceux du haut de la figure. On peut noter la $\tau - transition$ « i », qui correspond à la transmission interne d'une donnée du premier au deuxième tampon élémentaire.

3.2.4 Les bisimulations

Le premier problème qui se pose avec les LTS, est celui de l'unicité. Le graphe de la figure 3.2 représente tout aussi bien le comportement du $BufferDouble$ que celui de la figure 3.1. Alors, que faire ? Il est nécessaire d'établir une relation d'équivalence entre différents LTS, au moins pour être capable de trouver le graphe minimal correspondant à un comportement. De plus, la comparaison de LTS sera également utile pour comparer des modèles différents, par exemple pour vérifier qu'un modèle détaillé est bien équivalent à une description plus brève et plus abstraite de ce même système.

Une manière simple de définir une équivalence, est celle s'appuyant sur la notion de *(bi)simulation*. On cherche à savoir si un LTS est capable de « simuler » un autre, c.-à-d. s'il existe une relation particulière de l'ensemble produit $Q1 \times Q2$ des deux LTS $Q1$ et $Q2$, appelée *simulation* [10], qui à chacun des nœuds du 1^er^ LTS, va faire correspondre un nœud équivalent dans le 2^e^ LTS, c.-à-d. un

9. LTS : Labeled Transition System
10. ou *bi*simulation si elle est réciproque

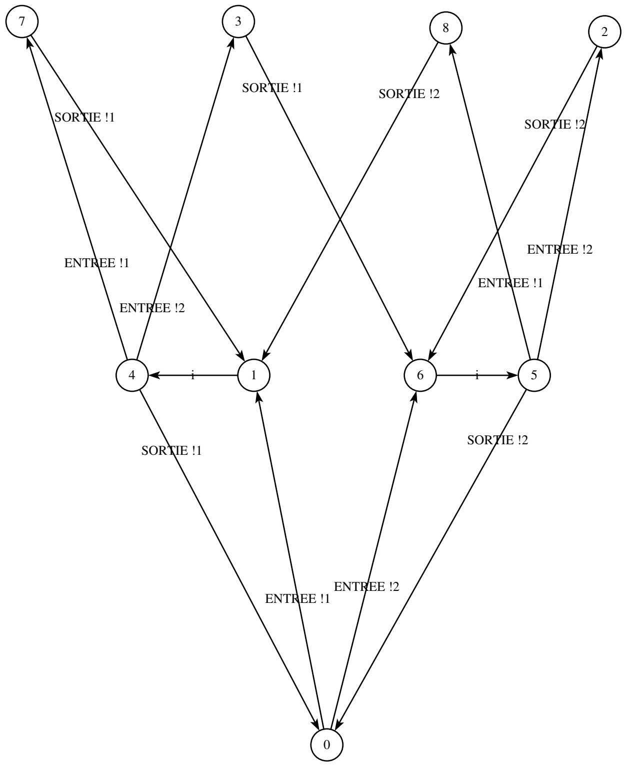


FIG. 3.1 – *Un LTS simple*

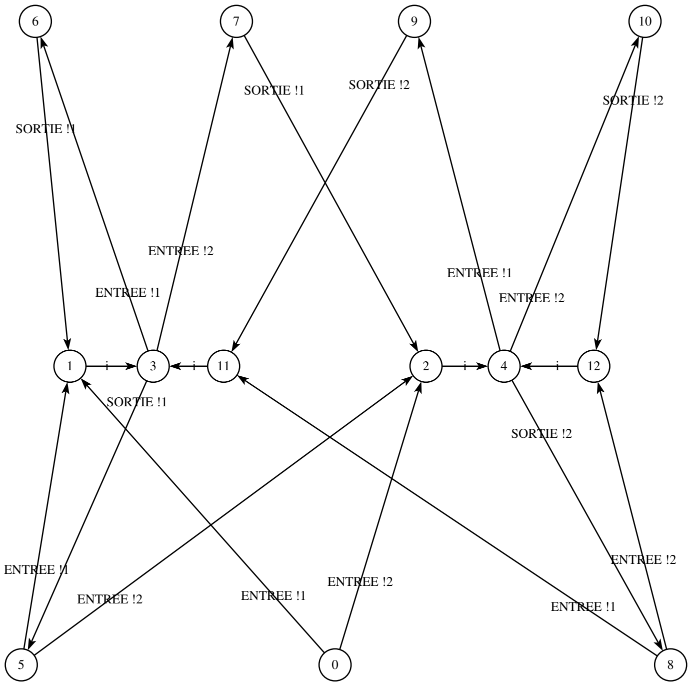


FIG. 3.2 – *Un LTS simple non minimal*

nœud capable de le « simuler ». L'équivalence entre nœuds est alors elle-même définie ainsi :

> Un état d'un 1^{er} LTS « simulant » est capable de simuler un état d'un 2^{e} LTS « simulé », si pour chaque transition partant de l'état du simulé, il existe la *même* transition partant de l'état du simulant. Mais de plus, cette transition simulante doit aboutir sur un état simulant *à son tour* l'état d'arrivée dans le graphe simulé.

Ce qui signifie que les états d'arrivée des transitions du LTS simulant devront simuler les états d'arrivée des transitions auxquelles elles correspondent. Grâce à cette définition, on peut trouver une bisimulation faisant correspondre à chaque état du LTS de la figure 3.1 un état du LTS de la figure 3.2, on peut donc dire que ces deux LTS sont équivalents.

On a noté au passage que la définition de la notion de simulation est récursive, car on a besoin d'états simulés pour définir ce qu'est un état simulé. On peut contourner cette difficulté en considérant tout d'abord l'ensemble produit $Q1 \times Q2$ des deux LTS, c.-à-d. l'ensemble des relations candidates potentielles au titre de (bi)simulation, puis définir un opérateur sur cet ensemble $Q1 \times Q2$, dont les point fixes seront des (bi)simulations. Le point fixe maximal, celui faisant correspondre le plus d'états des 2 LTS, étant le *pré-ordre* (ou *équivalence*) de (bi)simulation.

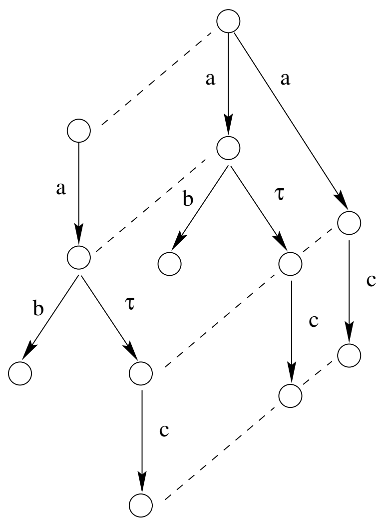


FIG. 3.3 – *2 LTS équivalents ?*

Lorsqu'on cherche à concevoir une relation d'équivalence, un problème majeur se pose : doit-on ignorer les $\tau - transitions$ lors des comparaisons ? On a plutôt envie de répondre oui a priori, car on a vu que l'intérêt des $\tau{-}transitions$ était justement de masquer les détails internes d'un système. Malheureusement, les choses ne sont pas aussi simples. Considérons les 2 LTS de la figure 3.3.

Si l'on choisit une définition de bisimulation qui ignore les $\tau - transitions$ (bisimulation *faible*), alors les états reliés par des lignes pointillés sont des états équivalents. Les deux LTS se simulent bien l'un l'autre complètement par une bisimulation faible. Pourtant, ces deux graphes ne sont pas *intuitivement* équivalents. En effet, dans le graphe de gauche, il sera toujours possible d'effectuer b après a, alors que dans celui de droite, la possibilité existe, mais elle n'est pas garantie. Le choix entre b et c peut s'effectuer *avant* a.

On peut alors décider à l'inverse de prendre en considération toutes les $\tau - transitions$ dans la définition des simulations. Mais on constate alors rapidement que cette définition est au contraire parfois trop stricte, et empêche de considérer comme équivalents des systèmes qu'on aimerait qu'ils le soient. La solution (ou plutôt les solutions), réside dans le choix d'une bisimulation intermédiaire entre ces 2 extrêmes. On peut décider d'autoriser les $\tau - transitions$ seulement avant, ou au contraire après, une action externe, ce qui définit autant de bisimulations différentes, préservant différentes propriétés. On se reportera à la thèse de Laurent Mounier [16] pour une étude précise et complète des différentes bisimulations.

3.2.5 Logiques modales et temporelles

D'un point de vue informatique, les LTS sont une forme déjà plus exploitable qu'un modèle en LOTOS. On peut les comparer entre eux, les « exécuter », c.-à-d. effectuer des simulations en tirant des transitions au hasard ou guidées. Mais cela reste limité. Une question qui se pose maintenant, est de savoir comment exprimer des propriétés logiques que l'on a envie de vérifier sur un système, des propriétés qui viennent à l'esprit en langage naturel. Une solution à ce problème est apportée par les logiques modales, et leur extension, les logiques temporelles. Une référence assez complète et rigoureuse sur ce sujet est l'article [21].

Logiques modales

On construit les formules de logique modale à partir d'éléments de base. Une formule Φ s'applique à un comportement E donné (pris à un instant donné), qui peut la *satisfaire* ($E \models \Phi$) ou pas ($E \not\models \Phi$). Les deux premiers éléments sont naturellement **true** et **false**, qui sont vérifiés respectivement par n'importe quel et aucun processus. On adjoint à cela les opérateurs binaires booléens classiques $\wedge$ (et) et $\vee$ (ou inclusif). Et enfin les opérateurs *modaux* proprement dits : $[\lambda]\Phi$ (« box-lambda phi ») et $\langle\lambda\rangle\Phi$ (« diamond-lambda phi »), où λ est un ensemble de transitions et Φ une précédente condition. Comme leur nom l'indique (opérateurs *modaux*), ces opérateurs expriment les en quelque sorte les locutions : « il est possible que » et « il faut que ».

$[\lambda]\Phi$ Après une transition appartenant à l'ensemble λ, le nouveau comportement *doit* vérifier Φ.

$\langle\lambda\rangle\Phi$ Après une transition appartenant à l'ensemble λ, *il existe* un nouveau comportement qui vérifie Φ.

$$\begin{aligned} E \models [\lambda]\Phi \quad & ssi \quad \forall F \in \{E' | E \xrightarrow{a} E', a \in \lambda\}, F \models \Phi \\ E \models \langle\lambda\rangle\Phi \quad & ssi \quad \exists F \in \{E' | E \xrightarrow{a} E', a \in \lambda\}, F \models \Phi \end{aligned}$$

La nécessité immédiate s'exprime alors par : la possibilité de certains actions + l'interdiction du reste. Prenons l'exemple du distributeur de l'équation 3.1. Dans son état initial, il va nécessairement d'accepter une pièce d'1F ou de 2F, et cela s'écrit ainsi

$$\langle 1F, 2F\rangle \mathbf{true} \wedge [\mathcal{A} - \{1F, 2F\}]\mathbf{false}$$

(Les opérateurs modaux sont prioritaires sur les opérateurs logiques et $\mathcal{A}$ représente l'ensemble de toutes actions possibles.)

Après l'introduction de 2F, comme le distributeur ne rend pas la monnaie, il est possible de prendre un Coca mais plus de Café :

$$[2F](\langle Coca\rangle \mathbf{true} \wedge [Café]\mathbf{false})$$

Il est possible à l'aide de la logique modale de caractériser complètement le distributeur de l'équation 3.1. Mais l'intérêt d'une telle logique est justement de pouvoir exprimer une propriété *partielle* d'un modèle.

De la même manière qu'il existe différentes bisimulations, il peut être intéressant d'introduire des opérateurs $[[\ ]]$ et $\langle\langle\ \rangle\rangle$ qui ignorent les $\tau - transitions$. On ne détaillera pas plus ce point dans ce rapport.

Logiques temporelles

La forte limitation des logiques modales vient du fait qu'elles ne peuvent exprimer que des propriétés immédiates, et pas le fait qu'une action sera toujours possible, ou qu'une autre adviendra obligatoirement un jour par exemple. Elles ne sont pas adaptées aux processus définis récursivement. Si l'on considère le simple tampon suivant :

$$Buffer = Entrée \cdot Sortie \cdot Buffer$$

On aimerait par exemple pouvoir exprimer tout simplement la propriété que l'action *Entrée* va se produire indéfiniment. Les logiques temporelles permettent cela. En fait, le $\mu - calcul$ défini plus loin permet de caractériser complètement, c.-à-d. de définir, le tampon ci-dessus.

Pour introduire les logiques temporelles, nous allons définir des équations modales récursives. Pour cela, nous avons besoin d'opérateurs sur des sous-ensembles de processus. Notons $||\Phi||^{\mathcal{E}}$ le sous-ensemble des processus de $\mathcal{E}$ qui vérifient la condition Φ.

$$||\Phi||^{\mathcal{E}} = \{E \in \mathcal{E} : E \models \Phi\}$$

On définit ensuite les opérateurs $||[\lambda]||$ et$||\langle\lambda\rangle||$ faisant correspondre à un ensemble $||\Phi||^{\mathcal{E}}$ respectivement les ensembles $||[\lambda]\Phi||^{\mathcal{E}}$ et $||\langle\lambda\rangle\Phi||^{\mathcal{E}}$.

$$\begin{aligned} ||[\lambda]\Phi||^{\mathcal{E}} &= ||[\lambda]||||\Phi||^{\mathcal{E}} \\ ||\langle\lambda\rangle\Phi||^{\mathcal{E}} &= ||\langle\lambda\rangle||||\Phi||^{\mathcal{E}} \end{aligned}$$

Ceci n'est possible que si $\mathcal{E}$ est un ensemble *fermé*, c.-à-d. tel que :

$$\forall (E, a) \in (\mathcal{E}, \mathcal{A}), \text{ si } E \xrightarrow{a} F \text{ alors } F \in \mathcal{E}$$

Une remarque importante est que ces opérateurs sont monotones relativement à la relation d'ordre d'inclusion.

Maintenant, en s'inspirant de la manière dont l'on a défini un ensemble P de processus vérifiant une propriété Φ ($P = ||\Phi||$), on introduit des ensembles de processus vérifiant une propriété définie *récursivement* : $P = ||\langle\lambda\rangle||P$. Les ensembles vérifiant ces conditions récursives sont les points fixes des opérateurs du type $||\langle\lambda\rangle||$. En règle générale plusieurs ensembles peuvent satisfaire ce genre de conditions. Pour l'exemple du $Buffer$ ci-dessus, la condition $P = ||\langle Sortie.Entrée\rangle||P$, qui exprime le fait qu'il est éternellement possible d'effectuer l'action $Sortie.Entrée$, est satisfaite par les ensembles de processus suivants : $\{\,\}$, le singleton $\{Sortie.Buffer\}$, et pas par la paire $\{Buffer, Sortie.Buffer\}$.

Deux des ensembles solutions d'une telle « équation modale récursive » méritent une attention particulière : le plus petit et le plus grand, c.-à-d. les points fixes extrêmes des opérateurs du type $f = ||[\lambda]||$. On prouve leur existence grâce à la monotonie des opérateurs du type f, relativement à la relation d'inclusion $\subseteq$ des sous-ensembles de processus (si $P \subseteq Q$ alors $f(P) \subseteq fQ)$). On note ces extrema respectivement $\mu X.\langle\lambda\rangle X$ et $\nu X.\langle\lambda\rangle X$ (pour un opérateur modal $\langle\lambda\rangle$), et on montre qu'il sont égaux à :

$$\begin{aligned}
\mu X.\langle\lambda\rangle X &= \cap\{P \subseteq \mathcal{E} : ||\langle\lambda\rangle||P \subseteq P\} \\
\nu X.\langle\lambda\rangle X &= \cup\{P \subseteq \mathcal{E} : P \subseteq ||\langle\lambda\rangle||P\}
\end{aligned}$$

C'est le fait qu'un processus donné appartienne ou non à l'un de ces deux ensembles, qui va permettre d'exprimer des propriétés temporelles intéressantes.

Prenons par exemple $\nu X.\langle Sortie\rangle X$. Il s'agit de l'ensemble des processus ayant la possibilité d'effectuer l'action $Sortie$ indéfiniment quelque soit leur état futur. La condition d'appartenance à $\nu X.(\Phi \wedge [\mathcal{A}]X)$ désigne elle le fait que la condition Φ est vraie dans tous les états futurs possibles. $\mu X.(\Phi \vee \langle\lambda\rangle X)$ exprime le fait que le processus peut effectuer des actions λ jusqu'à ce que la condition Φ soit vraie. $\nu X.(\Phi \vee \langle\lambda\rangle X)$ inclut en plus les processus capable d'effectuer éternellement λ (sans forcément atteindre Φ).

La manière dont nous avons défini une logique temporelle, forme ce qu'on appelle le $\mu - calcul$, un des langages utilisés dans CADP. Ce langage permet d'exprimer puissamment de nombreuses proriétés logiques. Malheureusement, son apprentissage n'est pas des plus faciles, et encore moins la maîtrise nécessaire lorsqu'on veut déterminer rapidement en $\mu - calcul$ l'expression d'une propriété ressentie de façon intuitive. De plus il ne permet pas par exemple de caractériser un tampon de taille supérieure à 1, car il est « sans mémoire ».

3.2.6 CADP (Caesar/Aldebaran Development Package)

La boîte à outils CADP est développée à Grenoble par l'équipe du projet VASY[1], qui est un groupe de recherche commun à l'INRIA Rhône-Alpes et la société Bull. C'est un ensemble de logiciels travaillant à partir d'une base qui est le langage LOTOS. Le principal objectif visé est la validation d'une architecture multiprocesseurs de Bull, mais CADP est utilisé dans le monde entier pour des protocoles de base de données, de mémoire répartie, etc.

La performance de l'équipe VASY réside dans le fait de travailler à la fois sur la partie théorique mathématique (nouveaux algorithmes de recherche) et la partie évolution/normalisation de LOTOS[11], tout en étant capable de fournir gratuitement des logiciels opérationnels et un support technique minimal. Nous allons décrire dans cette partie ces logiciels. Un problème rencontré lors de la découverte de cette boîte à outils est que la documentation, qui précise le mode d'emploi de chaque outil, peut parfaitement suffire à quelqu'un initié et au fait des *techniques* formelles et des théories présentées ci-dessus, mais il est difficile à l'heure actuelle de trouver des *méthodes* formelles à proprement parler guidant l'utilisateur dans la démarche d'analyse de son problème. De ce fait, on découvre à l'inverse d'abord les possibilités, et l'on en déduit ensuite les problèmes que l'on va pouvoir traiter. Cela est renforcé par la difficulté d'anticiper les limites pratiques des outils, quand on n'a pas idée de la nature des calculs qui sont derrière. Un peu comme si l'on demandait à un non aérodynamicien de déterminer la finesse d'un maillage d'aile d'avion, le résultat serait probablement inutilisable informatiquement parlant.

De plus, l'équipe VASY ne disposant pas de ressources humaines infinies, seule pour l'instant une version Sun Solaris de CADP est disponible, la version Linux étant annoncée. La machine Sun du LIGIM utilisée pendant ce stage était un peu « à l'abandon » (plus utilisée et donc plus vraiment administrée) et cela a parfois posé des problèmes de disponibilité du logiciel.

L'outil `caesar`

La première opération effectuée sur le code LOTOS, est soit de générer un LTS (non minimal a priori, voir section 3.2.3) décrivant le système, ou bien de le rendre réellement « exécutable », c.-à-d. de le traduire un programme en langage informatique C[12], qui est capable de simuler le comportement du système lorsqu'on lui transmet séquentiellement les actions de communication/ synchronisation, et qui sera interfacé avec d'autres outils, y compris ceux conçus par l'utilisateur. La principale et unique restriction théorique de l'outil CAESAR vis à vis de la norme LOTOS, est qu'il n'accepte pas la récursion non terminale, c.-à-d. l'instanciation d'un processus par lui-même lorsque les deux devraient continuer à exister.

L'outil `aldebaran`

Cet outil sert à manipuler des LTS. Grâce à lui, il est possible de comparer deux LTS, représentant par exemple une vision abstraite et une vision détaillée d'un système; et il est possible de rechercher un graphe minimal, à partir d'un LTS donné. Plusieurs algorithmes de parcours sont disponibles, et surtout plusieurs équivalences différentes.

L'environnement `OPEN/CAESAR`

L'environnement OPEN/CAESAR est ce qui permet d'exploiter indifféremment des spécifications au format LOTOS, LTS, ou déjà traduites en C, et de les exploiter avec un module fourni ou programmé par l'utilisateur. Les modules

11. VASY contribue à E-LOTOS, extension de LOTOS en cours de normalisation
12. pour des questions de performance essentiellement

fournis comprennent :

`[X]simulator` un simulateur pas à pas, en mode ligne de commande ou graphique. Utile pour des utilisations de débuggage, démonstration, etc.

`executor` un simulateur aléatoire

`exhibitor` une recherche de séquences d'actions données

`evaluator` une vérification de formules de logiques temporelles, en langages $\mu - calcul$ et XTL

`terminator (!)` une recherche de deadlock par la technique de G. Holzmann

`xeucalyptus` une interface graphique réalisée en langage TCL/TK, permettant de lancer ces outils de façon interactive.

On peut remarquer aussi dans la toute dernière version l'apparition d'un assistant d'installation convivial.

3.3 Modélisation MPI-BIP

3.3.1 Le logiciel MPICH

MPICH est développé à l'Argonne National Laboratory et à l'université du Missisipi.

Les deux objectifs probablement antinomiques de ce projet sont l'efficacité et la portabilité (puisqu'il devrait s'adapter à son environnement aussi bien qu'un *chameleon* à qui il a emprunté ses deux premières lettres). Le nombre de machines et de types de machines sur lesquelles MPICH a été porté montre ses qualités : machine parallèle à mémoires distribuées, à mémoire partagée et réseau de stations de travail. La figure 3.4 donne une représentation en couches du logiciel.

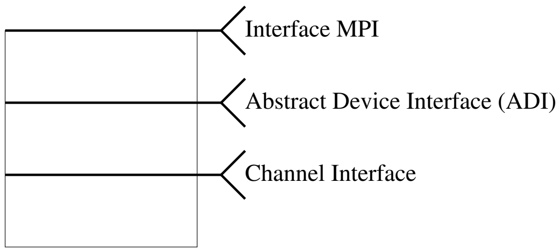


FIG. 3.4 – *le logiciel MPICH*

- l'*Abstract Device Interface* regroupe un nombre assez vaste de primitives. Quatre ensembles de fonctions doivent être fournis: pour spécifier l'envoi ou la reception d'un message, pour transférer les données depuis l'application utilisateur jusqu'au réseau, pour gérer une file de messages en attente et pour fournir des informations sur l'environnement.
- Le *Channel Interface* : Tout ce qu'il faut fournir ici, c'est un moyen de transferer des données d'une machine à une autre. Le développement de cette couche est donc très rapide même si il faut assurer un contrôle de flot à ce niveau.

MPI-BIP est la couche qui fournit le *Channel Interface* sur notre plate-forme.

3.3.2 Le protocole

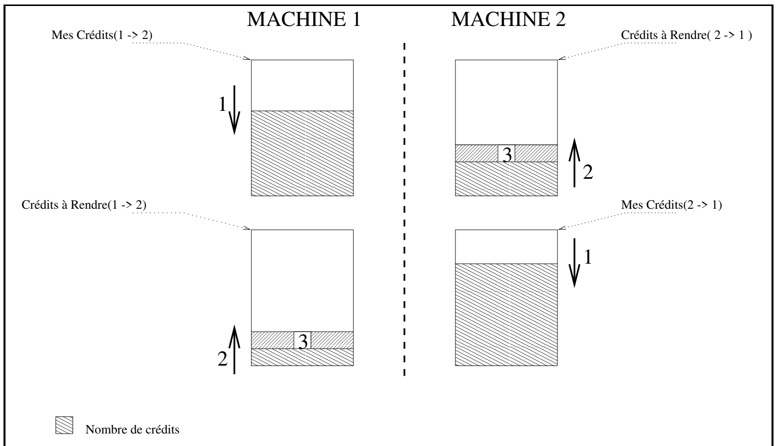
Dans la mise en œuvre de MPI-BIP[22], le logiciel a à gérer les files d'attente de BIP (voir figure 2.2). En effet, BIP fournit des services à haute performance, mais très limités, et l'utilisateur doit faire attention à ne pas dépasser la taille de la file d'attente des petits messages. (Les messages longs doivent quant à eux être envoyés selon un protocole de rendez-vous.) Pour gérer cette file, la mise en œuvre de MPI-BIP utilise un protocole de gestion par crédits présenté ici.

Chaque machine émettrice dispose en interne d'un nombre de crédits de messages pour la machine réceptrice, égal au nombre de messages qu'elle a le droit d'envoyer à cette machine (*crédits_chez_lui*). Elle mémorise ainsi l'état de la file d'arrivée en face d'elle. Lorsqu'elle n'a plus de crédits, c'est que la file est probablement pleine, elle arrête ses envois. Bien entendu, la machine réceptrice doit lui *rendre* régulièrement des crédits, correspondant aux messages qui ont été sortis de la file d'arrivée, et qui ont libéré de nouvelles places. Ce qui signifie que la machine réceptrice doit aussi maintenir un compteur, correspondant au nombre de *crédits_à_rendre*.

crédits_chez_lui, initialement égal à la taille de la file, est décrémenté à chaque émission de message sur le réseau physique. *crédits_à_rendre*, initialement nul, est incrémenté à chaque sortie par la couche supérieure d'un message de la file d'arrivée. On a en permanence :

$$\begin{aligned} &taille\ de\ la\ file = \\ &\quad cr\acute{e}dits\_chez\_lui + cr\acute{e}dits\_\grave{a}\_rendre \\ &\quad + nombre\ de\ messages\ encore\ dans\ la\ file\ d'arriv\acute{e}e \end{aligned}$$

crédits_à_rendre est remis à zéro à chaque fois que les crédits sont rendus. *crédits_chez_lui* est incrémenté par les crédits rendus. Bien entendu, ce fonctionnement est double : chaque machine est à la fois émettrice et réceptrice. Il est en plus multiplié par n-1 dans le cas de n machines. Et il est encore multiplié car BIP et MPI-BIP permettent l'utilisation simultanée de plusieurs files. On a considéré pour la modélisation que les gestions de ces différentes files étaient indépendantes. Ce protocole est semblable dans l'esprit à un contrôle par fenêtre d'émission.



1. Chaque fois que la machine i envoie un message de contrôle à la machine j, Mes_Crédits($i \rightarrow j$) diminue: la machine i peut envoyer moins de messages de contrôle à j.

2. Chaque fois que la machine i retire un message de contrôle en provenance de j de la file d'attente, Crédits_à_Rendre($i \rightarrow j$) augmente: un buffer a été libéré et peut donc recevoir un nouveau message.

3. Crédits_à_Rendre($j \rightarrow i$) + Mes_Crédits($i \rightarrow j$) est au plus égal à Max_Crédits mais, peut-être inférieur à cette borne: les messages qui sont stockés dans la file d'attente en attente de traitement représentent des crédits qui ne peuvent pas être rendus.

FIG. 3.5 – *Le contrôle de flot à crédits pour les petits messages*

Pour éviter l'envoi supplémentaire de messages spéciaux pour la gestion des crédits, on utilise le *piggybacking*, qui consiste à glisser l'information du nombre de crédits rendus à l'intérieur de messages existants, et envoyés de toute façon. Pour les cas extrêmes, un message spécial CREDIT est prévu.

Quels sont plus précisément maintenant les messages existants ? La couche immédiatement supérieure à MPI-BIP s'appelle ADI. Elle émet 2 types de messages :

- les messages longs, appelés DATA.
- et les courts, appelés ConTRoL (CTRL), servant à la fois pour son usage interne et aussi pour les transmissions de brefs paquets de données, pour lesquels l'utilisation de DATA serait du gaspillage.

La couche MPI-BIP va transmettre ces deux types de messages, et ajouter ceux pour son usage propre :

- ReQueST pour demander un rendez-vous (systématique avant l'envoi d'un message long DATA)
- ACKnowledge pour accepter ce rendez-vous.
- CREDIT déjà vu plus haut, pour rendre urgemment des crédits.

Le piggybacking utilisera les messages RQST et ACK pour rendre des crédits.

3.3.3 La modélisation

Ce protocole de contrôle de flot a été choisi comme cas d'étude parce qu'il semblait suffisamment classique pour être abordé même sans pratique préalable de LOTOS et CADP et malgré tout suffisamment complexe pour présenter un intérêt et faire apparaître des problèmes d'interblocage, et donc nécessiter une validation. En effet, il est facile de voir par exemple que si une place n'est pas *explicitement* réservée dans la file pour la réception d'un message CREDIT, les deux machines peuvent se retrouver toutes deux simultanément file pleine en attente éternelle d'un message CREDIT. Un objectif a donc été d'essayer de retrouver ce résultat à l'aide de la boîte à outils CADP. Un autre objectif était aussi de comparer l'ensemble des deux machines communiquant par le biais de ce protocole, à un simple tampon de capacité identique, et de vérifier que ces deux comportements étaient équivalents.

Pour l'écriture du modèle, seuls les aspects suivants ont été retenus : deux procédures bloquantes SENDHIGH et RECVHIGH sont disponibles pour la couche supérieure (c'est à dire l'extérieur de la spécification). Une garde protège la procédure RECVHIGH lorsque la file est vide : on fait l'hypothèse qu'il n'y a pas de demandes de réception s'il n'y a pas de messages disponibles. Pendant l'attente de la complétion de ces procédures, la réception bas niveau de messages en provenance du réseau (RECVLOW) continue à être possible en permanence. Voici le squelette LOTOS extrêmement simplifié du corps principal du modèle d'une seule machine. On a retiré la quasi totalité des paramètres (valeurs et gardes), quand ils n'apportaient rien à la compréhension du code, et supprimé des détails pratiques, pour essayer d'atteindre une certaine lisibilité. On consultera le listing complet en annexe pour constater la différence.

```
process BOUCLE ( FILE, CREDITCHEZLUI, CREDITARENDRE )

(
hide INITIATIVE in
 INITIATIVE ? X  [ CREDITARENDRE  >=  1 ] ;
   SENDHIGHproc ( , CREDITARENDRE, CREDITCHEZLUI - 1  )
                                   >> BOUCLE ( , , 0)
)
[]
[ CREDITCHEZLUI > 0 ou 1 ] ->
(
 SENDHIGH ? S : MSG
   SENDHIGHproc ( , S, CREDITCHEZLUI - 1 ) >>  BOUCLE ( , , )
)
[]
 RECVHIGH ! FIRST (FILE) ;
   RECVHIGHproc ( , )  >>  BOUCLE ( , , CREDITARENDRE + 1 )
[]
 RECVLOWproc ( ) >>  BOUCLE ( , CREDITCHEZLUI + RENDUS,  )
```

Voici maintenant la partie instanciant les 2 machines et les mettant en communication par le biais de 2 portes internes FROMA2B et FROMB2A.

```
behaviour

let QUEUEA,QUEUEB : LIST = <> in
hide FROMA2B, FROMB2A in

BOUCLE [ENTREEA,FROMA2B,SORTIEA,FROMB2A]  (QUEUEA,   2 , 0 )
     | [FROMA2B, FROMB2A] |
BOUCLE [ENTREEB,FROMB2A,SORTIEB,FROMA2B]  (QUEUEB,   2 , 0 )
```

Le problème qui s'est alors posé, pour l'utilisation pratique de ce modèle, est celui de l'explosion de l'espace des états. Ce problème est particulièrement provoqué par l'utilisation de données dans le code. En effet, à chaque augmentation de l'excursion possible d'une donnée, la taille du graphe en nombre d'états est multipliée. Reprenons l'exemple du tampon à deux places de la figure 3.1 page 28, on avait alors limité l'excursion des données transitant à seulement deux valeurs possibles, 1 et 2. Constatons dans la figure 3.6, ce qu'est devenu la taille du graphe du même tampon, simplement étendu à une capacité de 3 places et à 3 valeurs différentes pour les données.

Mais surtout, il faut bien comprendre que le graphe de la figure 3.6 est le graphe « minimum », obtenu uniquement grâce à une réduction du graphe généré initialement, environ 2 fois plus gros (voir la figure 3.7)

Le principal problème est là, dans la génération d'*un* graphe minimal au premier jet qui ne s'éloigne pas trop *du* minimum. Ce problème très complexe, à cheval sur la théorie mathématique et l'efficacité des architectures de machines, est encore assez vierge, et les experts estiment que de nombreux progrès sont encore possibles[13]. On trouvera des articles et des références sur ce sujet sur le

13. plusieurs ordres de grandeur

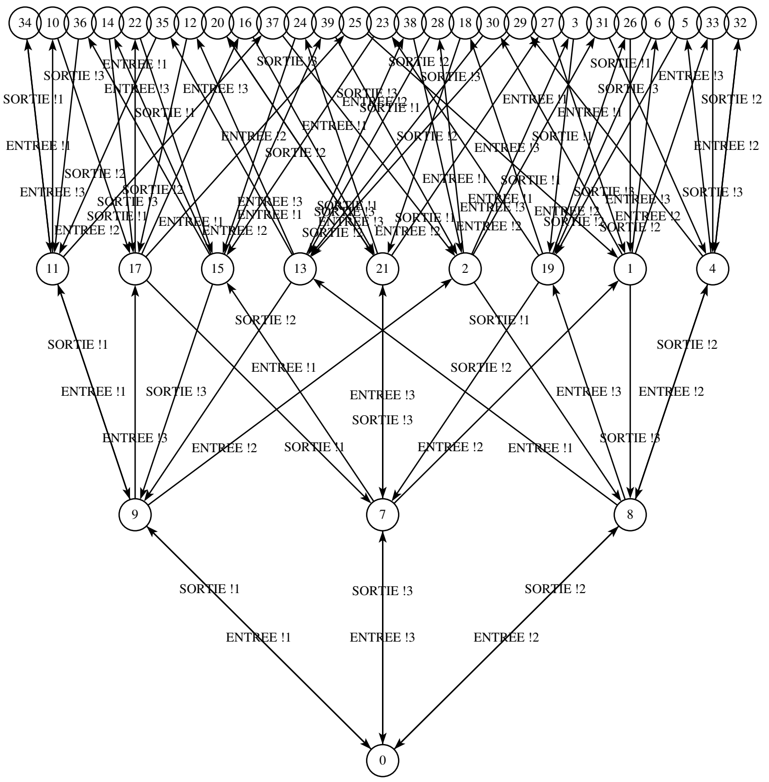


FIG. 3.6 – *Tampon 3 places 3 valeurs*

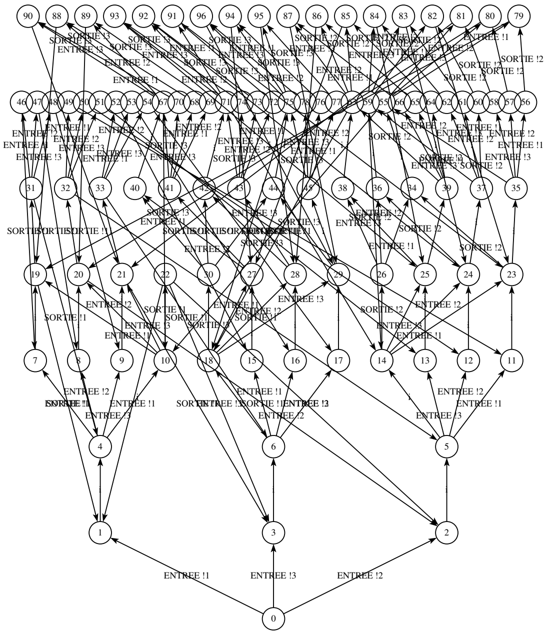
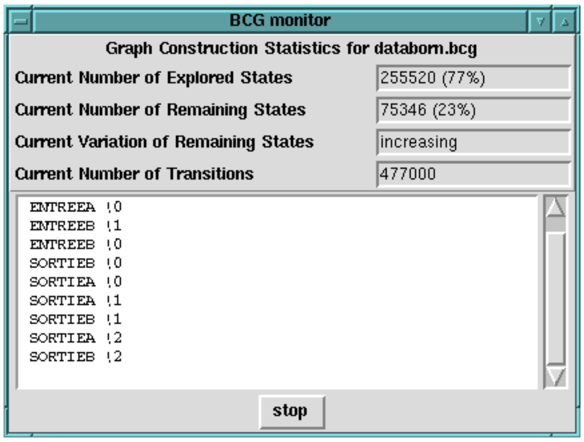


FIG. 3.7 – *Tampon 3 places 3 valeurs non réduit*

site Web de l'équipe VASY [1], et dans la boîte à outils CADP elle-même. L'idée d'une coopération entre les équipes VASY et LHPC afin d'utiliser la puissance de la plate-forme parallèle Myrinet dans l'exploration des états est à l'étude. Mais il faut bien savoir que les progrès les plus importants sont à faire du côté algorithmique.

Dans notre cas, `caesar` n'a pas été capable de générer un graphe du modèle. Le programme ralentit puis s'arrête, les ressources mémoire de la machine épuisées [14], des centaines de milliers d'états et de transitions parcourus, la proportion entre états parcourus et états restants à parcourir semblant d'ailleurs tendre vers une limite dépendant du modèle traité (figure 3.8).




```
  PID  %CPU   SZ  VSZ  RSS   %MEM        TIME COMMAND
 7840  3.3 16569 66276 49876 80.1        5:49 caesar
```

FIG. 3.8 – *Explosion d'états*

Une première réponse à ce problème apportée par CADP est l'utilisation d'algorithmes travaillant « à la volée », sur une version exécutable (en langage C) du modèle, sans chercher à générer le graphe complet du modèle. Dans notre cas, avec l'outil **`simulator`**, nous avons bien pu retrouver le résultat d'interblocage présenté dans la section 3.3.3 par des simulations guidées pas à pas, et son absence pour les mêmes séquences après correction du modèle.

La détection d'interblocage de G. Holzmann (**`terminator`**) a elle aussi permis de retrouver, cette fois-ci automatiquement, une séquence d'interblocage, et de vérifier son absence après correction. Mais la méthode de Holzmann ne garantit pas formellement l'absence de blocage. Cette séquence consiste simplement à envoyer des paquets ne transportant pas de crédits rendus, jusqu'à ce que les files soient pleines.

```
caesar.open: running ``terminator -first -depth 26 0 100''
actual size of bitmap: 97

*** deadlock found at depth 24
```

14. au bout d'un certain temps, la machine ne se consacre plus qu'exclusivement au « swap » entre la textscram et le disque.

```
<initial state>
"ENTREEA !0"
"i" (FROMA2B [54])
"ENTREEB !0"
"ENTREEA !0"
"i" (FROMA2B [54])
"i" (FROMB2A [54])
"ENTREEB !0"
"SORTIEA !0"
"i" (FROMB2A [54])
"SORTIEB !0"
"SORTIEB !0"
"SORTIEA !0"
<deadlock>
```

La profondeur de 24 ne correspond pas au nombre de transitions présentées, car des transitions internes « *i* » non significatives ont été retirées de la séquence imprimée ci-dessus.

Cependant, le fait de ne pas disposer d'une vue globale du comportement sous la forme d'un graphe est un handicap majeur pour l'exploitation des outils. Par exemple, en l'absence de graphe, l'outil `exhibitor` effectue une recherche systématique en aveugle très longue et inutilement coûteuse. Les capacités de la machine SUN [15] dont nous disposions étaient largement insuffisantes pour faire une exploration exhaustive du modèle à une profondeur nécessaire pour atteindre un interblocage. De toute manière, pour ce genre de problèmes, la plupart des mesures de complexité sont exponentielles, et l'utilisation de telle ou telle machine plus puissante ne change pas grand chose.

Il a également été impossible de comparer ce modèle à un modèle plus abstrait à l'aide de l'outil `aldebaran`, qui travaille en entrée sur des graphes complets [16].

L'objectif majeur lors de la conception du modèle, qui était déjà de réduire au maximum le nombre d'états possibles, s'est donc encore précisé. Pour arriver à ce résultat, il est important de limiter au plus l'utilisation et l'excursion des données, comme on l'a vu plus haut. Ici, les données sont les messages circulant sur le réseau, et les compteurs de crédits utilisés de part et d'autre. La première chose à effectuer, celle qui vient immédiatement à l'esprit, est de réduire la taille de la file, et donc le nombre de valeurs de compteurs. La valeur minimale de 2 a été atteinte. « 2 » est le minimum si l'on veut réserver une place pour le message CREDIT [17].

Toujours dans le même but de réduction du nombre d'états, le modèle pour décrire les paquets circulant sur le réseau a dû être conçu astucieusement : seul le nombre de crédits rendus « circulent » en réalité, les messages n'ont pas de contenu autre. Il n'y a que 2 types de messages dans le modèle : ceux qui rendent des crédits (valeur 1 à n) et ceux qui n'en rendent pas (valeur 0).

Une autre solution qui paraissait prometteuse, est celle des réseaux de LTS. L'outil `aldebaran` accepte en entrée non seulement des LTS, mais également des

15. SUN SPARC 20 / 64 Mo de RAM
16. ou de réseaux de LTS, voir plus loin
17. On peut remarquer au passage qu'il est de obligatoire de fixer de tels paramètres (longueur de files,...) quand on travaille avec ce genre d'outils. Ensuite, la généralisation des résultats obtenus à des valeurs quelconques est un problème d'expertise, et n'est plus du ressort de ces techniques formelles.

réseaux de LTS connectés selon un formalisme quasi identique à LOTOS, mais où l'on aurait supprimé la partie données et paramètres. L'intérêt de cette approche est qu'elle permet ce qu'on pourrait appeler une « compilation séparée » ce qui peut diminuer la taille des graphes. Malheureusement, le plus petit élément séparable dans notre modèle est une machine entière, car à l'intérieur d'une machine, les variables sont partagées, et l'exécution est séquentielle. L'essai de compilation d'une seule machine a tout de même été tenté. Cette opération a nécessité une modification assez profonde du code. En effet, isoler une machine signifie lever des contraintes sur les messages reçus, et donc *augmenter* l'espace d'états ! Il a donc fallu modifier le code LOTOS de façon à inclure des contrôles empêchant la réception de messages impossibles. C'est à dire quasiment simuler l'une dans l'autre.

Malgré toutes ces simplifications et différents essais de modélisation du protocole, le problème d'explosion d'états n'a pu être résolu et le graphe n'a pu être généré.

Conclusion

Ce stage effectué dans une équipe de petite taille m'a permis d'acquérir des compétences très diverses, des plus pratiques au plus théoriques. Du côté pratique, le développement d'un site web et l'administration réseau en général sont des enseignements très profitables et des atouts importants sur le marché du travail. Une petite structure oblige à être autonome et autodidacte dans de nombreux domaines, et cela est très formateur. Grâce à cela, j'ai par exemple eu la chance d'aller présenter les travaux de toute l'équipe à l'université d'Oujda au Maroc.

Ensuite, l'évaluation des performances de la plate-forme m'a fait figurer notre réseau Myrinet en tête des performances du benchmark TCP/UDP *Netperf*. L'évaluation des performances a donné lieu à une publication dans la revue *Calculateurs Parallèles*. Cette immersion dans l'univers des nouveaux calculateurs parallèles, c'est à dire les *clusters* de stations, m'a permis d'être en prise directe avec l'actualité, et donc d'acquérir une connaissance précise des performances, des coûts, et des évolutions des matériels et logiciels en pointe dans ce domaine. Cela est habituellement difficile, à cause de la rapidité des progrès, la diversité des solutions, et les effets d'annonce des médias [18].

Enfin, j'ai réalisé la spécification d'un protocole de la plate-forme en langage LOTOS, et j'ai testé la viabilité de divers outils sur ce modèle. Certains outils m'ont permis de reproduire des résultats d'interblocage, mais un problème de génération de graphe minimal a empêché l'utilisation des outils principaux. Cette démarche m'a permis de prendre connaissance et de rendre compte à l'équipe des possibilités et limitations dans ce domaine. Cette évaluation sera utile pour la suite des développements logiciels. Cet ambitieux projet de modélisation formelle m'a fait connaître l'univers et les méthodes de travail de la recherche en informatique, et m'a apporté des enseignements très théoriques, difficiles d'accès par d'autres biais.

Même si ces notions d'informatique fondamentale me sont moins utiles au jour d'aujourd'hui, la place croissante de l'informatique et de l'automatisation dans la société risque de leur donner une valeur grandissante. À condition bien sûr que se confirme la prometteuse progression de la puissance des outils afférents et de leur capacité à modéliser et résoudre des problèmes pratiques.

18. on pourra noter tout de même la une de l'hebdomadaire 01 Informatique du 14 Novembre à propos de notre plate-forme

Glossaire

ADI Abstract Device Interface

ANL Argonne National Laboratory

API Application Programming Interface

ATM Asynchronous Tranfer Mode

BIP Basic Interface for Parallelism

BSD Berkeley System Distribution

CADP Caesar/Aldebaran Development Package

CCS Calculus of Communicating Systems

CNRS Centre National pour la Recherche Scientifique

CSP Communicating Sequential Processus

CalTech California Institute of Technology

DEC Digital Equipement Compagny

DLPI Data Link Provider Interface

DMA Direct Memory Access

DOSMOS Distributed Objects Shared MemOry System

DTD Document Type Definition

E-LOTOS Extended LOTOS

ENS Lyon École Normale Supérieure de Lyon

FDDI Fiber Distributed Data Interface

HP Hewlett Packard

HTML Hyper Text Markup Langage

HiPPI High Performance Parallel Interface

INRIA Institut National de Recherche en Informatique et Automatique

INT Institut National des Télécommunications

IP Internet Protocol

ISI Information Science Institute

ISO International Standardisation Organisation

ITSEC Information Technology Security Evaluation Criteria

LAN Local Area Network

LHPC Laboratoire pour les Hautes Performances en Calcul

LIGIM Laboratoire d'Informatique Graphique Image et Modélisation

LIP Laboratoire d'Informatique du Parallélisme

LOTOS Langage Of Temporal Ordering Specification

LTS Labeled Transition System

MCP Myrinet Control Program

MPI Message Passing Interface

MPICH MPI CHameleon

NAS Numerical Aerospace Simulation

NASA National Aeronautics and Space Administration

NPB NAS Parallel Benchmarks

OSF Open Software Foundation

PC Personal Computer

PCI Peripheral Computer Interconnect

PM^2 Parallel Multithread Machines

PVM Parallel Virtual Machines

RAM Random Access Memory

RHDAC Réseaux Haut Débit et Applications Coopératives

ReMaP Régularité et Parallélisme Massif

SAN System Area Network

SGI Silicon Graphics Inc.

SPI Sciences Physiques pour l'Ingénieur

SRAM Static Random Access Memory

TCP Transport Control Protocol

UDP User Datagram Protocol

USC University of South California

VASY VAlidation de SYstèmes

VLSI Very Large Scale Integration

Bibliographie

[1] Site web de l'Équipe INRIA VASY (CADP).
http://www.inrialpes.fr/vasy/.

[2] Site web de l'équipe RHDAR. http://www-bip.univ-lyon1.fr/.

[3] Site web de Myricom inc. http://www.myri.com/.

[4] Site web des NAS parallel benchmarks.
http://science.nas.nasa.gov/Software/NPB/.

[5] Site web du LHPC. http://www.lhpc.fr/.

[6] Site web MPI de l'Argonne National Laboratory.
http://www.mcs.anl.gov/mpi.

[7] Le CERN présente le standard HiPPI sur son web, Novembre 1997.
http://www.cern.ch/HSI/hippi/.

[8] Site web de Netperf, Novembre 1997.
http://www.cup.hp.com/netperf/NetperfPage.html.

[9] Brunie, L., and Reymann, O. Integration of performance evaluation facilities into distributed shared memory based programming environments. In *TDP '96: Telecommunication Distribution Parallelism* (26-28 Juin, 1996).

[10] Felderman, R., DeSchon, A., Cohen, D., and Finn, G. Atomic: A high speed local communication architecture. *Journal of High Speed Networks, IOS Press* (1994).

[11] Garavel, H. *Compilation et vérification de programmes Lotos.* Informatique, Université Joseph Fourier - Grenoble I, 1989.

[12] Haore, C. A. R. *Processus Séquentiels Communiquants.* Masson, 1987.

[13] Herbert, M., Naquin, F., Prylli, L., Tourancheau, B., and Westrelin, R. Protocole pour le Gbit/s en réseau local, l'expérience myrinet. *Calculateurs Parallèles* (à paraître).

[14] Milner, R. *Communication and Concurrency.* Prentice Hall, 1989.

[15] Monin, J.-F. *Comprendre les Méthodes Formelles (Panorama et Outils Logiques).* Masson, 1996.

[16] Mounier, L. *Méthodes de Vérification de Spécifications Comportementales : étude et mise en œuvre*. Informatique, Université Joseph Fourier - Grenoble I, Janvier 1992.

[17] Prylli, L., and Tourancheau, B. BIP : a new protocol designed for high performance networking on myrinet. In *Workshop PC-NOW* (Orlando, USA, À paraître).

[18] Prylli, L., and Tourancheau, B. New protocol design for high performance networking. Tech. Rep. 97-22, LIP-ENS Lyon, 69364 Lyon, France, 1997.

[19] Seitz, C. L. Mosaic C: An experimental fine-grain multicomputer. Tech. rep., California Institute of Technology, 1992.

[20] Seitz, C. L., Boden, N. J., Seizovic, J., and Wen-King. The design of Caltech Mosaic C multicomputer. In *Proceedings of the University of Washington Symposium on Integrated Systems, MIT Press* (1993).

[21] Stirling, C. Modal and temporal logics for processes. *Lecture Notes in Computer Science*, 1043 (1996), 149–237.

[22] Westrelin, R. Bibliothèques pour le calcul parallèle sur réseaux haut débit : étude de myrinet. Tech. rep., LHPC-INRIA, 1997.

Annexe A

Listing LOTOS d'un modèle MPI-BIP

```
specification credits
  [ENTREEA,SORTIEA,ENTREEB,SORTIEB,FROMA2B,FROMB2A] : noexit


(* inclusion des fichiers décrivant les types *)

library  LISTTYPE, BOOLEAN, NATURAL
endlib


type LISPTYPE is LISTTYPE

   opns
      CDR : LIST -> LIST

   eqns forall L : LIST, N : MSG

      ofsort LIST

         CDR (<>)      = <> ;
         CDR ( N + L ) =  L ;

endtype


behaviour

let QUEUEA,QUEUEB : LIST = <> in

 hide FROMA2B, FROMB2A in

BOUCLE [ENTREEA,FROMA2B,SORTIEA,FROMB2A]
            (QUEUEA,          2 of  MSG, 0 of  MSG)
```

```
       |[FROMA2B, FROMB2A]|

BOUCLE [ENTREEB,FROMB2A,SORTIEB,FROMA2B]
            (QUEUEB,          2 of  MSG, 0 of  MSG)


where

process BOUCLE [SENDHIGH,SENDLOW,RECVHIGH,RECVLOW]
           ( FILE : LIST, CREDITCHEZLUI, CREDITARENDRE : MSG) : noexit :=


[CREDITCHEZLUI gt 0 of MSG] ->
(
hide INITIATIVE in
INITIATIVE ! CREDITARENDRE
  [CREDITARENDRE ge (  1 of MSG)] ;
   SENDHIGHproc[SENDHIGH,SENDLOW,RECVLOW]
        (FILE,CREDITARENDRE of MSG,CREDITCHEZLUI - 1 of MSG)
       >> accept NEWFILE : LIST, NEWCHEZLUI : MSG in
            BOUCLE [SENDHIGH,SENDLOW,RECVHIGH,RECVLOW]
                      (NEWFILE,NEWCHEZLUI,0 of MSG)
)

[]

[CREDITCHEZLUI gt 1 of MSG] ->
(
 ( SENDHIGH ? S : MSG
      [S eq DATA] ;
SENDHIGHproc[SENDHIGH,SENDLOW,RECVLOW]
   (FILE, DATA of MSG, CREDITCHEZLUI - 1 of MSG)
      >> accept NEWFILE : LIST, NEWCHEZLUI : MSG in
          BOUCLE[SENDHIGH,SENDLOW,RECVHIGH,RECVLOW]
                    (NEWFILE,NEWCHEZLUI,CREDITARENDRE)
 )
 []
 ( SENDHIGH ? S : MSG
      [S eq CTRL] ;
SENDHIGHproc[SENDHIGH,SENDLOW,RECVLOW]
     (FILE,CREDITARENDRE,CREDITCHEZLUI - 1 of MSG)
          >> accept NEWFILE : LIST, NEWCHEZLUI : MSG in
          BOUCLE[SENDHIGH,SENDLOW,RECVHIGH,RECVLOW]
                    (NEWFILE,NEWCHEZLUI,0 of MSG)
 )
)

[]

( RECVHIGH ! FIRST (FILE) [not (FIRST(FILE) eq LISTEVIDE)];
     RECVHIGHproc[RECVHIGH,RECVLOW](CDR(FILE),CREDITCHEZLUI)
    >> accept NEWFILE : LIST , NEWCHEZLUI : MSG in
           BOUCLE [SENDHIGH,SENDLOW,RECVHIGH,RECVLOW]
                 (NEWFILE,NEWCHEZLUI,CREDITARENDRE + 1 of MSG) )
```

```
[]

( RECVLOWproc [RECVLOW] (FILE)
          >> accept NEWFILE : LIST , RENDUS : MSG in
           BOUCLE [SENDHIGH,SENDLOW,RECVHIGH,RECVLOW]
                 (NEWFILE,CREDITCHEZLUI+RENDUS,CREDITARENDRE)
)

endproc



process SENDHIGHproc [SENDHIGH,SENDLOW,RECVLOW]
               (FILE : LIST, M : MSG, CHEZLUI: MSG): exit (LIST,MSG):=

        SENDLOW ! M ; exit(FILE,CHEZLUI)
        []
        (  RECVLOWproc [RECVLOW] (FILE)
         >> accept NEWFILE : LIST , RENDUS : MSG in
         SENDHIGHproc [SENDHIGH,SENDLOW,RECVLOW](NEWFILE,M, CHEZLUI + RENDUS)
        )

endproc


process RECVHIGHproc [RECVHIGH,RECVLOW]
                (FILE : LIST, CHEZLUI : MSG): exit(LIST,MSG) :=

          exit(FILE,CHEZLUI)
         []
       (  RECVLOWproc [RECVLOW] (FILE)
                    >> accept NEWFILE : LIST , RENDUS : MSG in
          RECVHIGHproc [RECVHIGH,RECVLOW](NEWFILE, CHEZLUI + RENDUS)
       )

endproc


process RECVLOWproc [ENTREE](FILE : LIST): exit(LIST,MSG) :=

       ENTREE ? E : MSG
         [ (LEN(FILE) lt (          2 of Nat))];
            exit(REVERSE(E + REVERSE (FILE)), E )

endproc


endspec
```